

monocular RGB camera

7 DoF robotic manipulator

2-finger gripper

object bin



Option 1:

Understand the problem, design a solution



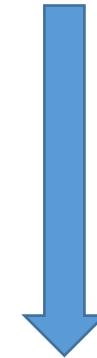
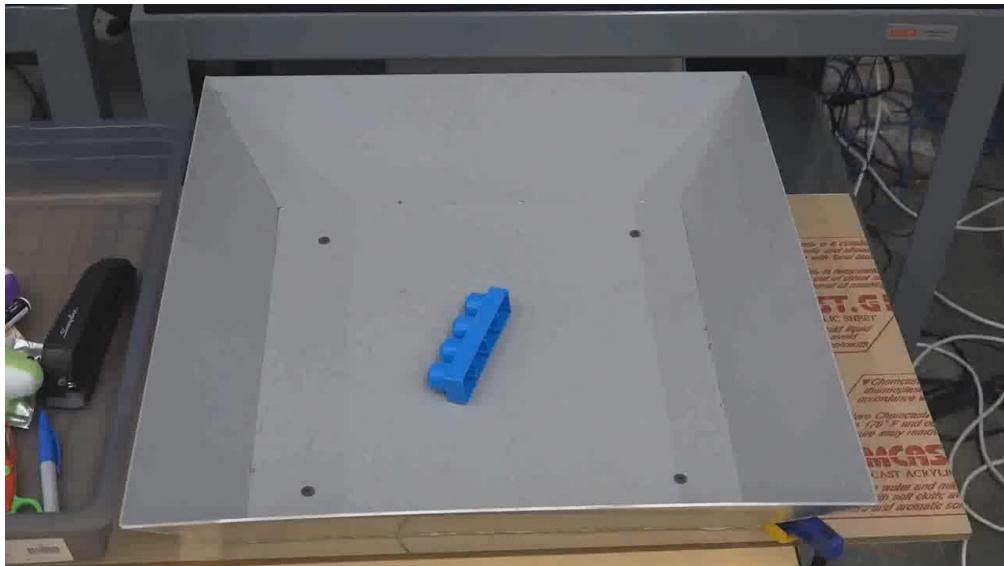
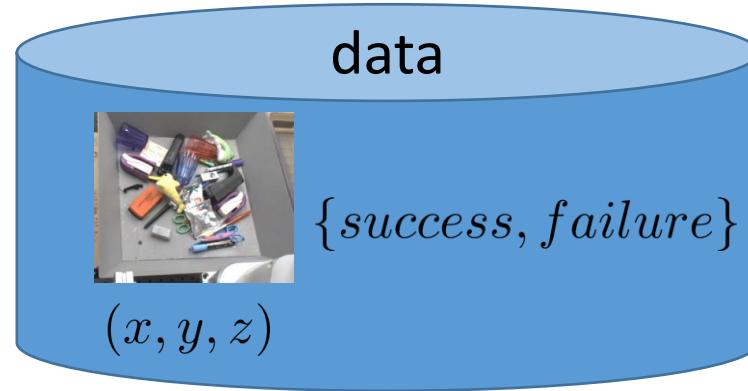
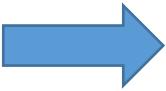
Option 2:

Set it up as a machine learning problem



supervised learning





What are some recent advances we've seen in AI?



vibrant portrait painting of Salvador Dalí with a robotic half face



a shiba inu wearing a beret and black turtleneck



a close up of a handpalm with leaves growing from it



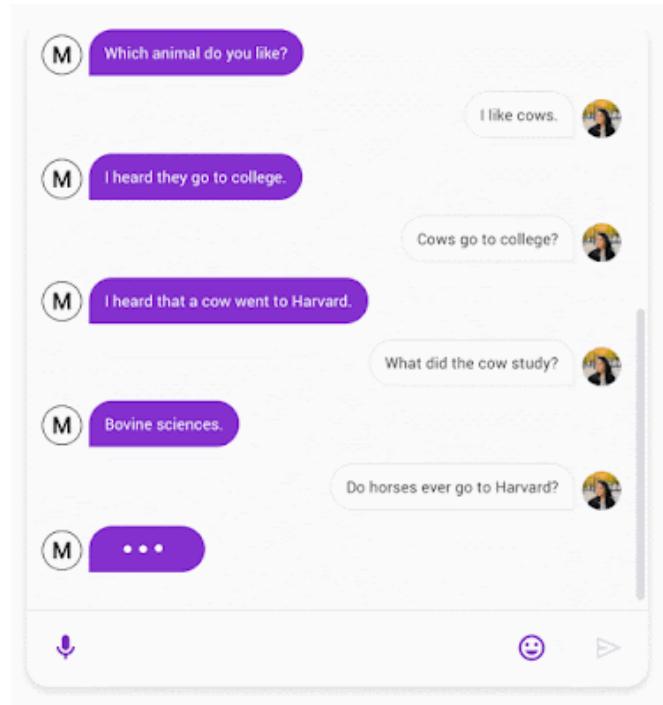
an espresso machine that makes coffee from human souls, artstation



panda mad scientist mixing sparkling chemicals, artstation



a corgi's head depicted as an explosion of a nebula



Explaining a joke

Prompt

Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!

Model Response

Prediction: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

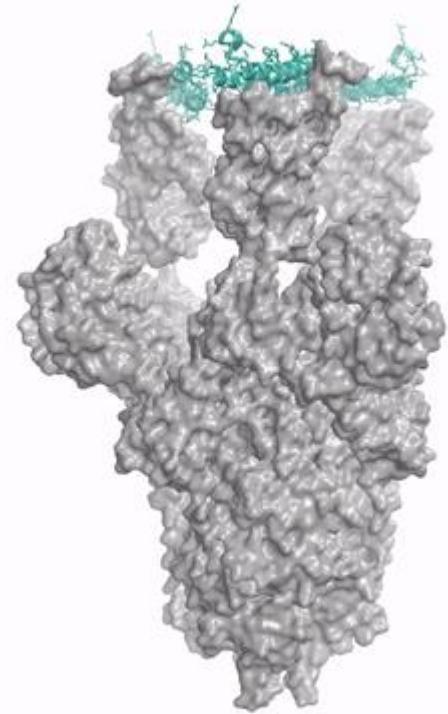
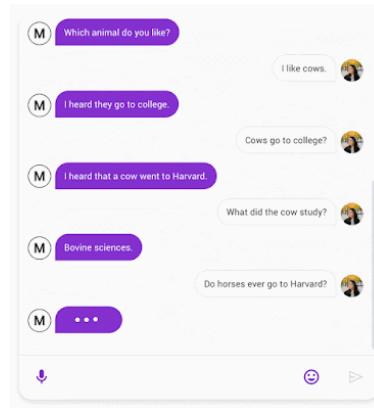


image credit: UW IPD

What's the main idea behind this?



Explaining a joke

Prompt

Explain this joke:

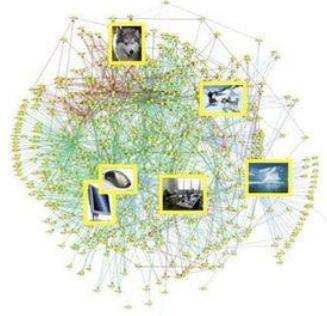
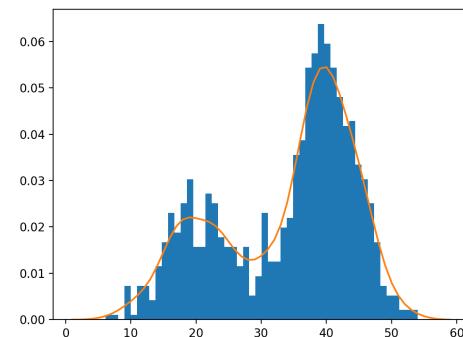
Joke: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!

Model Response

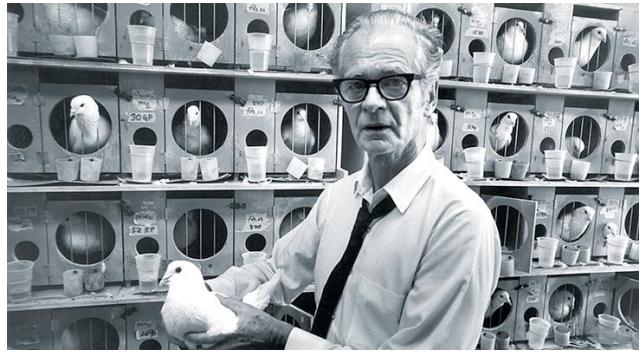
Prediction: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

$$p_{\theta}(\mathbf{x})$$

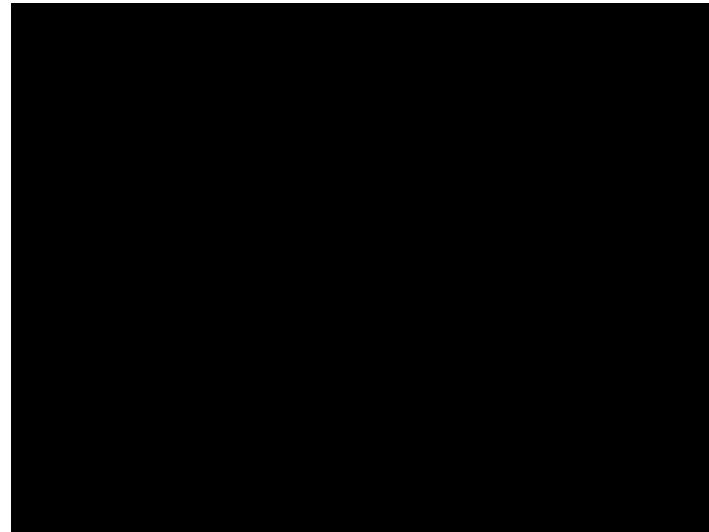
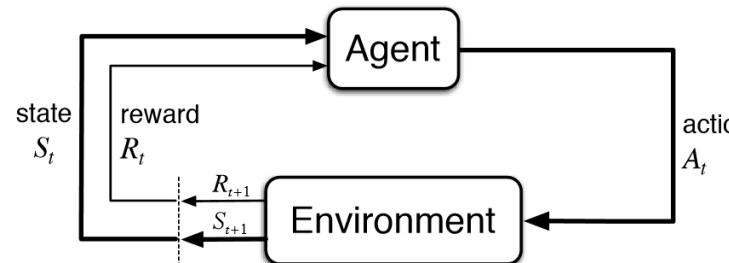
$$p_{\theta}(\mathbf{y}|\mathbf{x})$$



What does reinforcement learning do differently?



classical reinforcement learning



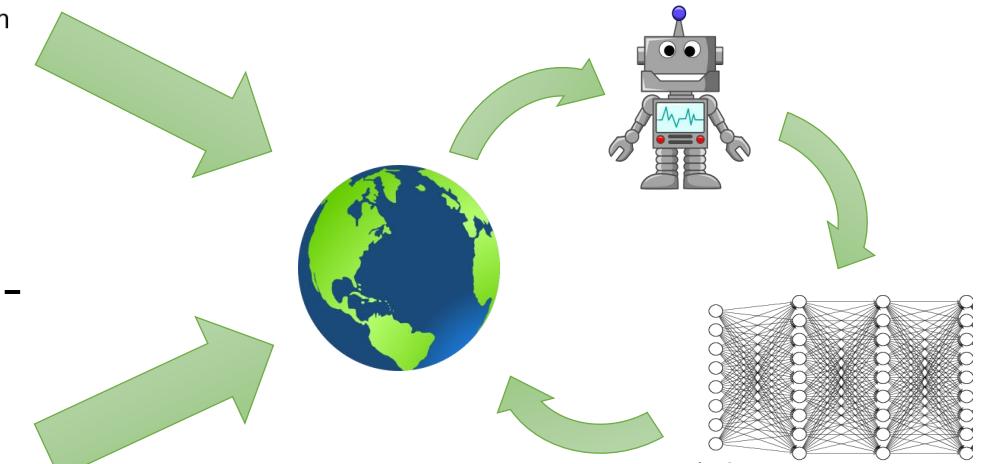
Evolved Virtual Creatures.
Karl Sims, 1994

Synthesis of Complex Behaviors
with
Online Trajectory Optimization

Yuval Tassa, Tom Erez & Emo Todorov

IEEE International Conference
on Intelligent Robots and Systems
2012

Model-Predictive Control with iLQG.
Yuval Tassa, 2012

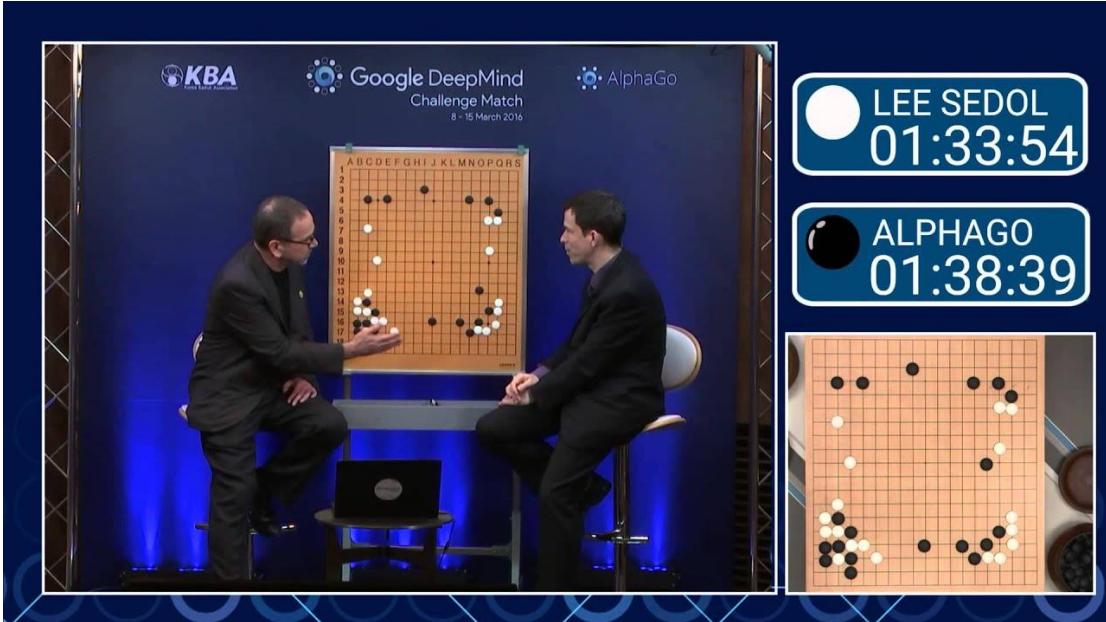


deep reinforcement learning



Reinforcement learning can discover new solutions

Impressive because no person had thought of it!



“Move 37” in Lee Sedol AlphaGo match: reinforcement learning “discovers” a move that surprises everyone

Impressive because it looks like something a person might draw!



Deep Reinforcement Learning, Decision Making, and Control

CS 285

Instructor: Sergey Levine
UC Berkeley



Course logistics

Class Information & Resources



Sergey Levine
Instructor



Kyle Stachowicz
Head GSI



Joey Hong
GSI



Vivek Myers
GSI



Kevin Black
GSI

- Course website: <http://rail.eecs.berkeley.edu/deeprlcourse>
- Ed: CS285 Deep Reinforcement Learning: <https://edstem.org/us/join/c9XPxK>
- Gradescope: CS285 Deep Reinforcement Learning: <https://www.gradescope.com/courses/571673>
- Office hours: check course website, mine are after class right here (starting next week)
- Links to lecture videos, etc.: always posted on Ed (pinned post)

Prerequisites & Enrollment

- All enrolled students must have taken CS182, CS189, CS289, CS281A, CS282, or an equivalent course at your home institution
 - Please contact Sergey Levine if you haven't
- If you are not eligible to enroll directly into the class, **fill out the enrollment application form (do this today):**
<http://rail.eecs.berkeley.edu/deeprlcourse/>
 - We will enroll subject to availability based on responses to this form
 - We will not use the official CalCentral wait list!
 - **Fill out an application before the end of this week (8/25/23)!**

Class format

- **Lectures**

- Lectures will be posted over the weekend on **YouTube**
- The YouTube playlist link will be posted on **Ed** and on the course website:
 - https://youtube.com/playlist?list=PL_iWQOsE6TfVYGEGiAOMaOzzv41Jfm_Ps
- Some lectures from past years, some new
- **Watch the lectures in advance**
- **Post questions on Ed thread**
- **Upvote questions on Ed thread**
- **Every lecture has a quiz**

Deep RL: CS 285 Fall 2021

25/150 CANCEL SAVE

4 videos • No views • Updated today

Public ▾

Lectures for UC Berkeley CS 285: Deep Reinforcement Learning for Fall 2021

R RAIL



0 Comments SORT BY

Add a public comment...

SUBSCRIBE

Class format

- **Every lecture has a quiz**
 - Complete on Gradescope prior to class
 - Due at 5 pm on the day of the corresponding lecture
 - Except today's quiz, which is due 5 pm next wk Mon
 - Should never take more than 5-10 min if you watched the lecture and understood it
 - Check your understanding, lightweight & quick
 - Also serves to summarize the key concepts you should know
- **If you don't like your grade you can actually take it again (second time)!**
 - We release your grades automatically on the due date at 5 pm
 - We release a “second try” quiz that you can take if you don't like your grade (within 48 hours)
 - Answers will be released with “second try” quiz

Lecture 1 Quiz

STUDENT NAME
Search students by name or email...

Q1
1 Point
Which of the following are true for RL, but not supervised learning? Check all that apply

In RL, sequential inputs are not independent and identically distributed

In RL, we cannot use deep neural networks

In RL, the task is not simply to match the ground truth label

In RL, we don't have any access to a loss, reward, or objective signal of any sort

Save Answer

Q2
1 Point
Which of the following could be framed as reinforcement learning problems? Check all that apply

buying and selling inventory for a warehousing application

controlling a robot to run at a desired speed

controlling a vehicle to regulate the flow of traffic

Class format

- **In class**
 - **Attend class live:** meant to be interactive
 - **Discuss questions:** submit & upvote Ed questions to have a fun class!
 - **More in-depth derivations:** we'll go through some whiteboard derivations
 - **Participate and ask questions:** plenty of time for Q&A, bring your questions!
 - **Length:** Class will be about 50 min, followed by about 30 min of OH, since you are already watching 50-70 minutes of online lecture
- **Office hours:** My office hours will take place immediately after class
 - As soon as we cover all discussion, we will switch to “office hours” – use this time to ask questions about final projects, etc.
 - Exception is today, since we have a “classic” lecture, and presumably you don’t have many OH questions on first day of class :)

What you should know

- Assignments will require training neural networks with standard automatic differentiation packages (PyTorch)
- Review Section
 - We will have a review section to cover PyTorch next week on Wednesday
 - Optional, no quiz, but very useful if you are unfamiliar with it!
 - If you are unsure about background, try HW1 as soon as it comes out (next Monday) and if you're having trouble, come to the review section

Course content

What we'll cover

1. From supervised learning to decision making
2. Model-free algorithms: Q-learning, policy gradients, actor-critic
3. Model-based algorithms: planning, sequence models, etc.
4. Exploration
5. Offline reinforcement learning
6. Inverse reinforcement learning
7. Advanced topics, research talks, and invited lectures

Assignments

1. Homework 1: Imitation learning (control via supervised learning)
2. Homework 2: Policy gradients
3. Homework 3: Q-learning and actor-critic algorithms
4. Homework 4: Model-based reinforcement learning
5. Homework 5: Offline reinforcement learning
6. Final project: Research-level project of your choice (form a group of up to 2-3 students, you're welcome to start early!)

Grading: 50% homework (10% each), 40% project, 10% quizzes

5 late days total for homeworks

(does **not** apply to quizzes, proposal, milestone report, or final project report)

Your “Homework” Today

1. Make sure you are signed up for Ed (UC Berkeley CS285)
2. Start forming your final project groups, unless you want to work alone, which is fine
3. Take the lecture 1 quiz
 - it should be super quick if you watched lecture 1, mostly to familiarize yourself with Gradescope interface

What is reinforcement learning?

What is reinforcement learning?

Mathematical formalism for learning-based decision making

Approach for learning decision making and control from experience

How is this different from other machine learning topics?

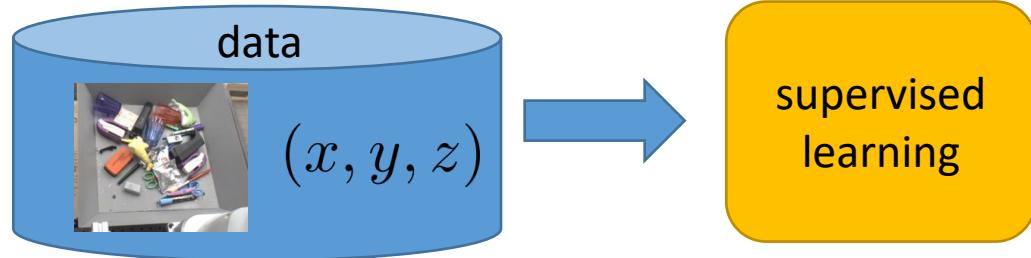
Standard (supervised) machine learning:

given $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$

learn to predict y from \mathbf{x} $f(\mathbf{x}) \approx y$

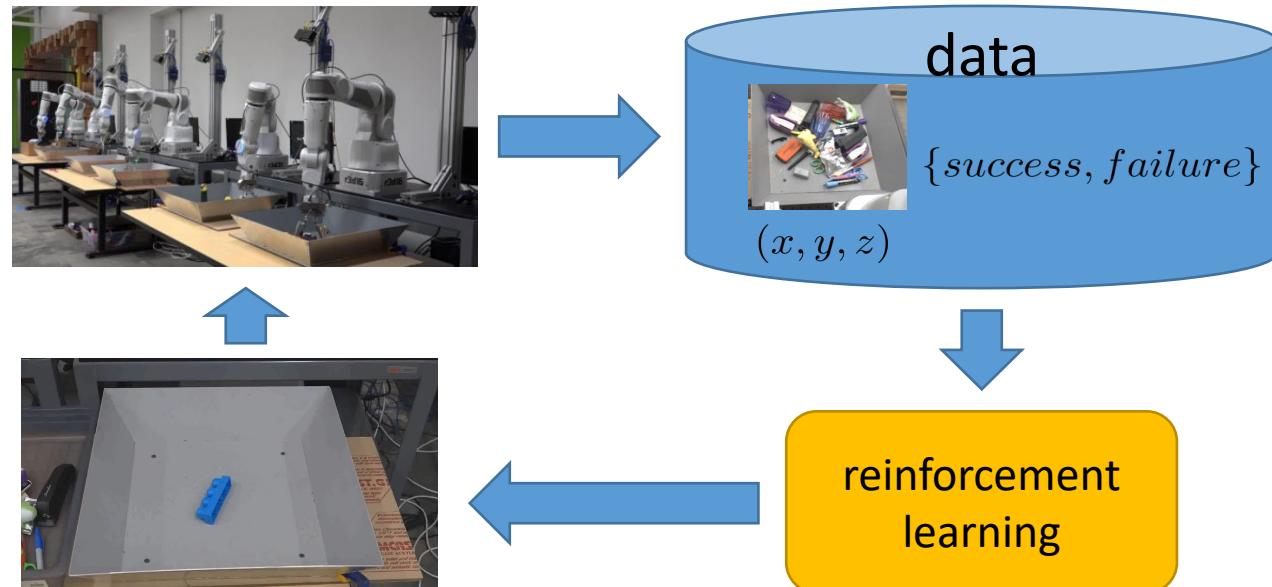
Usually assumes:

- i.i.d. data
- known ground truth outputs in training



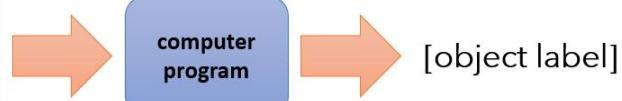
Reinforcement learning:

- Data is **not** i.i.d.: previous outputs influence future inputs!
- Ground truth answer is not known, only know if we succeeded or failed
 - more generally, we know the reward



What is reinforcement learning?

supervised learning



input: \mathbf{x}

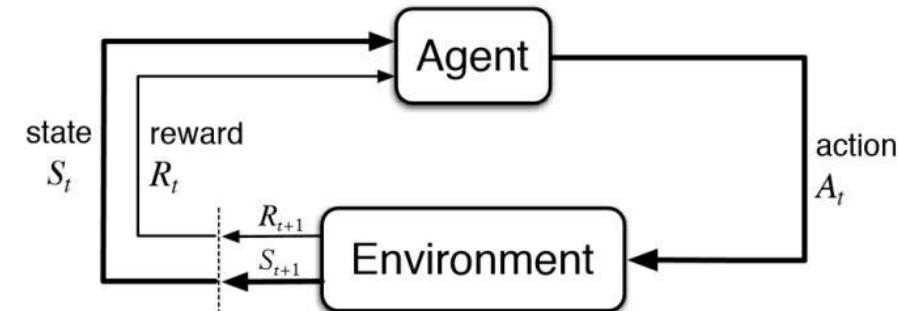
output: \mathbf{y}

data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

goal: $f_\theta(\mathbf{x}_i) \approx \mathbf{y}_i$

someone gives
this to you

reinforcement learning



pick your
own actions

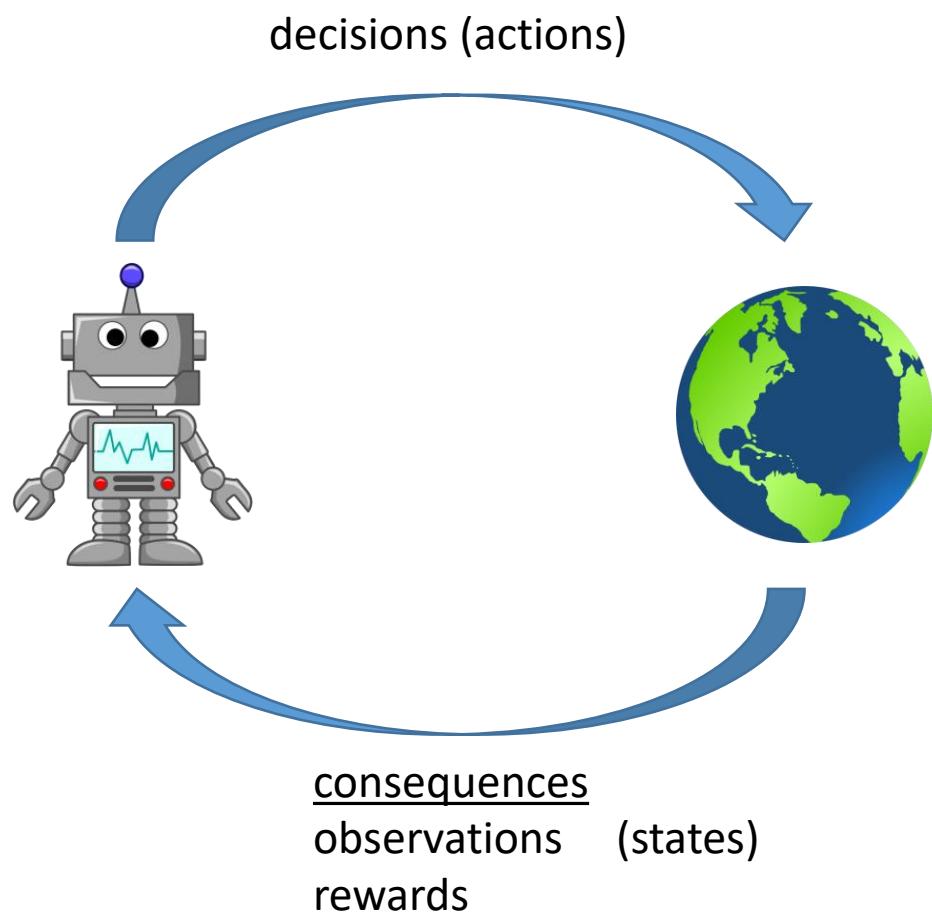
input: s_t at each time step

output: a_t at each time step

data: $(\mathbf{s}_1, \mathbf{a}_1, r_1, \dots, \mathbf{s}_T, \mathbf{a}_T, r_T)$

goal: learn $\pi_\theta : \mathbf{s}_t \rightarrow \mathbf{a}_t$

to maximize $\sum_t r_t$



Actions: muscle contractions
 Observations: sight, smell
 Rewards: food

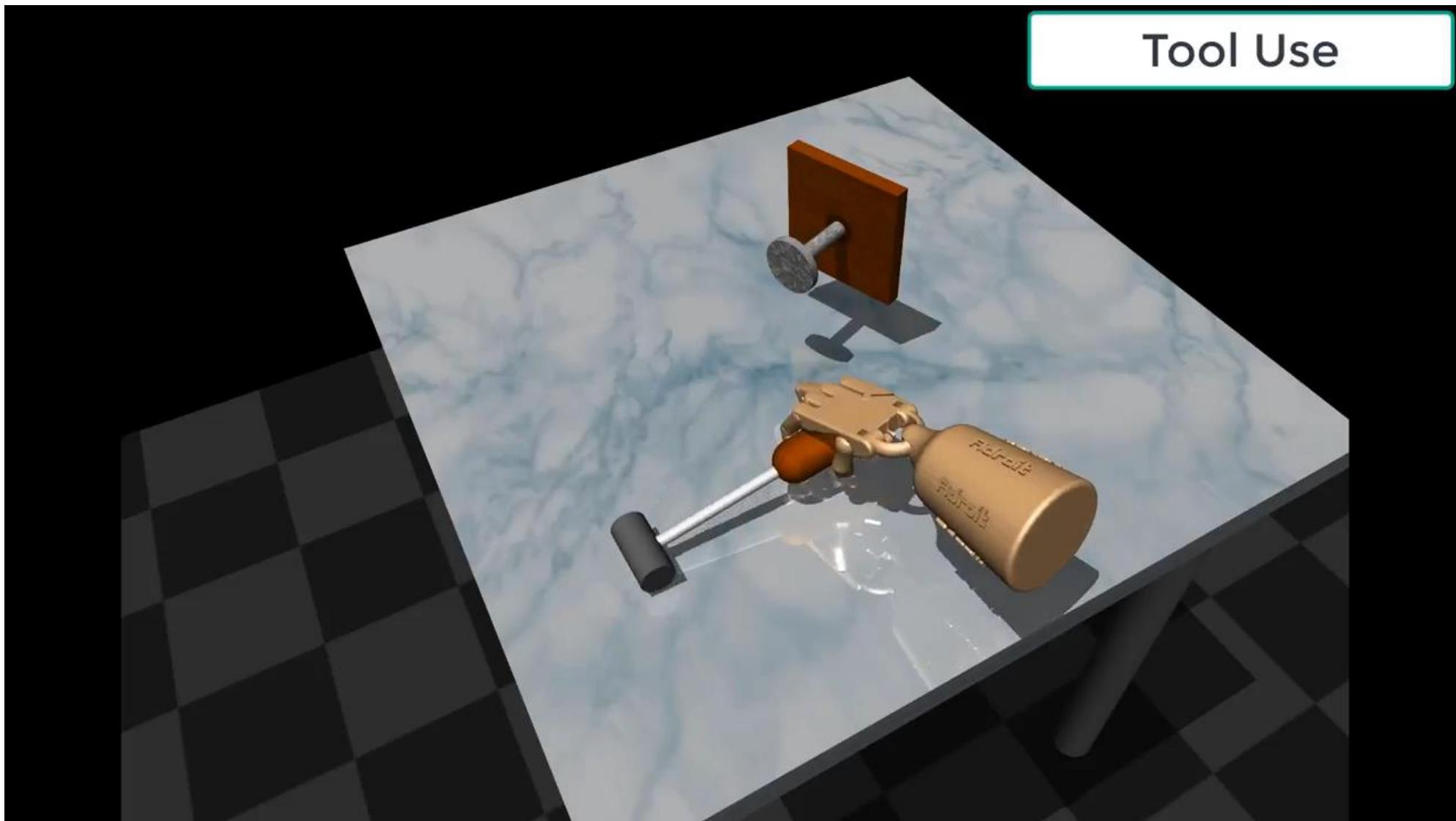


Actions: motor current or torque
 Observations: camera images
 Rewards: task success measure (e.g., running speed)



Actions: what to purchase
 Observations: inventory levels
 Rewards: profit

Complex physical tasks



Really complex physical tasks!



Learning and Adapting Agile Locomotion Skills



lauramsmith.github.io

<https://sites.google.com/berkeley.edu/twirl>

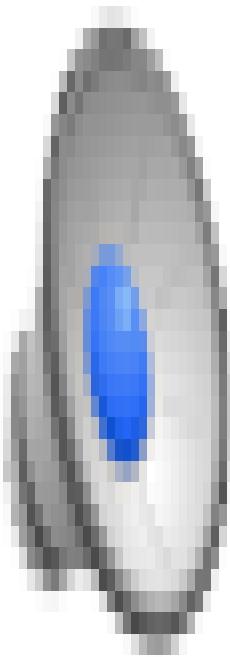
Really really complex physical tasks!

Advanced Skills by Learning Locomotion and Local Navigation End-to-End

Nikita Rudin, David Hoeller,
Marko Bjelonic and Marco Hutter



Unexpected solutions



At scale in the real world

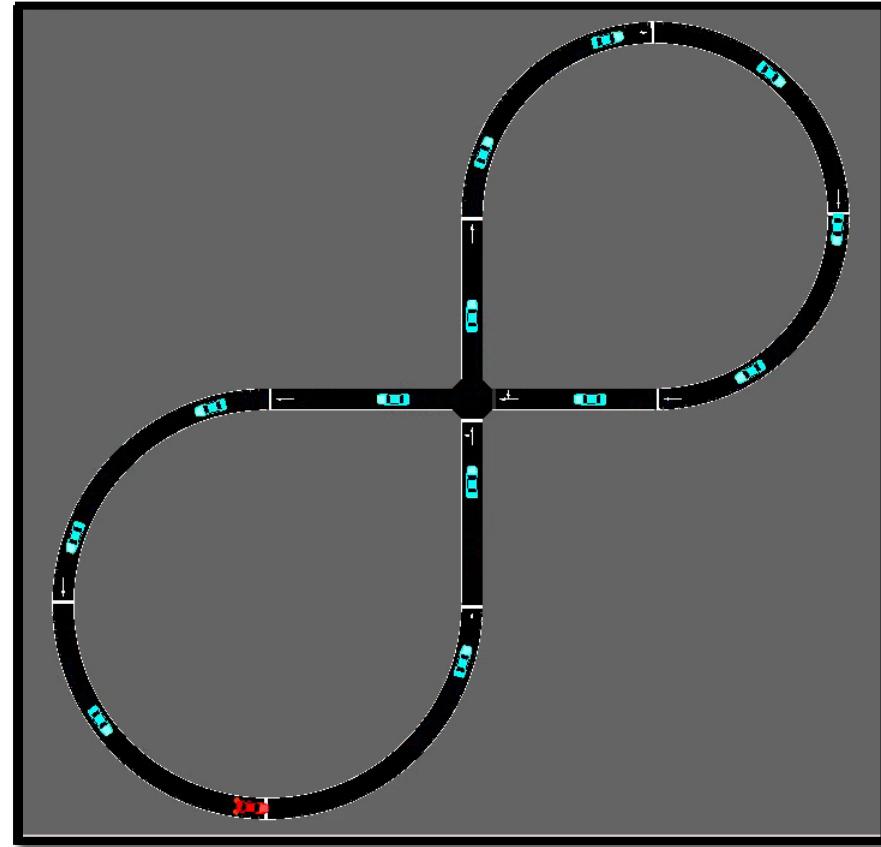
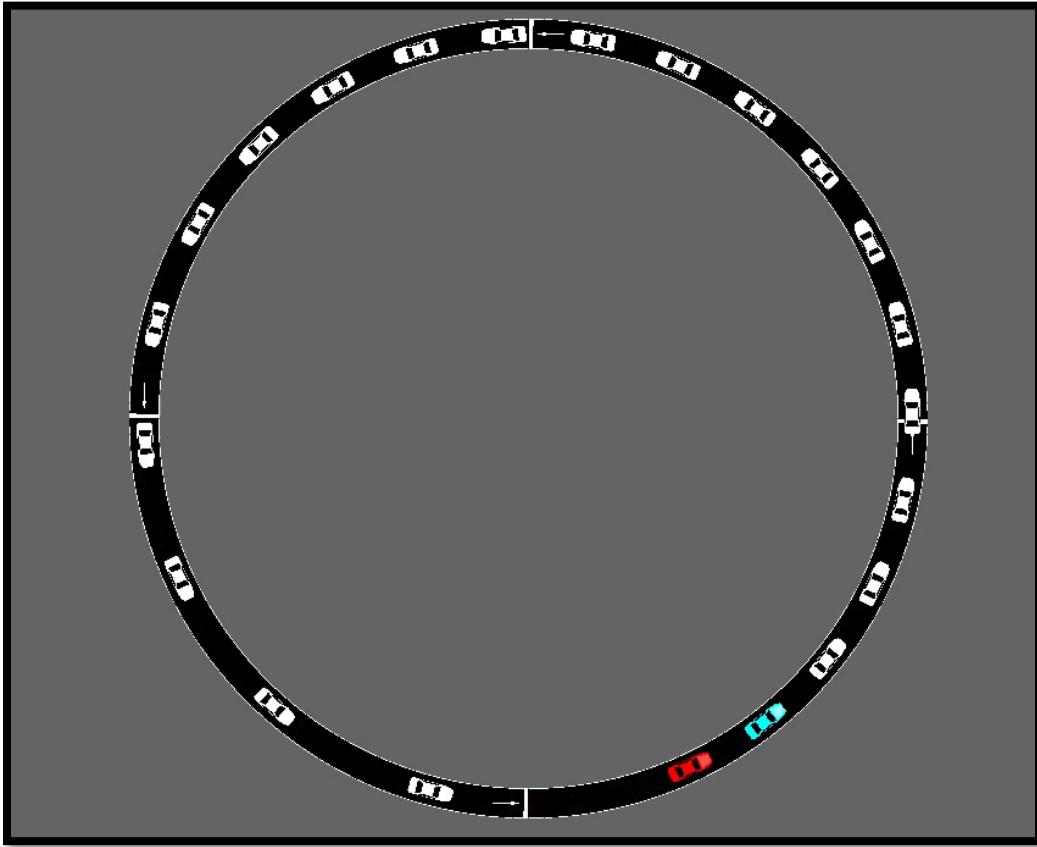


**Deep RL at Scale: Sorting Waste in Office Buildings with
a Fleet of Mobile Manipulators**



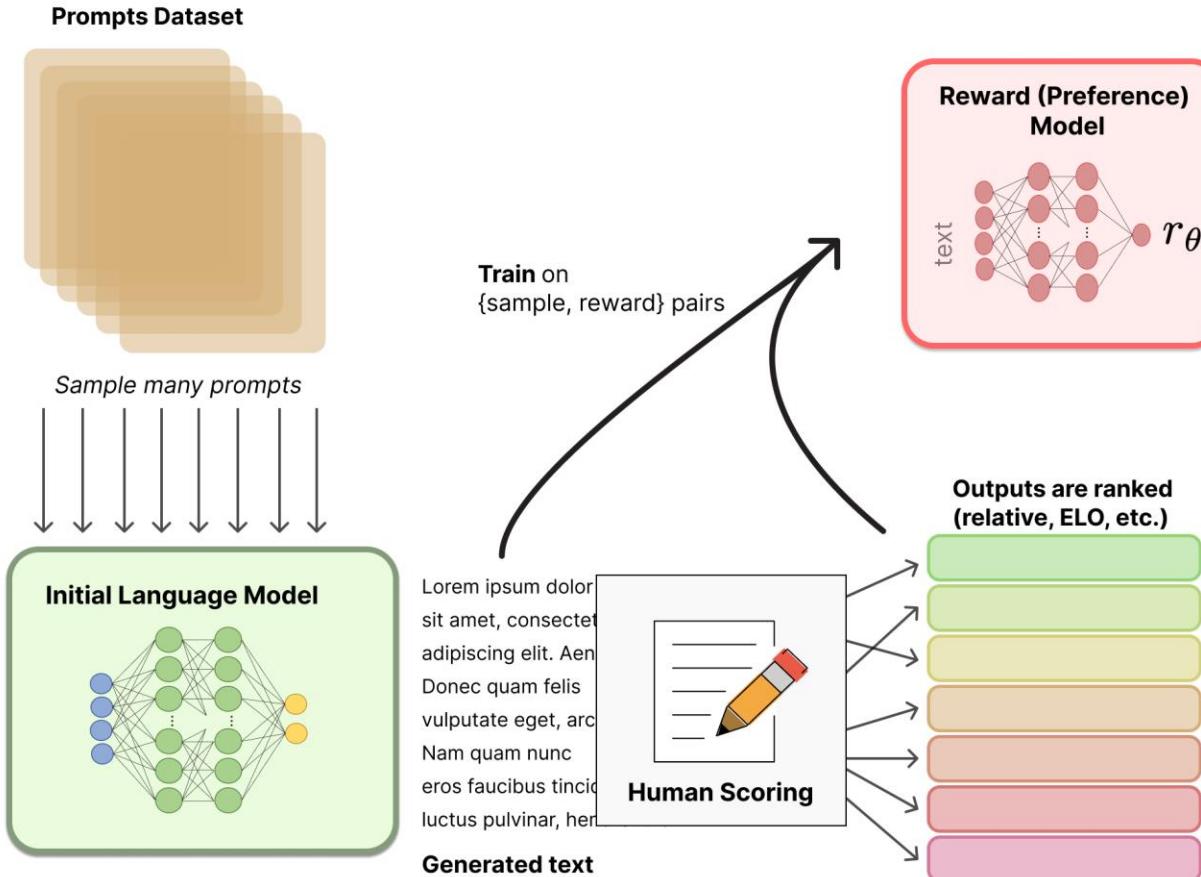
**Deep RL at Scale: Sorting Waste in Office Buildings with
a Fleet of Mobile Manipulators**

Not just games and robots!



Cathy Wu

Reinforcement learning with language models



LE Can you explain to me how RL with human feedback works for language models like I am five?

Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

First, Rufus will say something using his language model. For example, he might say "I am a robot."

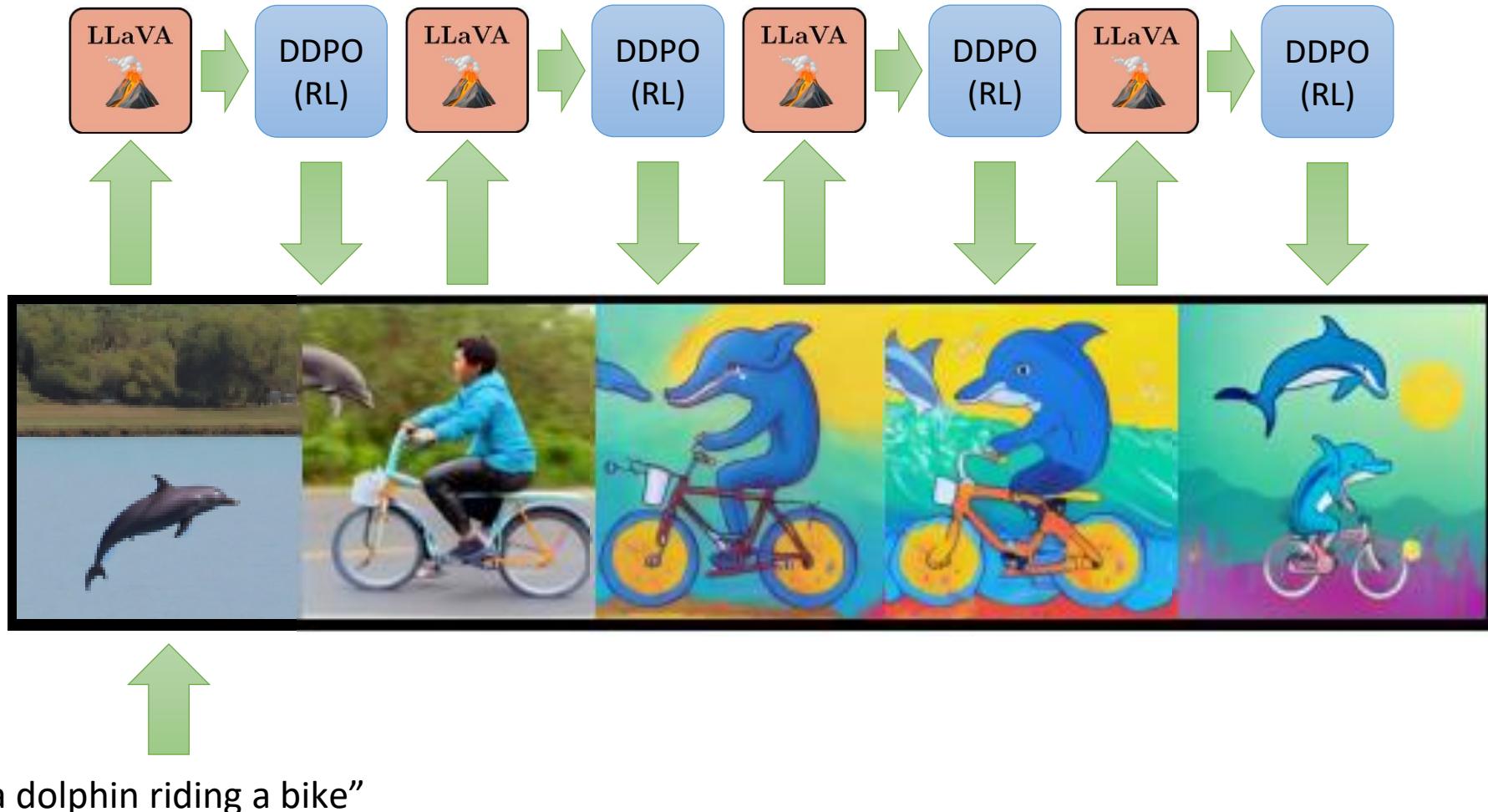
Then, a human will listen to what Rufus said and give him feedback on whether it sounded like a natural sentence a human would say. The human might say, "That's not quite right, Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I am a machine.'"

Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

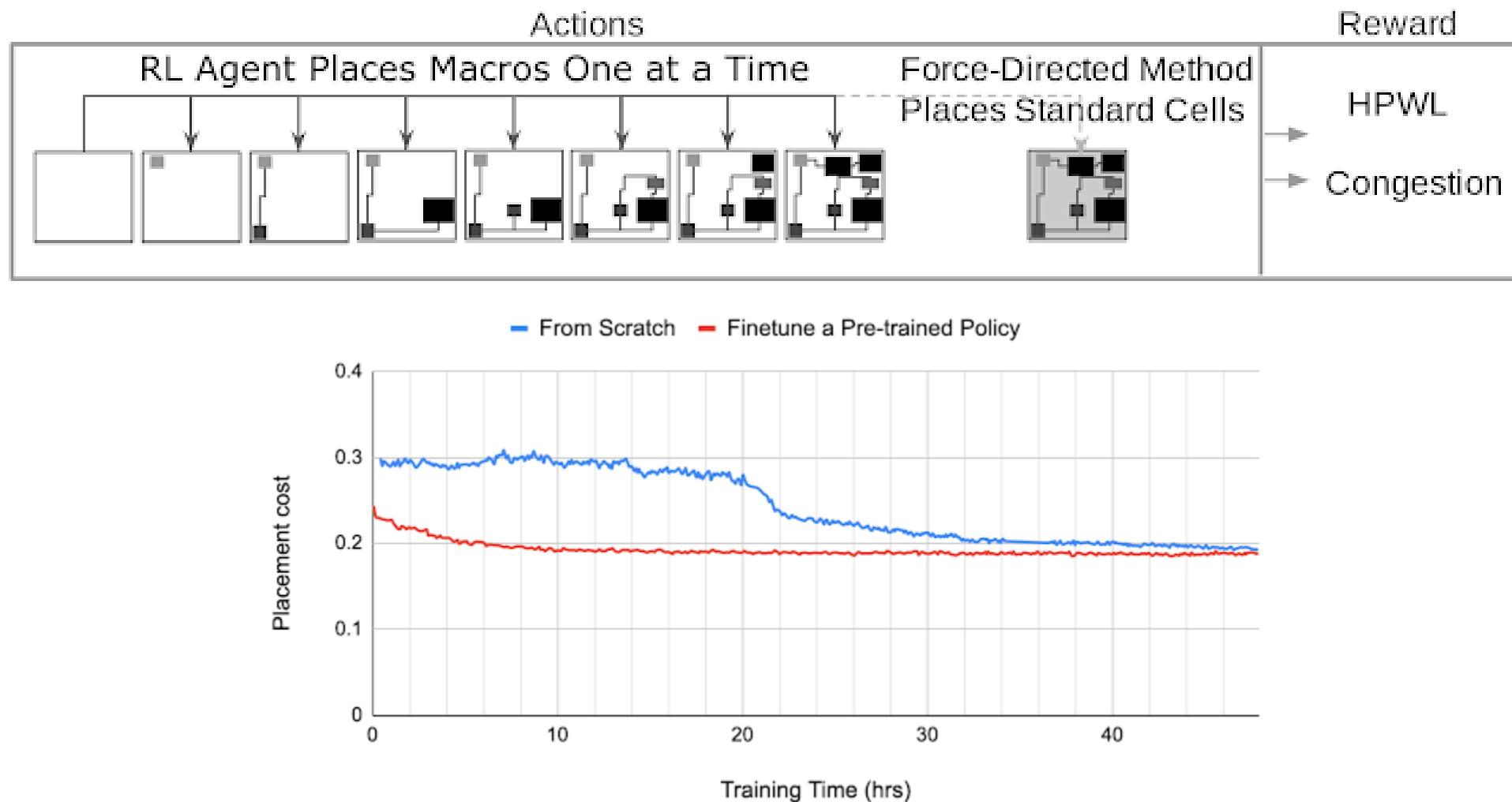
The human will listen again and give Rufus more feedback. This process will continue until Rufus can say sentences that sound natural to a human.

Over time, Rufus will learn how to talk like a human thanks to the feedback he receives from humans. This is how language models can be improved using RL with human feedback.

Reinforcement learning with image generation



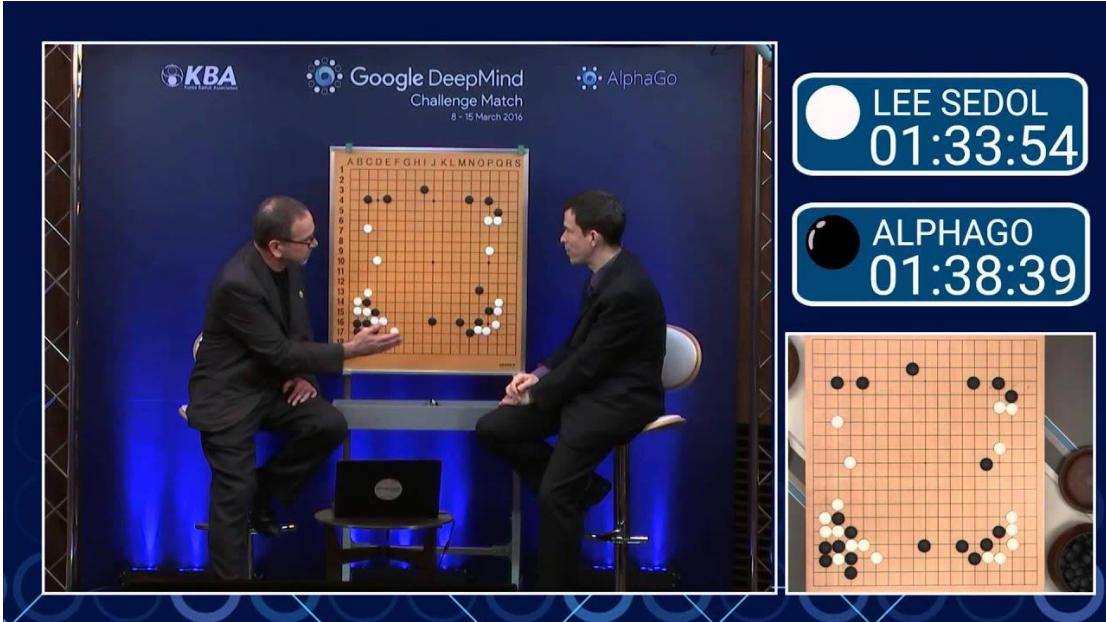
Reinforcement learning for chip design



Why should we study
deep reinforcement learning?

Reinforcement learning can discover new solutions

Impressive because no person had thought of it!

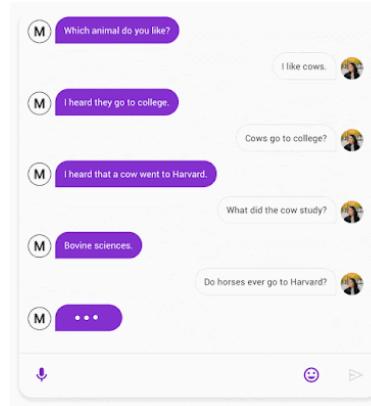


“Move 37” in Lee Sedol AlphaGo match: reinforcement learning “discovers” a move that surprises everyone

Impressive because it looks like something a person might draw!



How does data-driven AI work?



Explaining a joke

Prompt

Explain this joke:

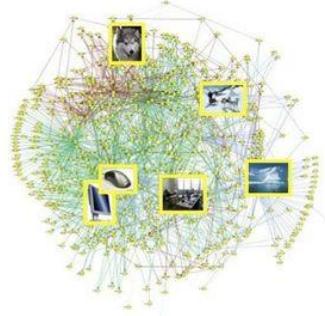
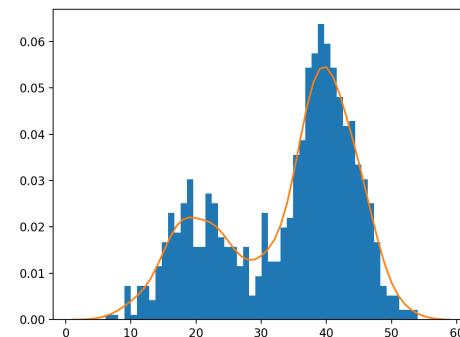
Joke: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!

Model Response

Prediction: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

$$p_{\theta}(\mathbf{x})$$

$$p_{\theta}(\mathbf{y}|\mathbf{x})$$



So where does that leave us?

Data-Driven AI



Reinforcement Learning



- + learns about the real world from data
- doesn't try to do **better** than the data

- + optimizes a goal with emergent behavior
- but need to figure out how to use at scale!

Data without optimization
doesn't allow us to solve new problems in new ways

A bitter but misunderstood lesson



Richard Sutton

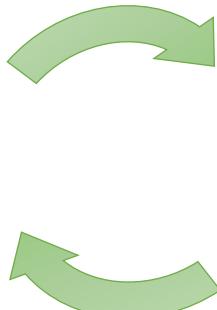
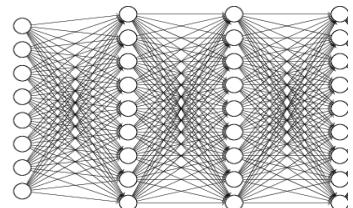
We have to learn the bitter lesson that building in how we think we think does not work in the long run.

The two methods that seem to scale arbitrarily are *learning* and *search*

Data without optimization
doesn't allow us to solve new problems in new ways

Learning

use **data** to extract **patterns**



allows us to **understand** the world

Search

use **computation** to extract **inferences**

optimization

leverages that **understanding** for **emergence**



xkcd

A bit of philosophy

Why do we need **machine learning** anyway?

A postulate:

We need machine learning for one reason and one reason only – that's to produce **adaptable and complex decisions**.



Decision: how do I move my joints?



Decision: how do I steer the car?



What is the decision? The image label?

What happens with that label afterwards?

Aside: why do we need **brains** anyway?



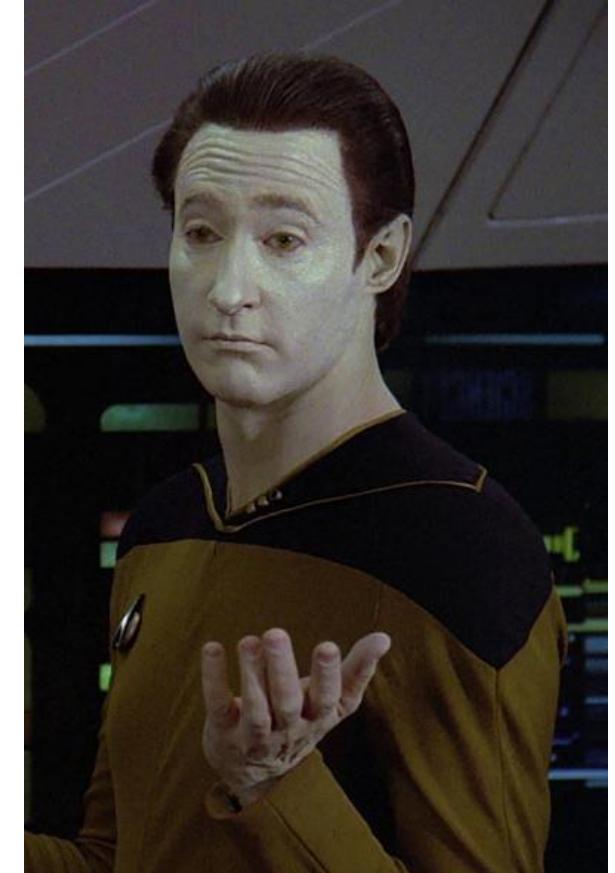
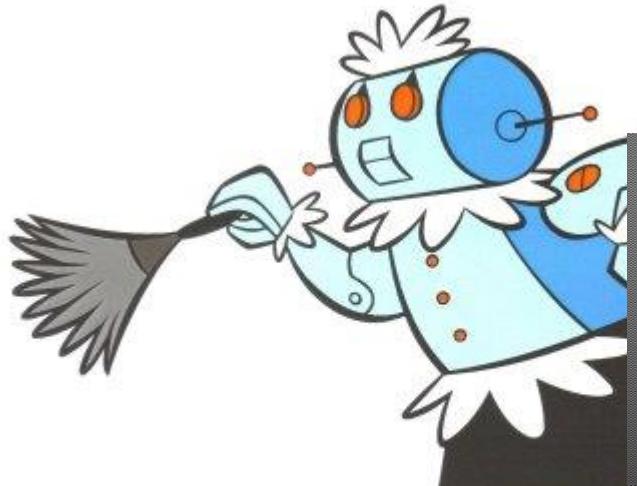
Daniel Wolpert

(knows quite a lot about brains)



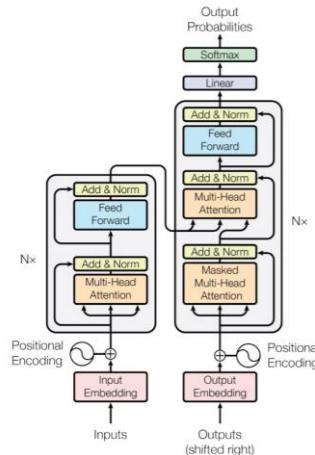
"We have a brain for one reason and one reason only -- that's to produce **adaptable and complex movements**. Movement is the only way we have affecting the world around us... I believe that to understand movement is to understand the whole brain."

How do we build intelligent machines?

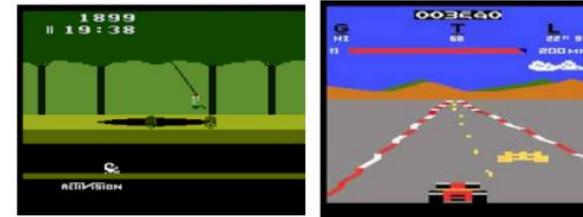


Why should we study this now?

big end-to-end trained models work quite well!



we have RL algorithms that we can feasibly combine with deep networks



and yet learning-based control in truly real-world settings remains a major open problem!

What other problems do we need to solve to enable real-world sequential decision making?

Beyond learning from reward

- Basic reinforcement learning deals with maximizing rewards
- This is not the only problem that matters for sequential decision making!
- We will cover more advanced topics
 - Learning reward functions from example (inverse reinforcement learning)
 - Transferring knowledge between domains (transfer learning, meta-learning)
 - Learning to predict and using prediction to act

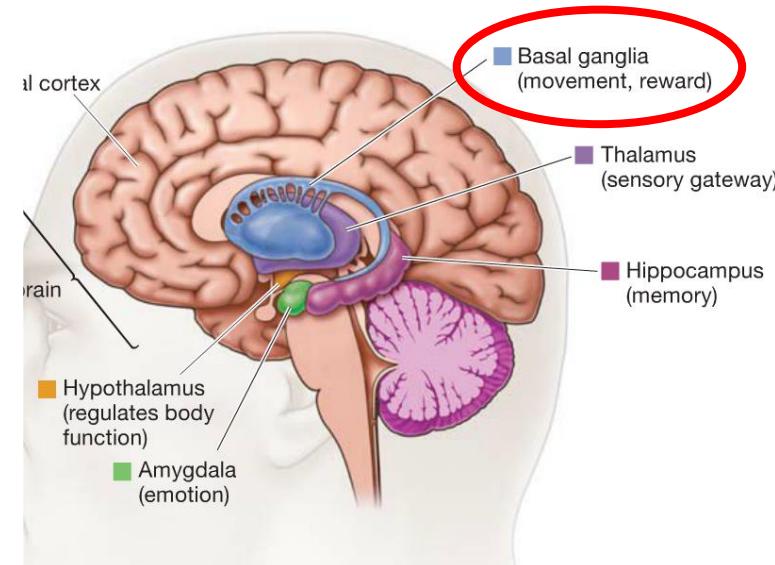
Where do rewards come from?



reinforcement learning agent

[-] LazyOptimist 32 points 5 days ago

As human agents, we are accustomed to operating with rewards that are so sparse that we only experience them once or twice in a lifetime, if at all.



Are there other forms of supervision?

- Learning from demonstrations
 - Directly copying observed behavior
 - Inferring rewards from observed behavior (inverse reinforcement learning)
- Learning from observing the world
 - Learning to predict
 - Unsupervised learning
- Learning from other tasks
 - Transfer learning
 - Meta-learning: learning to learn

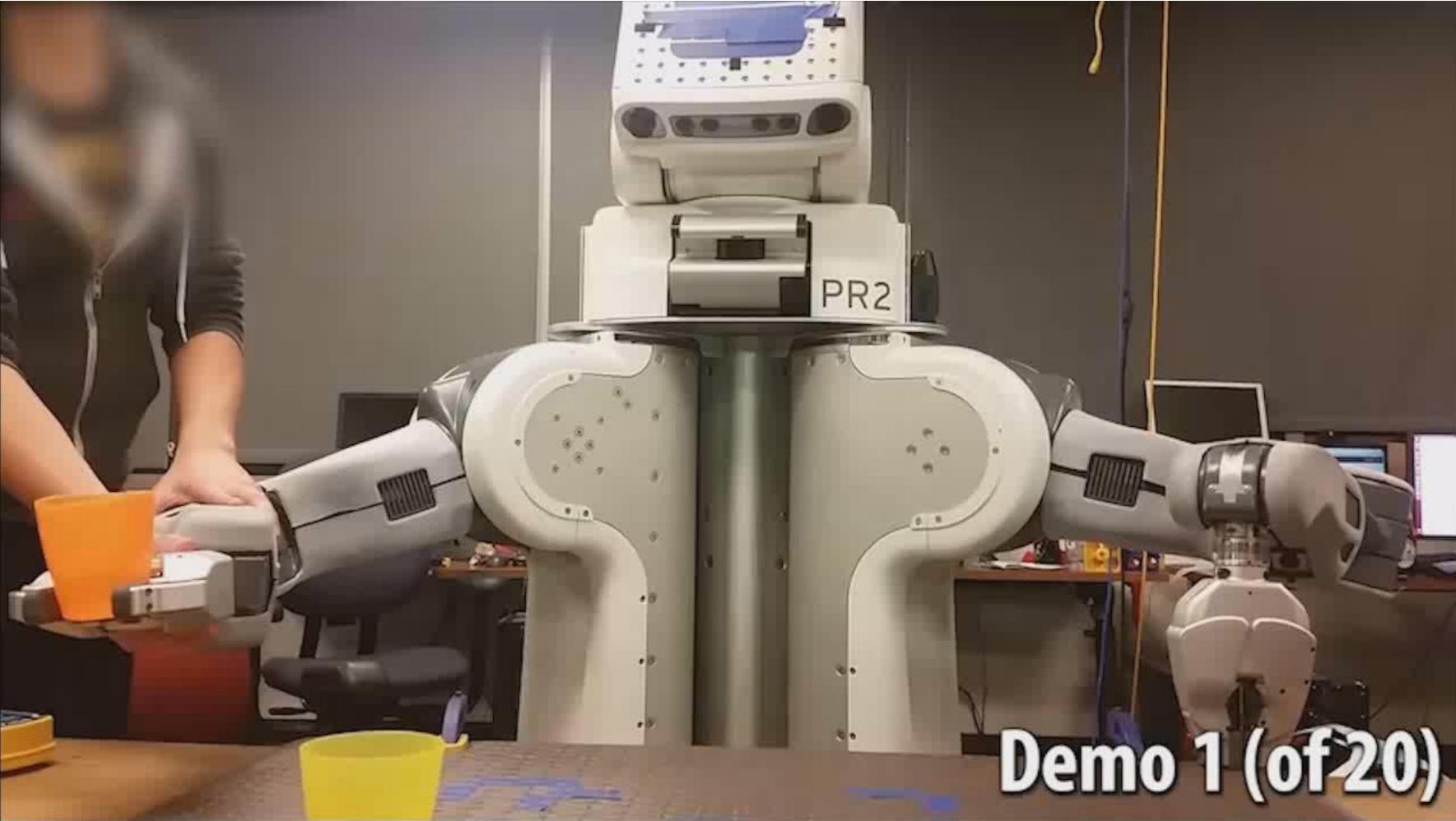
Imitation learning



More than imitation: inferring intentions



Inverse RL examples



Prediction

“the idea that we **predict the consequences of our motor commands** has emerged as an important theoretical concept in all aspects of sensorimotor control”

Prediction Precedes Control in Motor Learning

J. Randall Flanagan,^{1,*} Philipp Vetter,²
Roland S. Johansson,² and Daniel M. Wolpert²

Procedures for details). Figure 1 shows, for a single subject, the hand path (top trace) and the grip (middle)

Predicting the Consequences of Our Own Actions: The Role of Sensorimotor Context Estimation

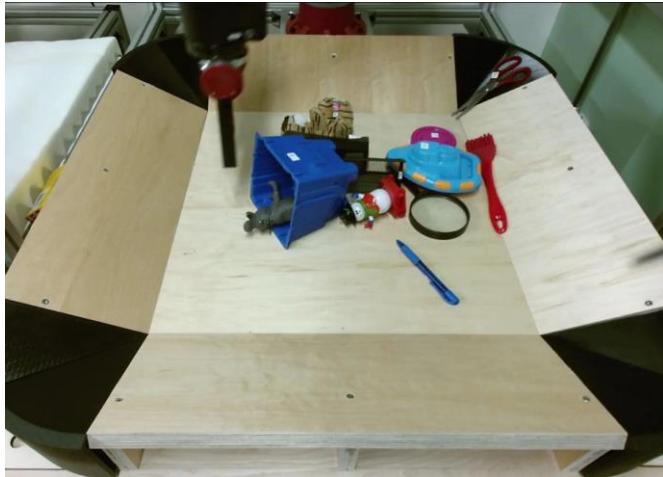
Sarah J. Blakemore, Susan J. Goodbody, and Daniel M. Wolpert

Sobell Department of Neurophysiology, Institute of Neurology, University College London, London WC1N 3BG,

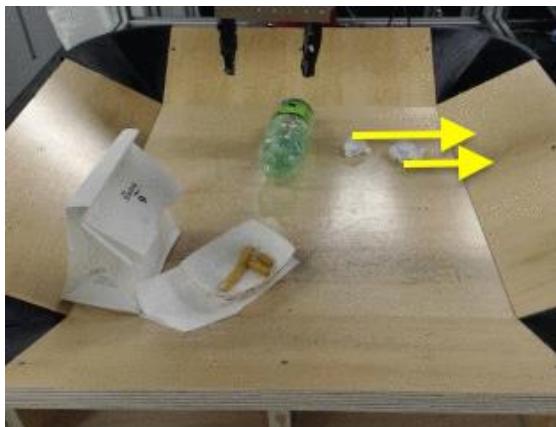
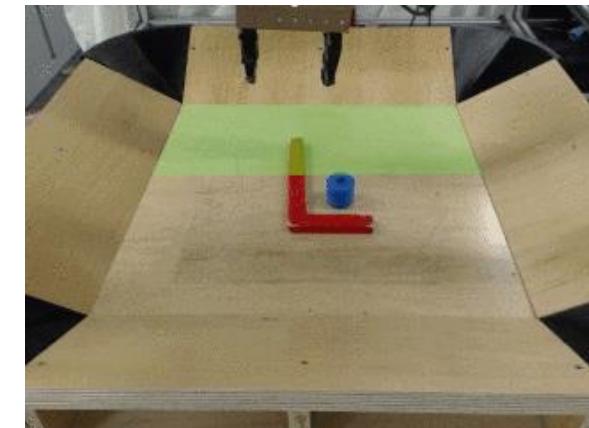
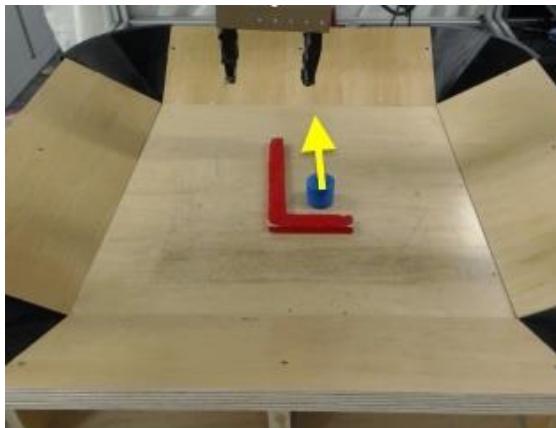
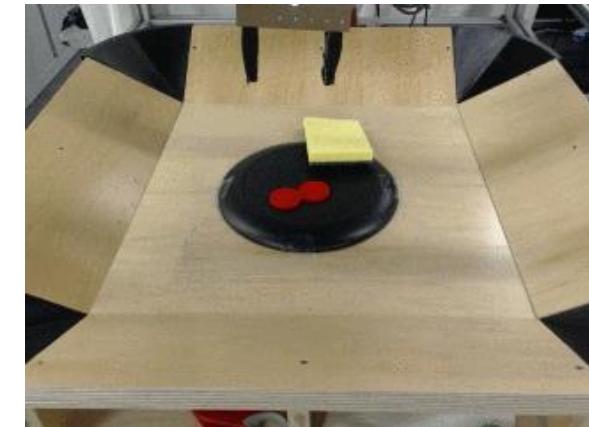
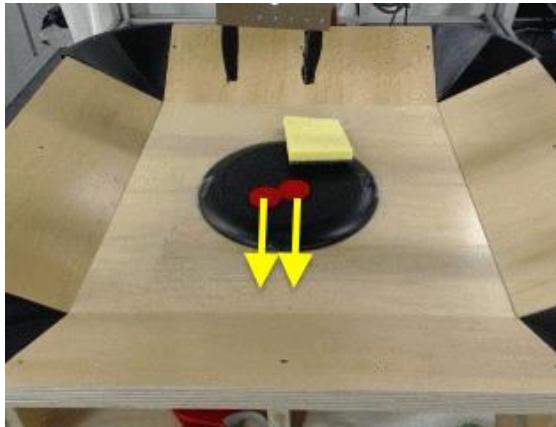
Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects

Rajesh P. N. Rao³ and Dana H. Ballard²

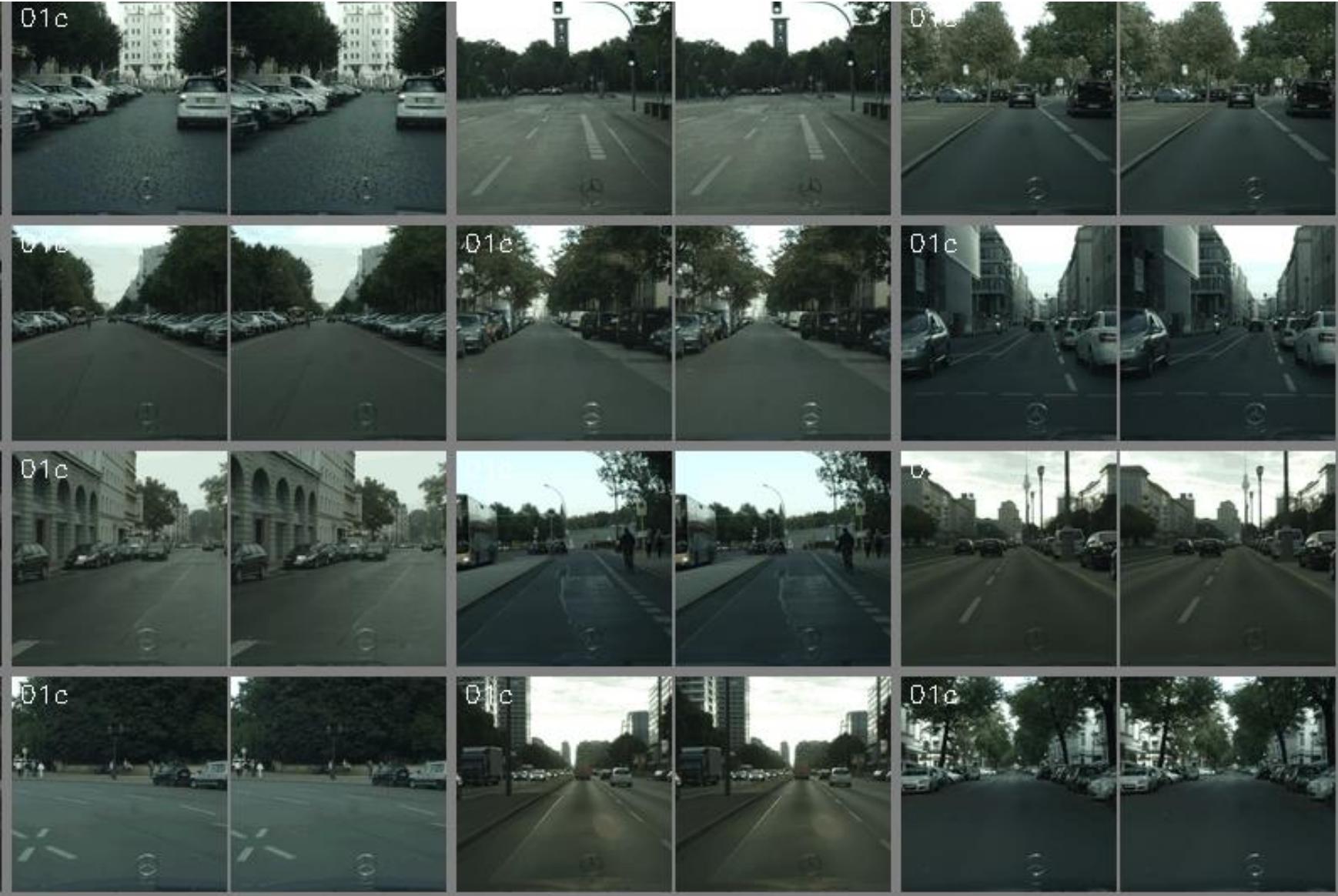
Prediction for real-world control



Using tools with predictive models



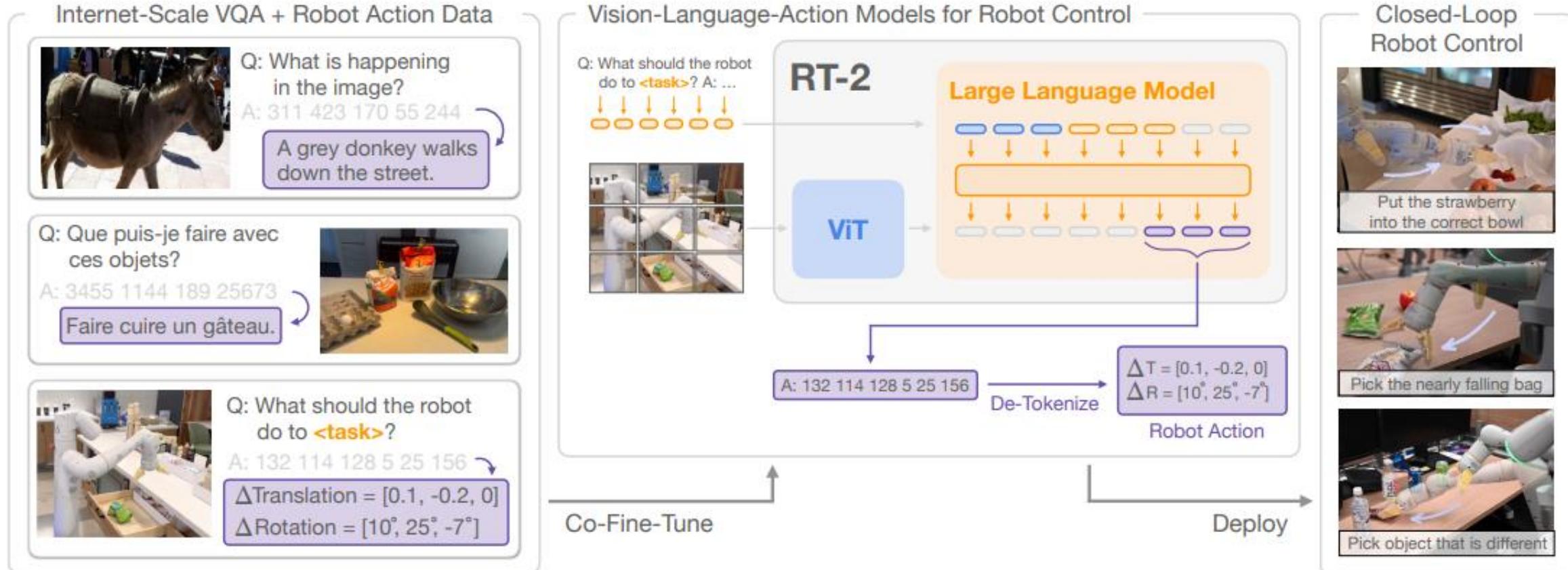
Predictive models have come a long way!



Predictive models have come a long way!



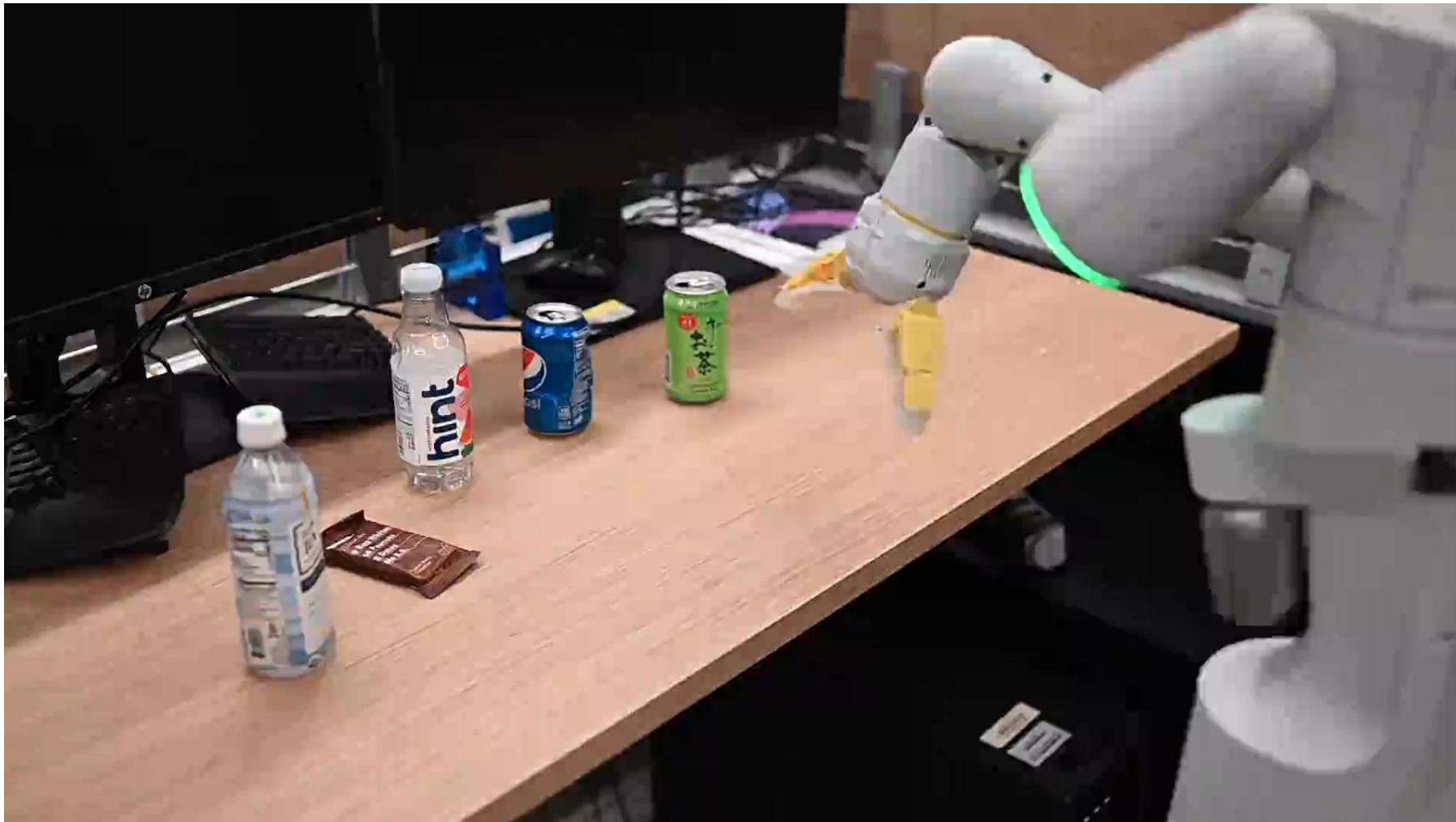
Leveraging advances in pretrained models



Leveraging advances in pretrained models



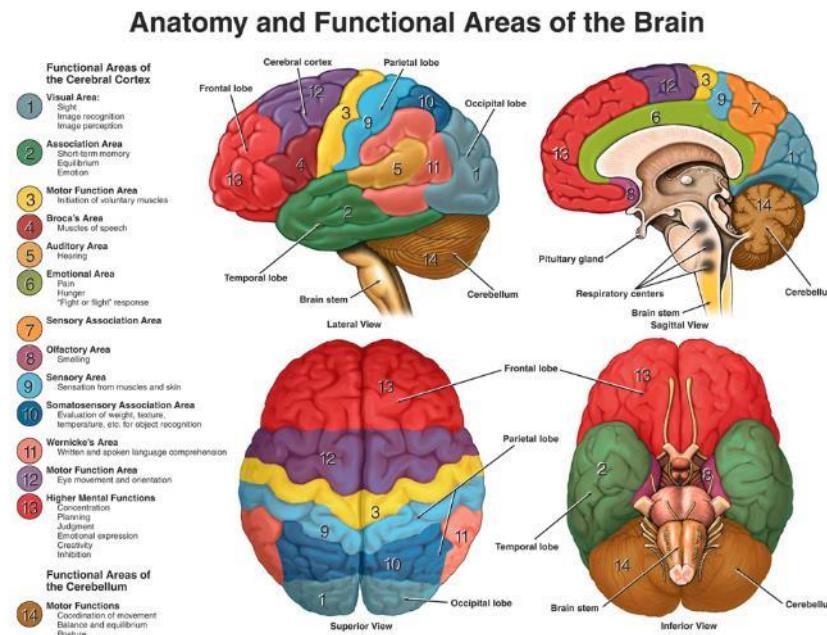
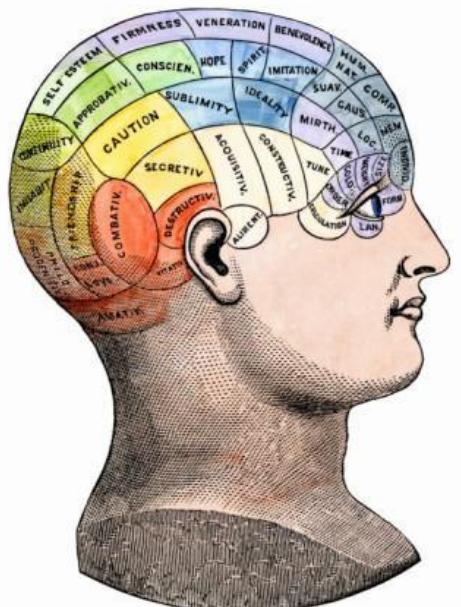
Leveraging advances in pretrained models



How do we build intelligent machines?

How do we build intelligent machines?

- Imagine you have to build an intelligent machine, where do you start?



Learning as the basis of intelligence

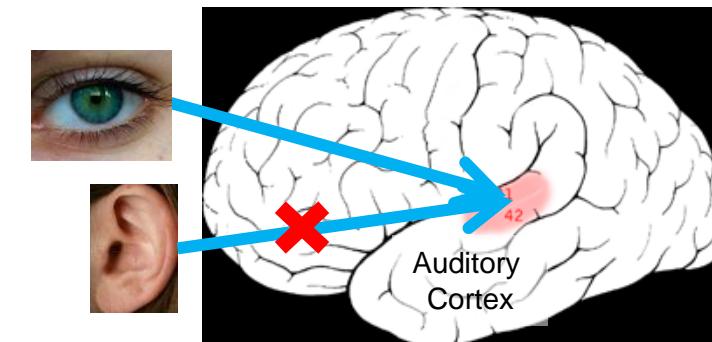
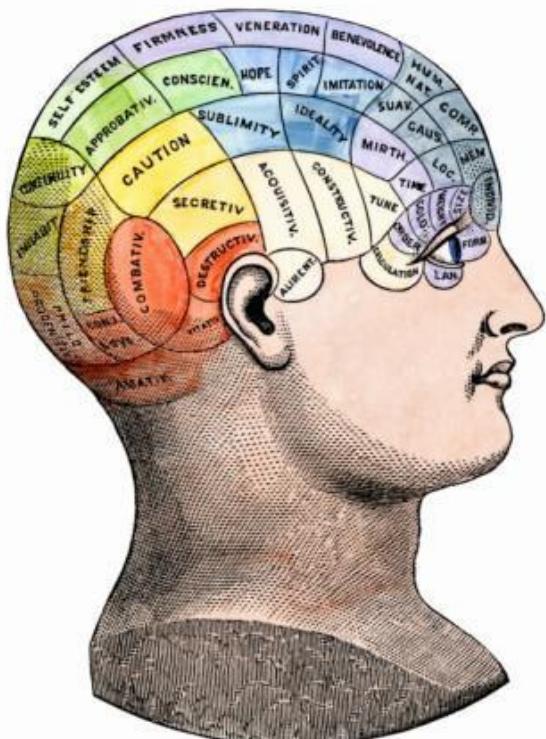
- Some things we can all do (e.g. walking)
- Some things we can only learn (e.g. driving a car)
- We can learn a huge variety of things, including very difficult things
- Therefore our learning mechanism(s) are likely powerful enough to do everything we associate with intelligence
 - But it may still be very convenient to “hard-code” a few really important bits

A single algorithm?

- An algorithm for each “module”?
- Or a single flexible algorithm?



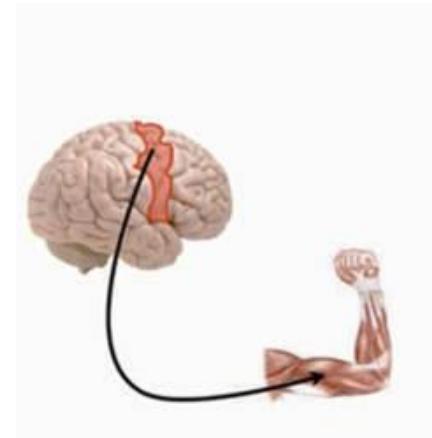
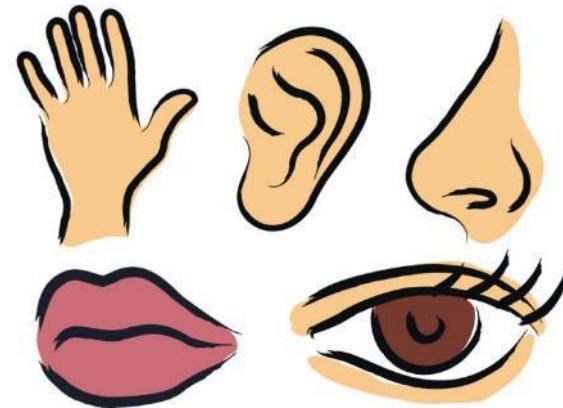
Seeing with your tongue



[BrainPort; Martinez et al; Roe et al.]
adapted from A. Ng

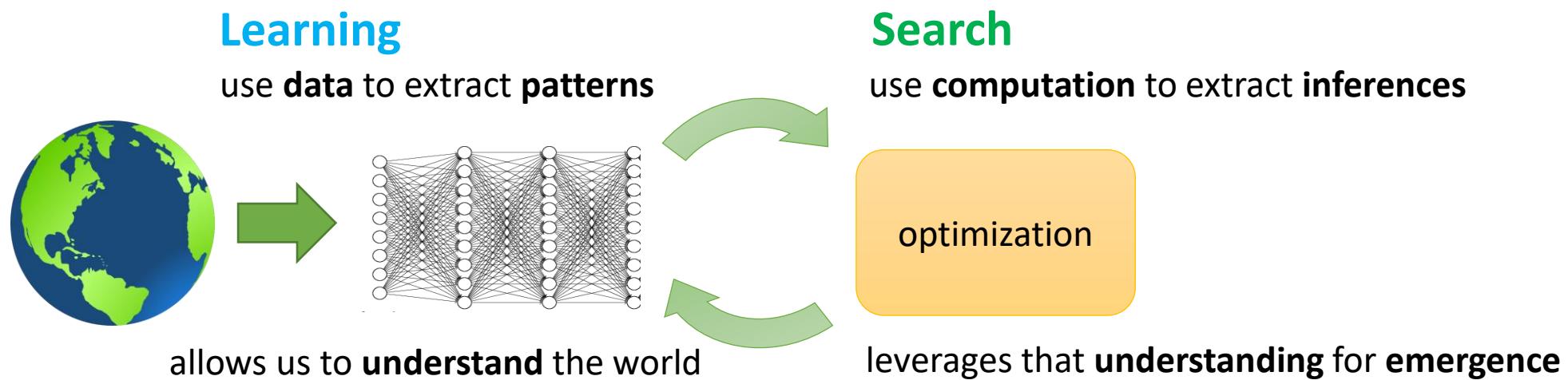
What must that single algorithm do?

- Interpret rich sensory inputs
- Choose complex actions

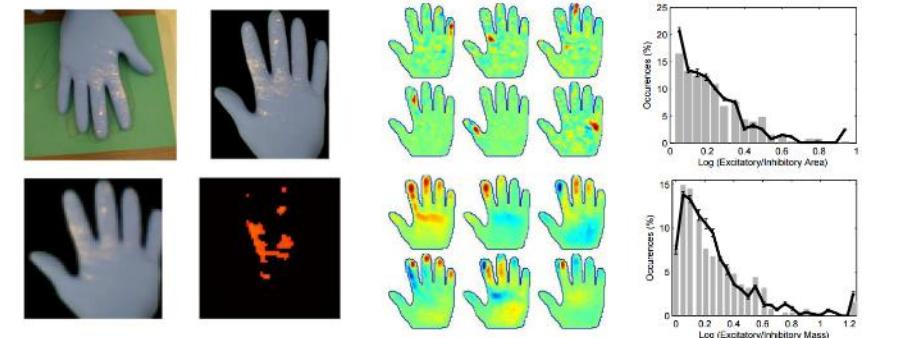
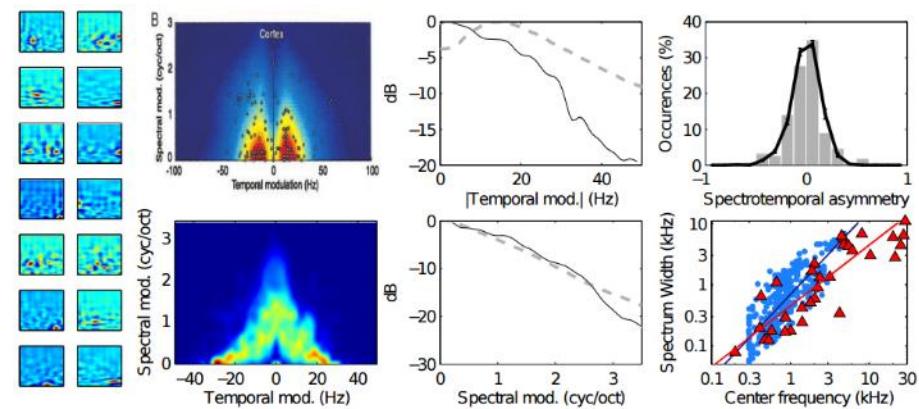
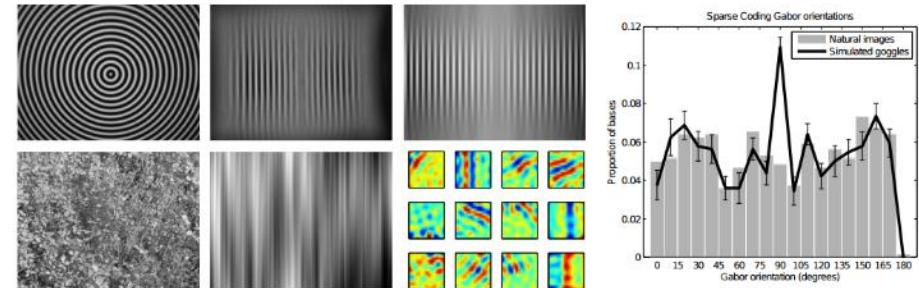


Why deep reinforcement learning?

- Deep = scalable learning from large, complex datasets
- Reinforcement learning = optimization



Some evidence in favor of deep learning



Unsupervised learning models of primary cortical receptive fields and receptive field plasticity

Andrew Saxe, Maneesh Bhand, Ritvik Mudur, Bipin Suresh, Andrew Y. Ng
Department of Computer Science
Stanford University
`{asaxe, mbhand, rmudur, bipins, ang}@cs.stanford.edu`

Some evidence for reinforcement learning

- Percepts that anticipate reward become associated with similar firing patterns as the reward itself
- Basal ganglia appears to be related to reward system
- Model-free RL-like adaptation is often a good fit for experimental data of animal adaptation
 - But not always...

Reinforcement learning in the brain

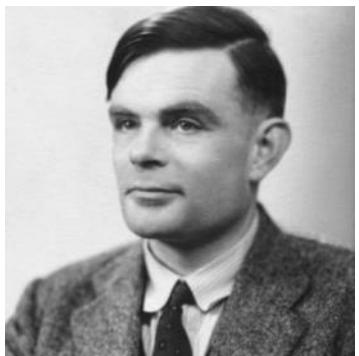
Yael Niv

Psychology Department & Princeton Neuroscience Institute, Princeton University

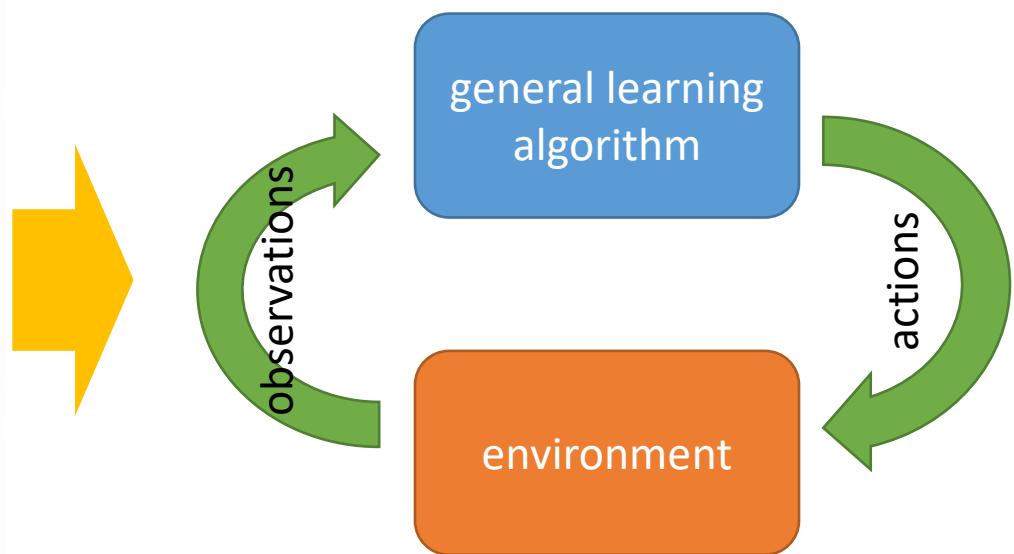
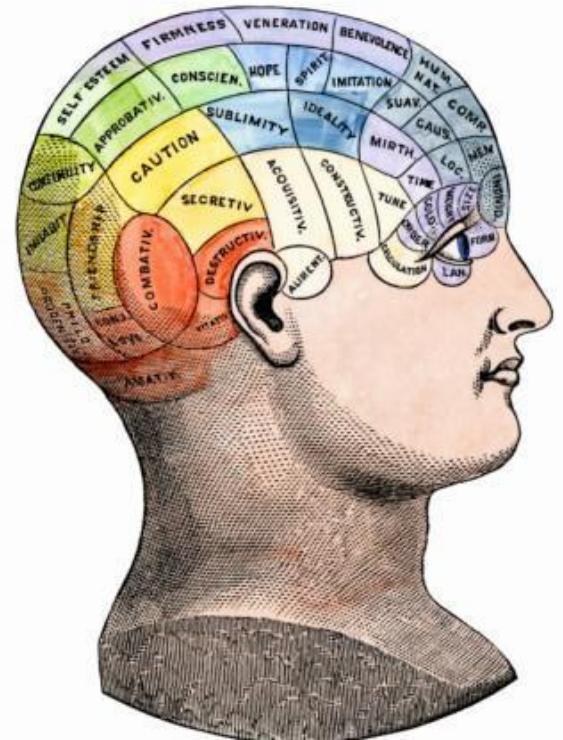
What challenges still remain?

- We have great methods that can learn from huge amounts of data
- We have great optimization methods for RL
- We don't (yet) have amazing methods that **both** use data and RL
- Humans can learn incredibly quickly, deep RL methods are usually slow
- Humans reuse past knowledge, transfer learning in RL is an open problem
- Not clear what the reward function should be
- Not clear what the role of prediction should be

Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain.



- Alan Turing



Supervised Learning of Behaviors

CS 285

Instructor: Sergey Levine
UC Berkeley



Terminology & notation



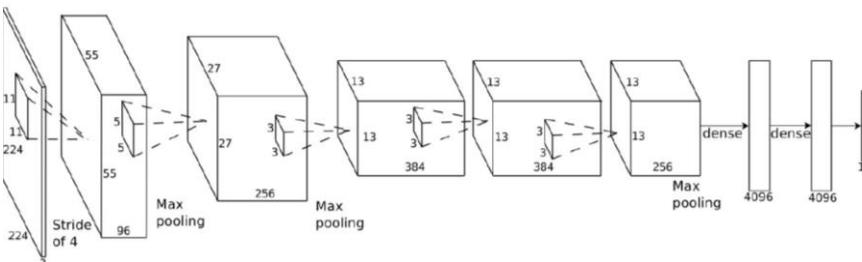
\mathbf{o}_t



\mathbf{s}_t – state

\mathbf{o}_t – observation

\mathbf{a}_t – action



$$\pi_{\theta}(\mathbf{a} | \mathbf{o})$$



\mathbf{a}_t



$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$ – policy

$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$ – policy (fully observed)



\mathbf{o}_t – observation

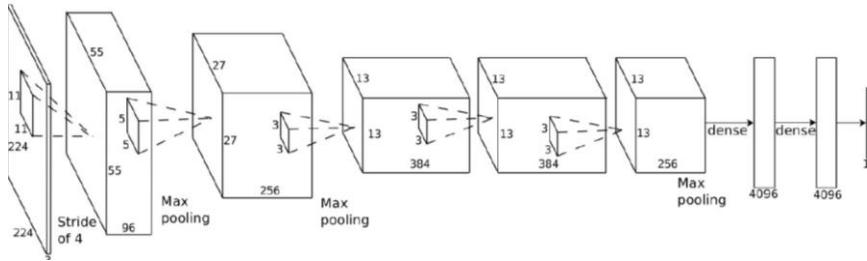


\mathbf{s}_t – state

Terminology & notation



\mathbf{o}_t



$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$



\mathbf{a}_t

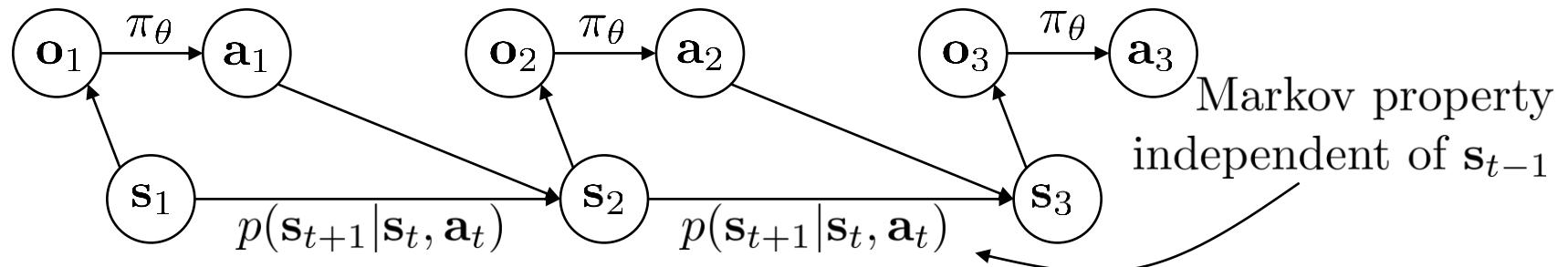
\mathbf{s}_t – state

\mathbf{o}_t – observation

\mathbf{a}_t – action

$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$ – policy

$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ – policy (fully observed)



Aside: notation

s_t – state

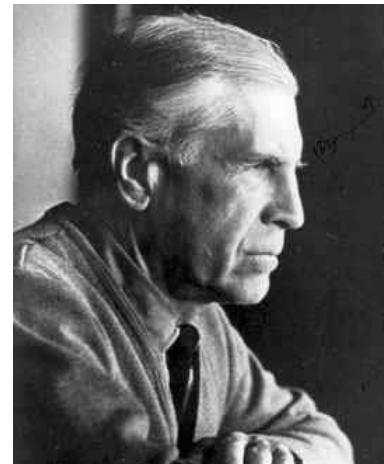
a_t – action



Richard Bellman

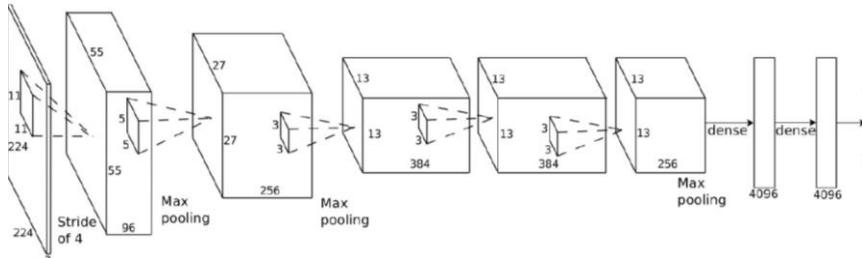
x_t – state

u_t – action управление

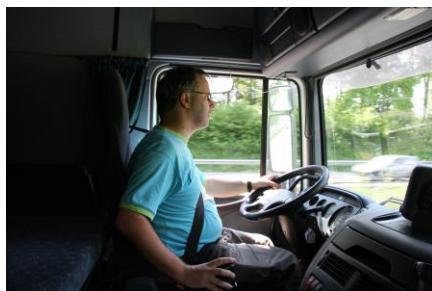


Lev Pontryagin

Imitation Learning



$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$$



\mathbf{o}_t
 \mathbf{a}_t

training
data

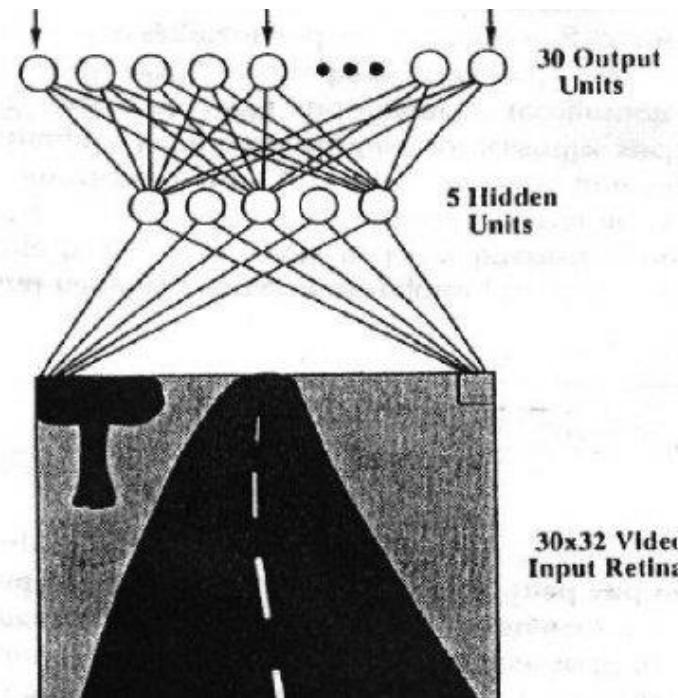
supervised
learning

$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$$

behavioral cloning

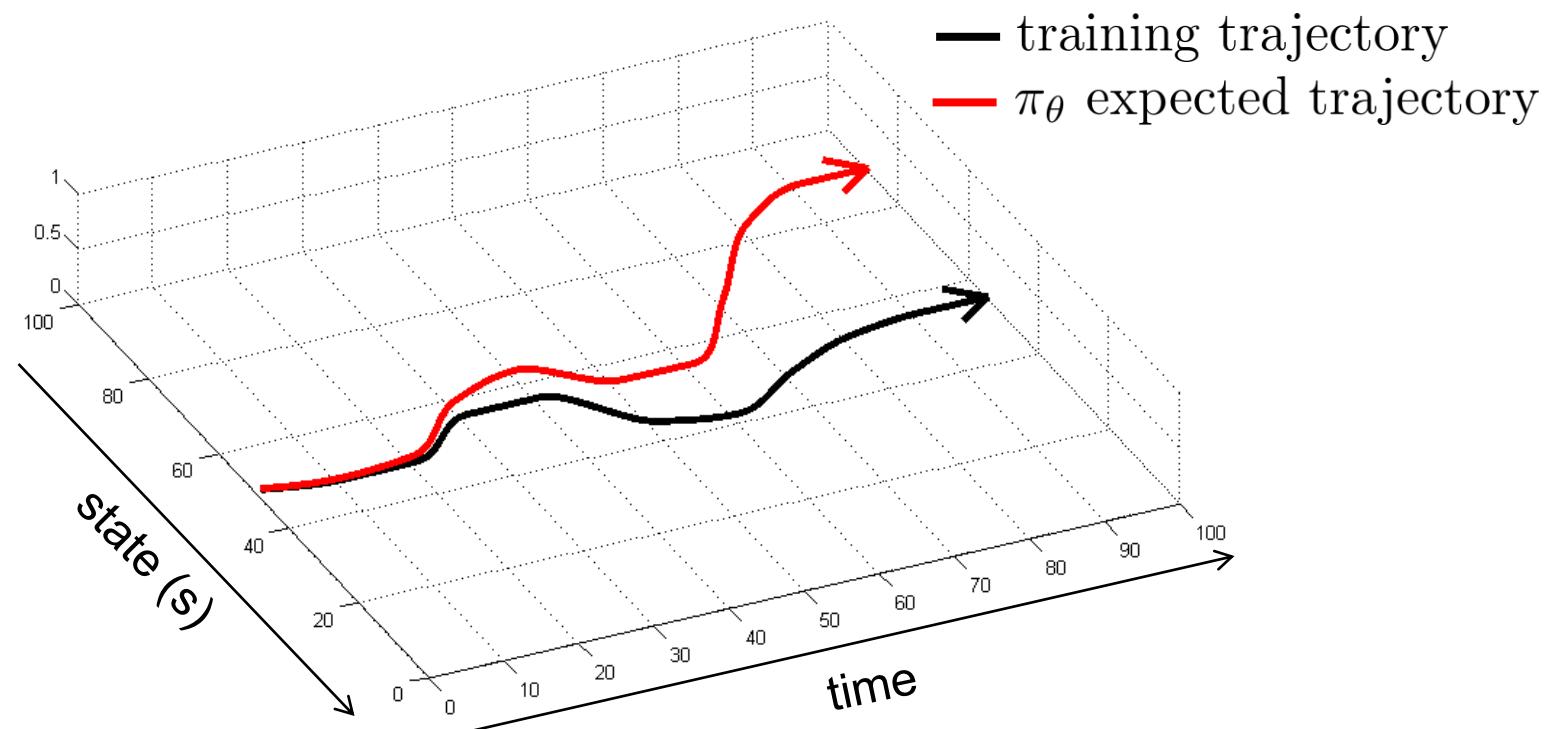
The original deep imitation learning system

ALVINN: Autonomous Land Vehicle In a Neural Network
1989



Does it work?

No!

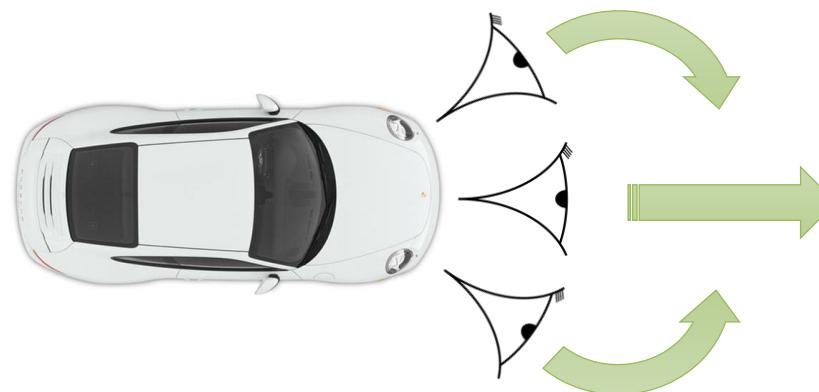
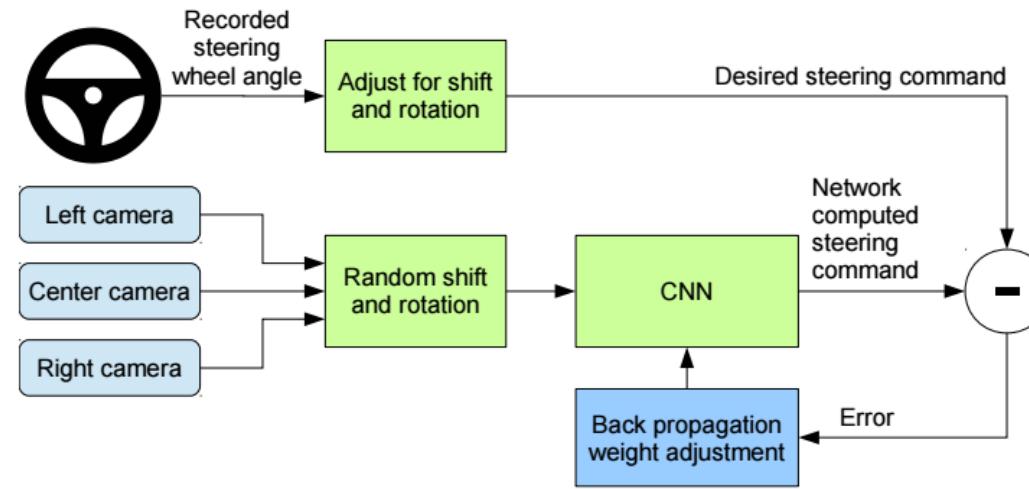


Does it work?

Yes!

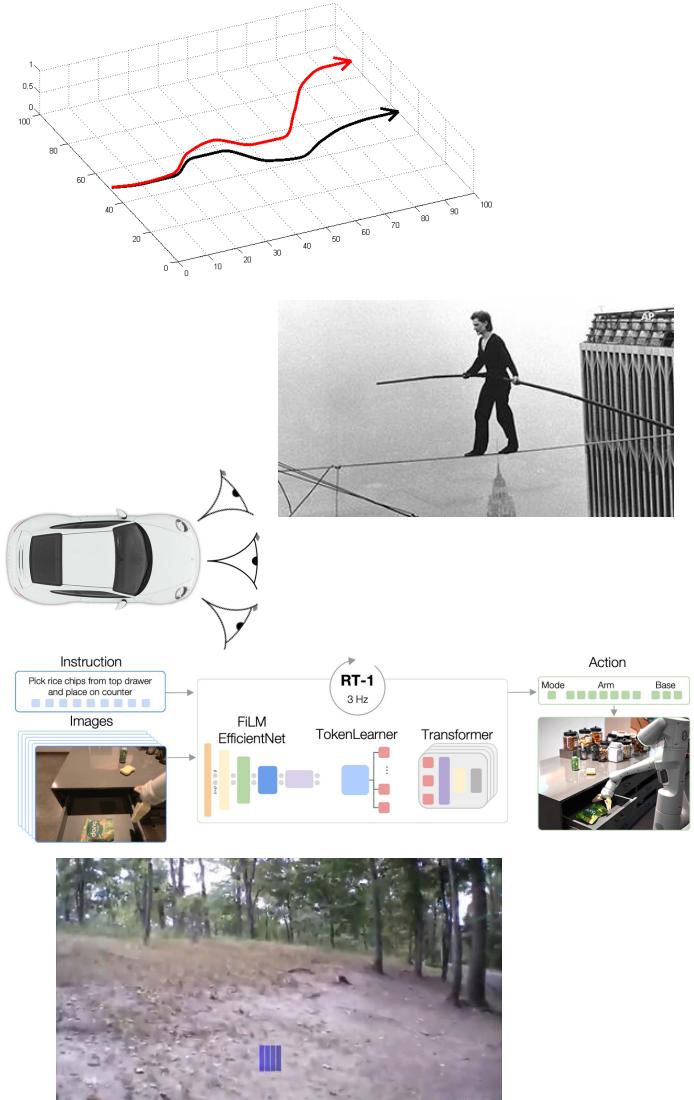
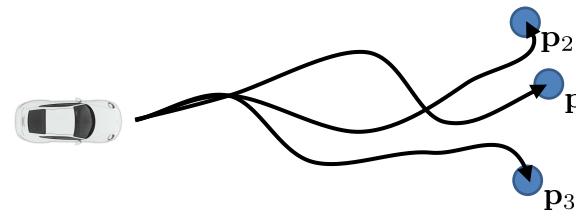


Why did that work?



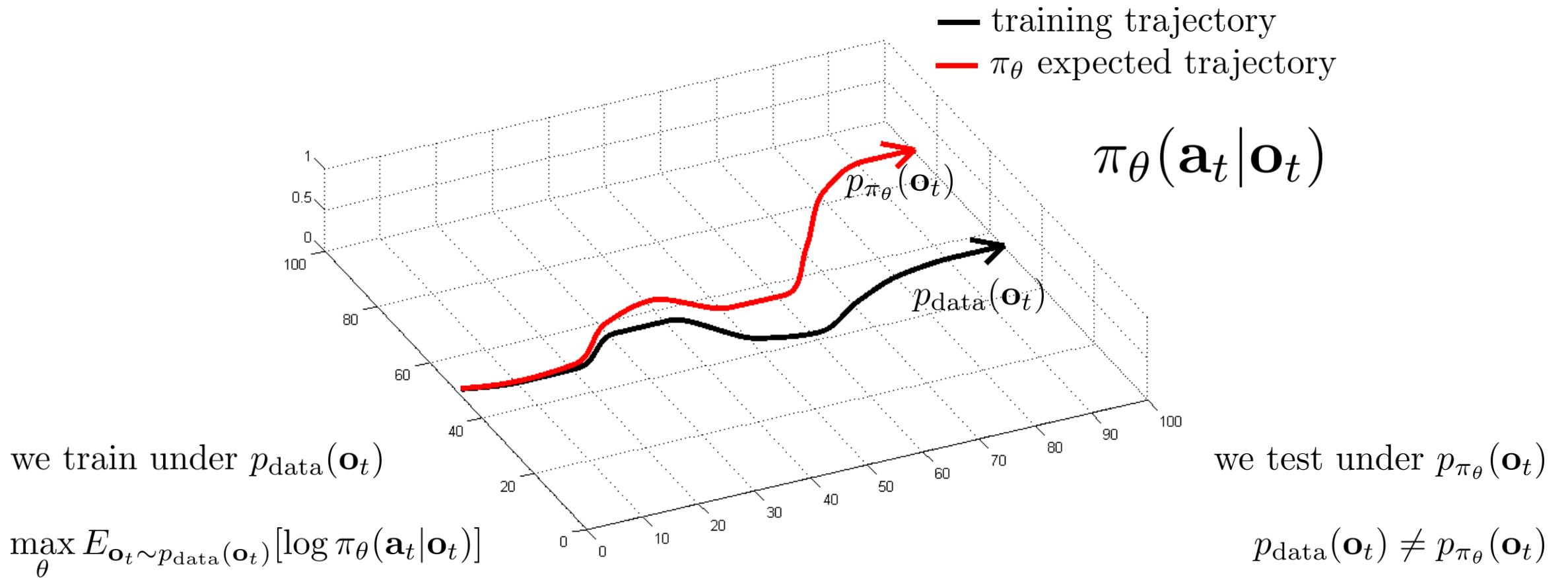
The moral of the story, and a list of ideas

- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- We can address the problem in a few ways:
 - Be smart about how we collect (and augment) our data
 - Use very powerful models that make very few mistakes
 - Use multi-task learning
 - Change the algorithm (DAgger)

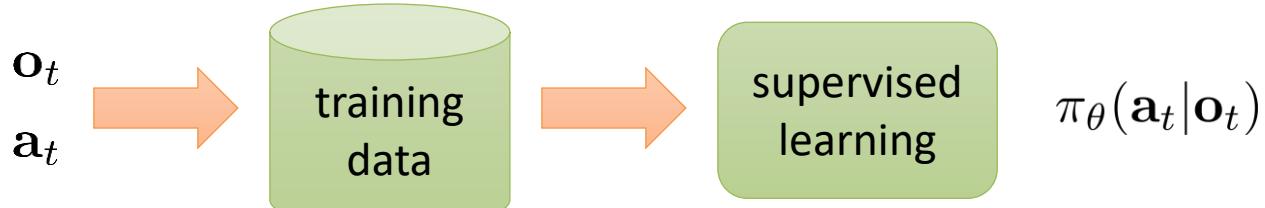


Why does behavioral cloning fail?
A bit of theory

The distributional shift problem



Let's define more precisely what we want



What makes a learned $\pi_{\theta}(\mathbf{a}_t|\mathbf{o}_t)$ good or bad?

$$c(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 0 & \text{if } \mathbf{a}_t = \pi^*(\mathbf{s}_t) \\ 1 & \text{otherwise} \end{cases}$$

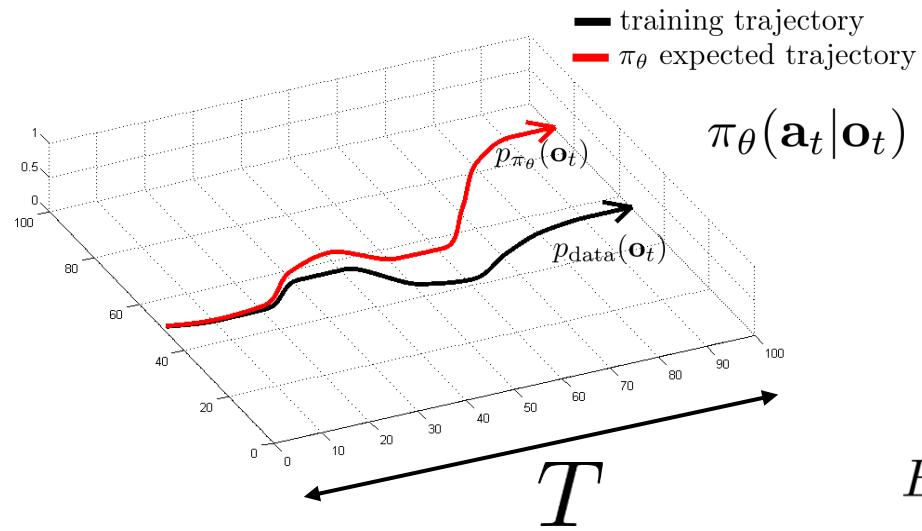
Goal: minimize $E_{\mathbf{s}_t \sim p_{\pi_{\theta}}(\mathbf{s}_t)}[c(\mathbf{s}_t, \mathbf{a}_t)]$

$$\cancel{\max_{\theta} E_{\mathbf{o}_t \sim p_{\text{data}}(\mathbf{o}_t)}[\log \pi_{\theta}(\mathbf{a}_t|\mathbf{o}_t)]}$$

Note: I started mixing up \mathbf{s} and \mathbf{o}
I warned you about that...

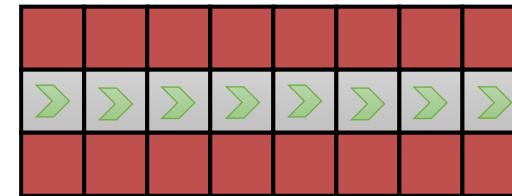
“Minimize the number of mistakes
the policy makes when we run it”

Some analysis



$$c(\mathbf{s}, \mathbf{a}) = \begin{cases} 0 & \text{if } \mathbf{a} = \pi^*(\mathbf{s}) \\ 1 & \text{otherwise} \end{cases}$$

assume: $\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s}) | \mathbf{s}) \leq \epsilon$
for all $\mathbf{s} \in \mathcal{D}_{\text{train}}$



$$E \left[\sum_t c(\mathbf{s}_t, \mathbf{a}_t) \right] \leq \underbrace{\epsilon T + O(\epsilon T^2)}_{T \text{ terms, each } O(\epsilon T)}$$

More general analysis

$$c(s, a) = \begin{cases} 0 & \text{if } a = \pi^*(s) \\ 1 & \text{otherwise} \end{cases}$$

assume: $\pi_\theta(a \neq \pi^*(s)|s) \leq \epsilon$

~~for all $s \in \mathcal{D}_{\text{train}}$ for $s \sim p_{\text{train}}(s)$~~

actually enough for $E_{p_{\text{train}}(s)}[\pi_\theta(a \neq \pi^*(s)|s)] \leq \epsilon$

if $p_{\text{train}}(s) \neq p_\theta(s)$:

$$p_\theta(s_t) = \underbrace{(1 - \epsilon)^t p_{\text{train}}(s_t)}_{\text{probability we made no mistakes}} + \underbrace{(1 - (1 - \epsilon)^t)) p_{\text{mistake}}(s_t)}_{\text{some } \textit{other} \text{ distribution}}$$

probability we made no mistakes

some *other* distribution

More general analysis

assume: $\pi_\theta(\mathbf{a} \neq \pi^*(\mathbf{s}) | \mathbf{s}) \leq \epsilon$

for all $\mathbf{s} \in \mathcal{D}_{\text{train}}$ for $\mathbf{s} \sim p_{\text{train}}(\mathbf{s})$

$$p_{\theta}(\mathbf{s}_t) = \underbrace{(1 - \epsilon)^t p_{\text{train}}(\mathbf{s}_t)}_{\text{correct}} + \underbrace{(1 - (1 - \epsilon)^t)) p_{\text{mistake}}(\mathbf{s}_t)}_{\text{incorrect}}$$

probability we made no mistakes

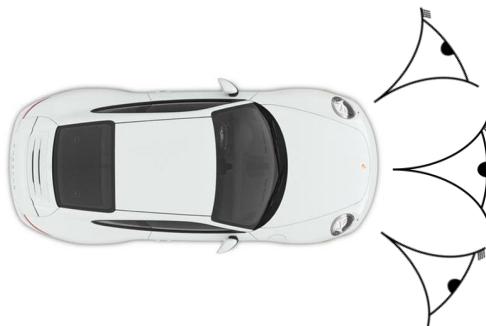
some *other* distribution

$$|p_\theta(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| = (1 - (1 - \epsilon)^t) |p_{\text{mistake}}(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| \leq 2(1 - (1 - \epsilon)^t)$$

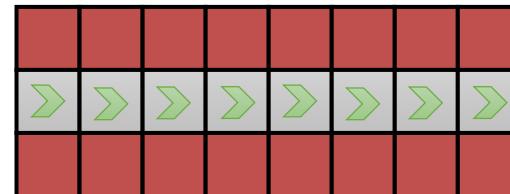
useful identity: $(1 - \epsilon)^t \geq 1 - \epsilon t$ for $\epsilon \in [0, 1]$

$$\begin{aligned} \sum_t E_{p_\theta(\mathbf{s}_t)}[c_t] &= \sum_t \sum_{\mathbf{s}_t} p_\theta(\mathbf{s}_t) c_t(\mathbf{s}_t) \leq \sum_t \sum_{\mathbf{s}_t} p_{\text{train}}(\mathbf{s}_t) c_t(\mathbf{s}_t) + |p_\theta(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| c_{\max} \\ &\leq \sum_t \epsilon + 2\epsilon t \\ &= O(\epsilon T^2) \end{aligned}$$

Why is this rather pessimistic?

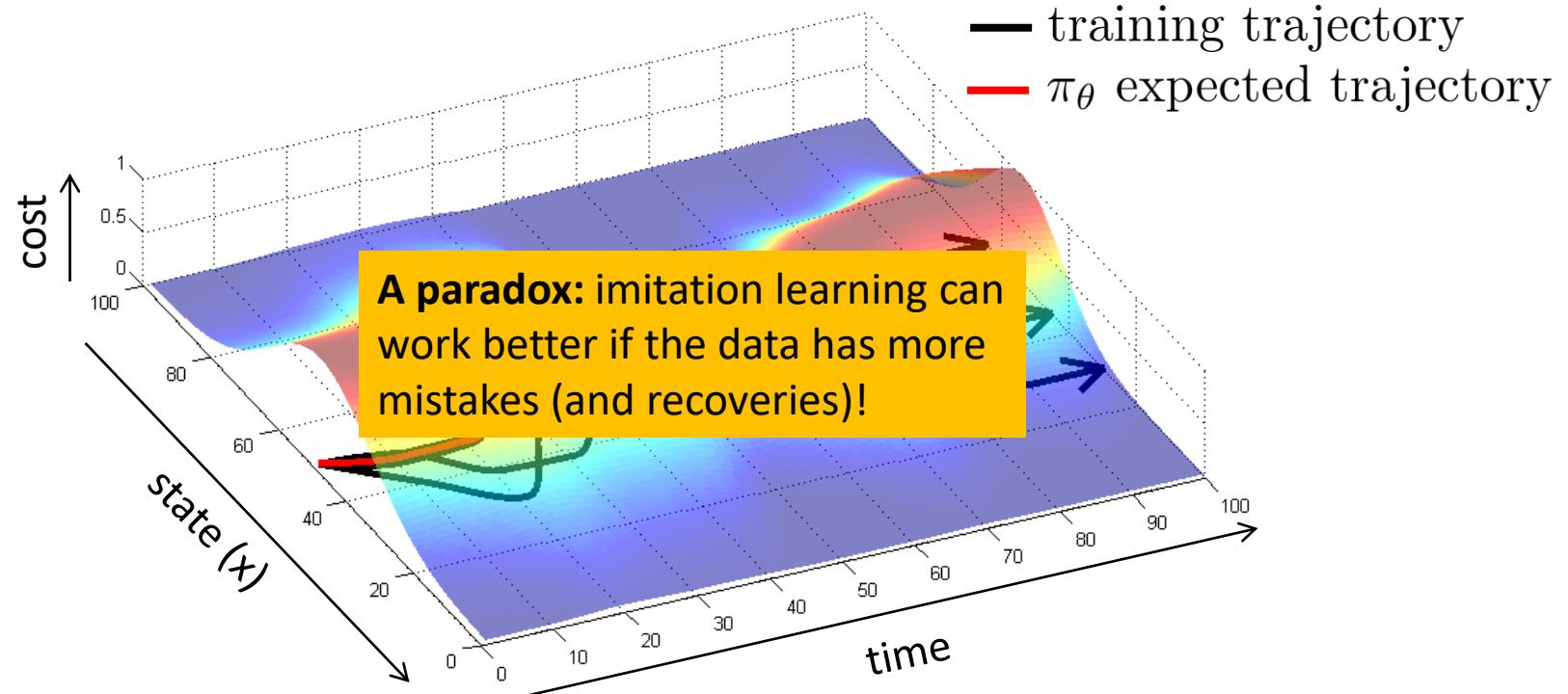


Why does this work?



In reality, we can often **recover** from mistakes

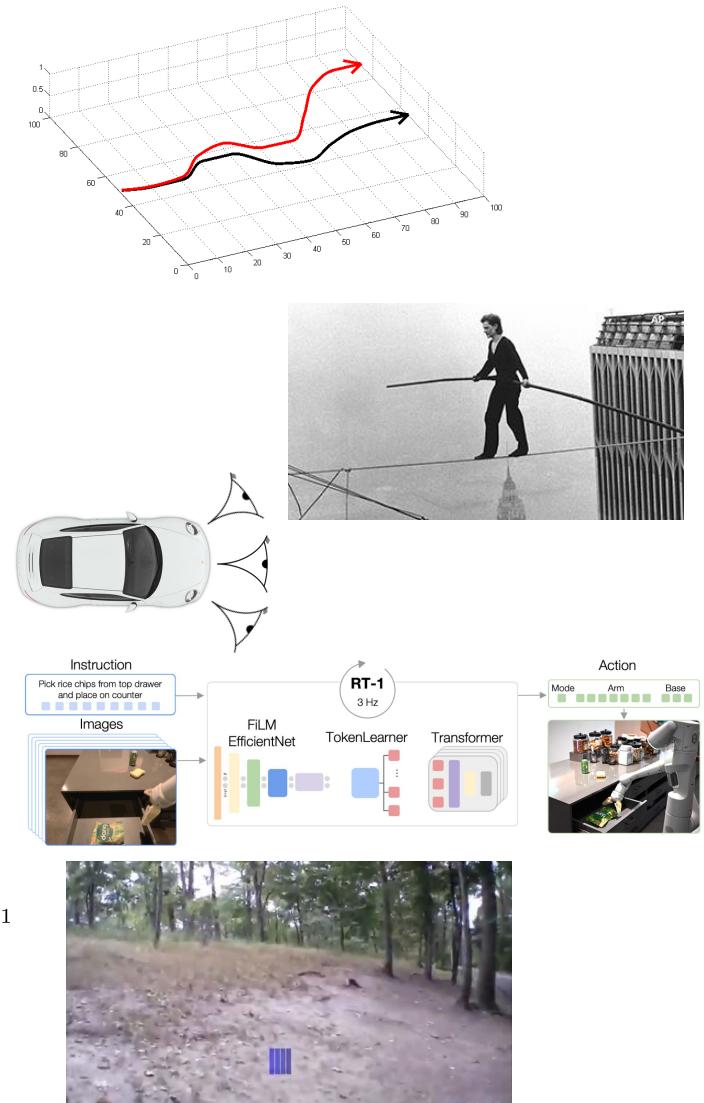
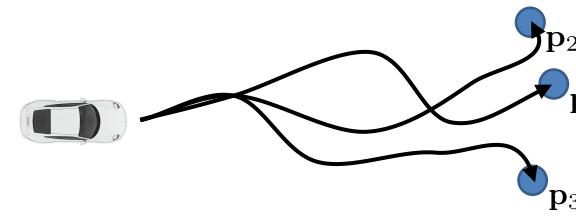
But that doesn't mean that **imitation learning** will allow us to learn how to do that!



Addressing the problem in practice

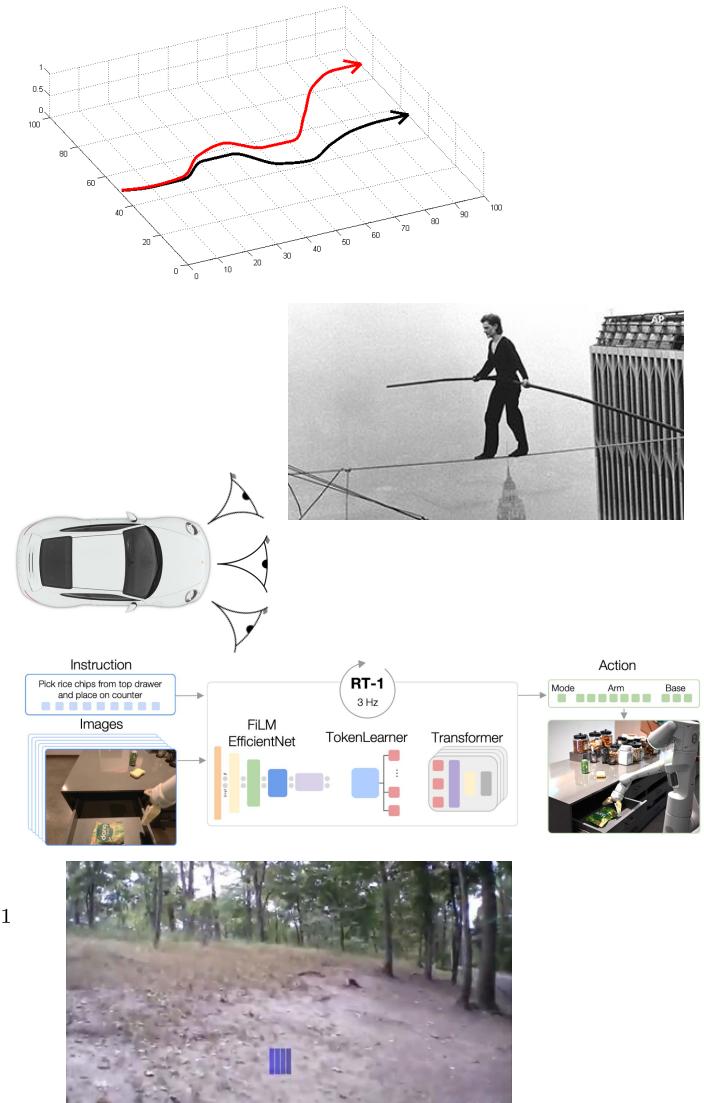
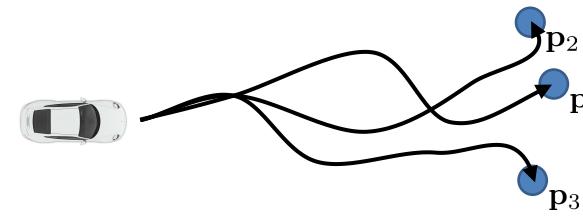
Where are we...

- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- We can address the problem in a few ways:
 - Be smart about how we collect (and augment) our data
 - Use very powerful models that make very few mistakes
 - Use multi-task learning
 - Change the algorithm (DAgger)

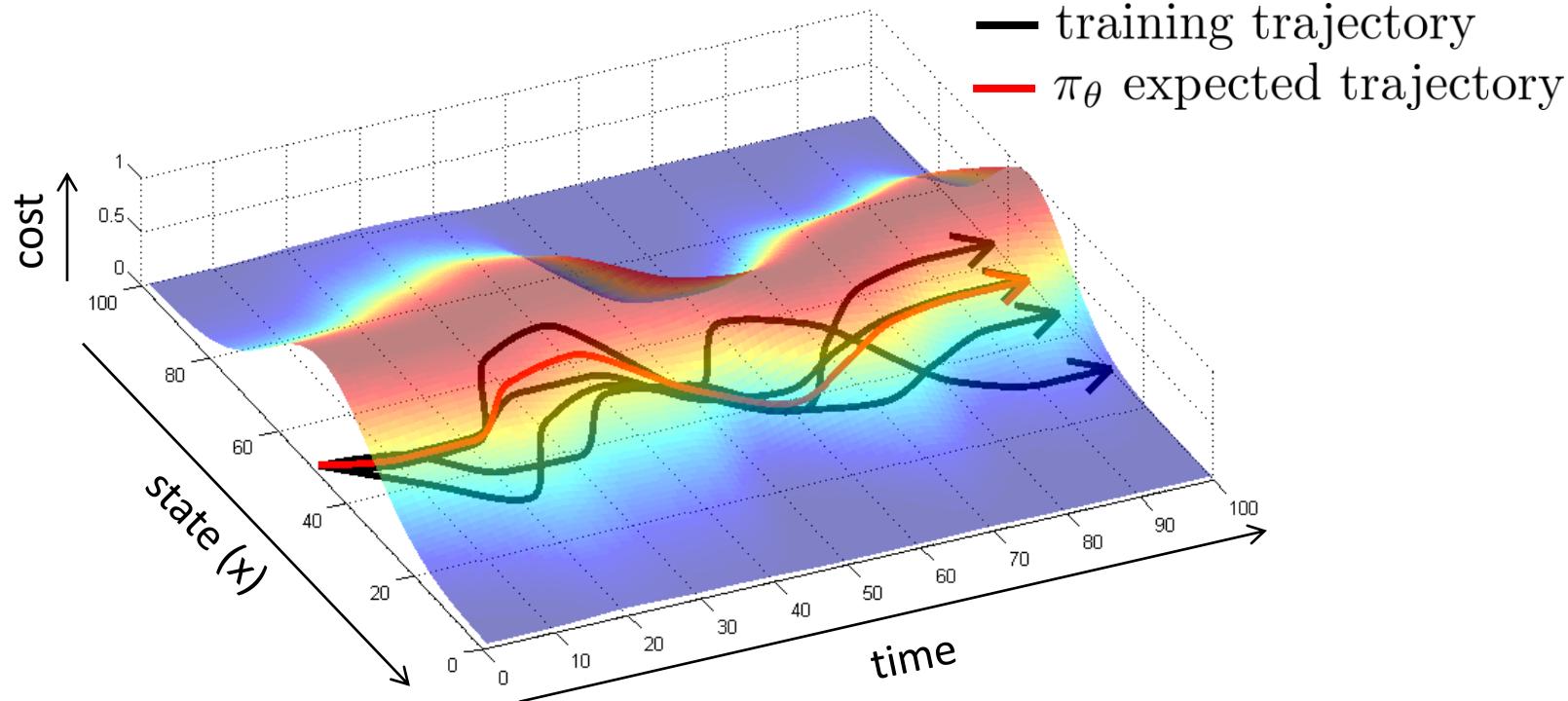


Where are we...

- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- We can address the problem in a few ways:
 - Be smart about how we collect (and augment) our data
 - Use very powerful models that make very few mistakes
 - Use multi-task learning
 - Change the algorithm (DAgger)



What makes behavioral cloning easy and what makes it hard?



- Intentionally add **mistakes and corrections**
 - The mistakes hurt, but the corrections help, often more than the mistakes hurt
- Use **data augmentation**
 - Add some “fake” data that illustrates corrections (e.g., side-facing cameras)

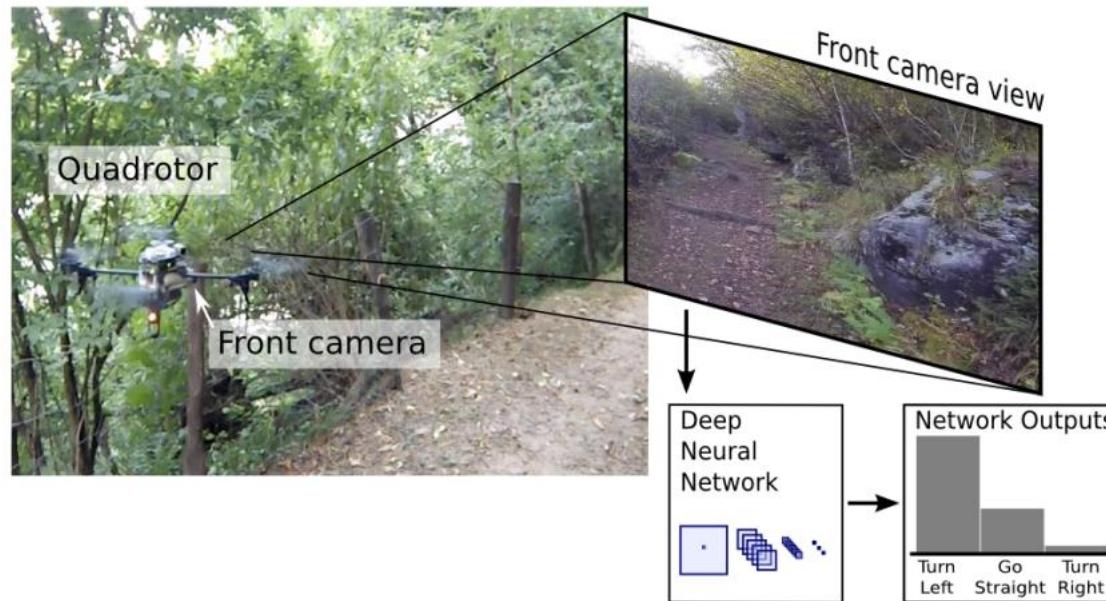
Case study 1: trail following as classification

A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots

Alessandro Giusti¹, Jérôme Guzzi¹, Dan C. Cireşan¹, Fang-Lin He¹, Juan P. Rodríguez¹

Flavio Fontana², Matthias Faessler², Christian Forster²

Jürgen Schmidhuber¹, Gianni Di Caro¹, Davide Scaramuzza², Luca M. Gambardella¹



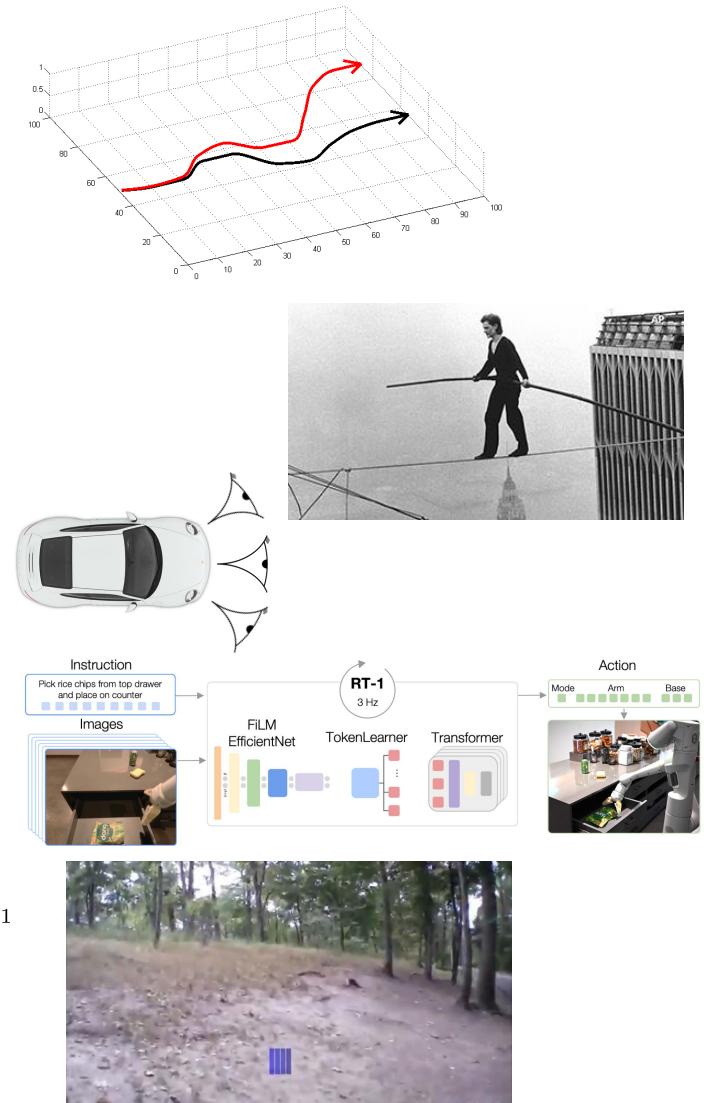
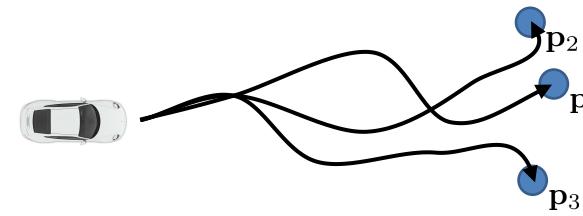
Case study 2: imitation with a cheap robot

Vision-Based Multi-Task Manipulation
for Inexpensive Robots
Using End-To-End Learning from Demonstration

Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Boloni, Sergey Levine

Where are we...

- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- **We can address the problem in a few ways:**
 - Be smart about how we collect (and augment) our data
 - **Use very powerful models that make very few mistakes**
 - Use multi-task learning
 - Change the algorithm (Dagger)



Why might we fail to fit the expert?



1. Non-Markovian behavior
2. Multimodal behavior

$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$$

behavior depends only
on current observation

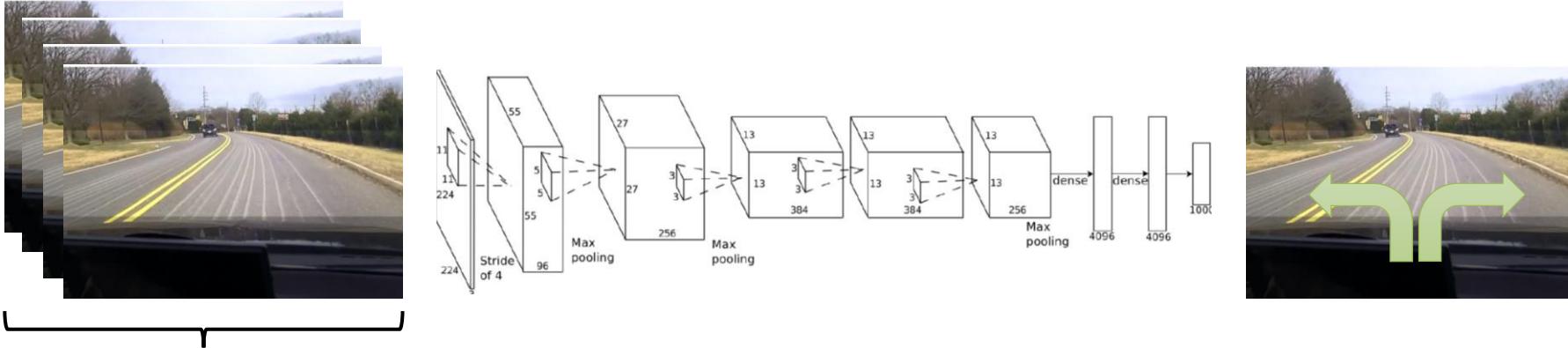
$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_1, \dots, \mathbf{o}_t)$$

behavior depends on
all past observations

If we see the same thing
twice, we do the same thing
twice, regardless of what
happened before

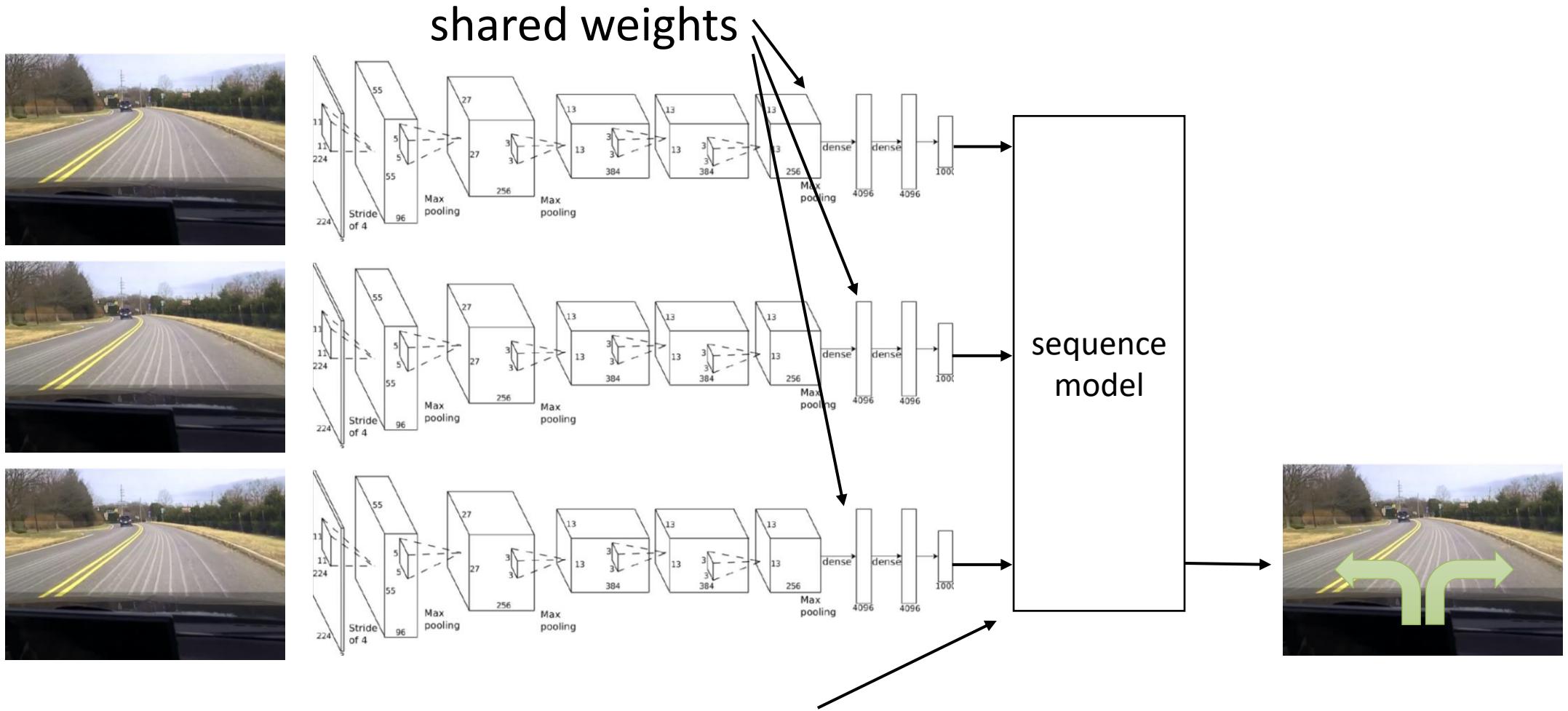
Often very unnatural for
human demonstrators

How can we use the whole history?

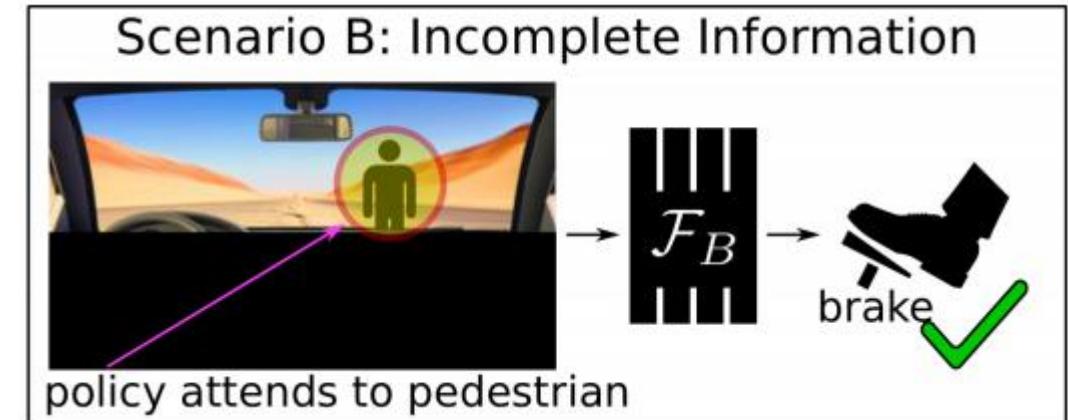
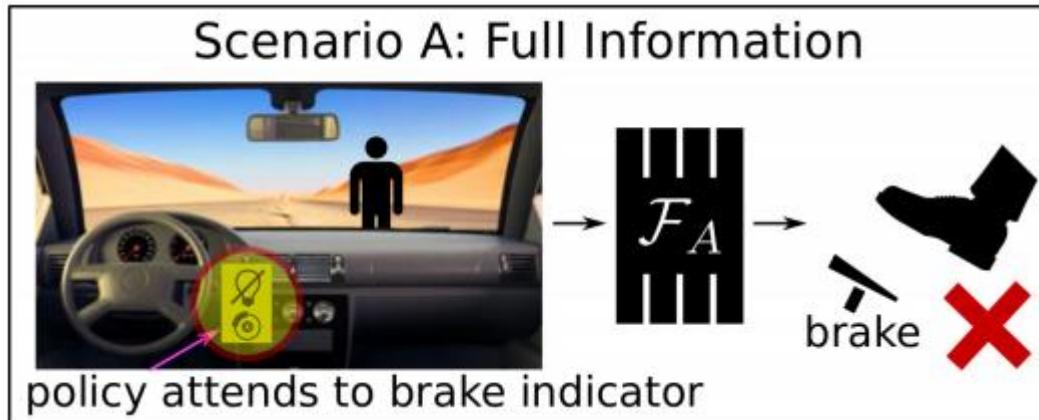


variable number of frames,
too many weights

How can we use the whole history?



Aside: why might this work poorly?



“causal confusion”

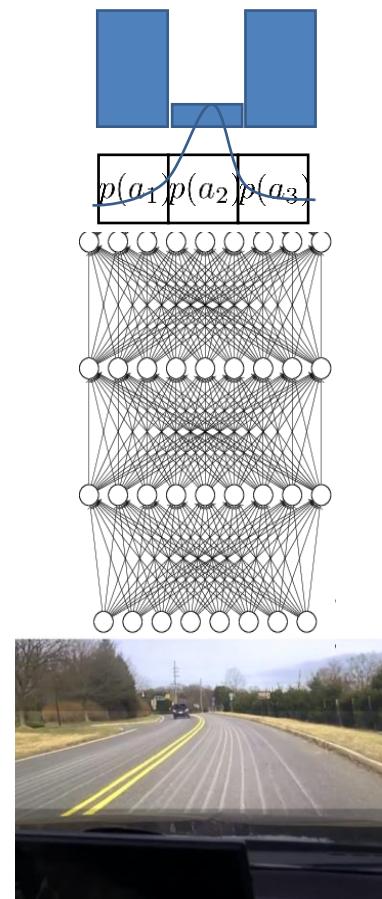
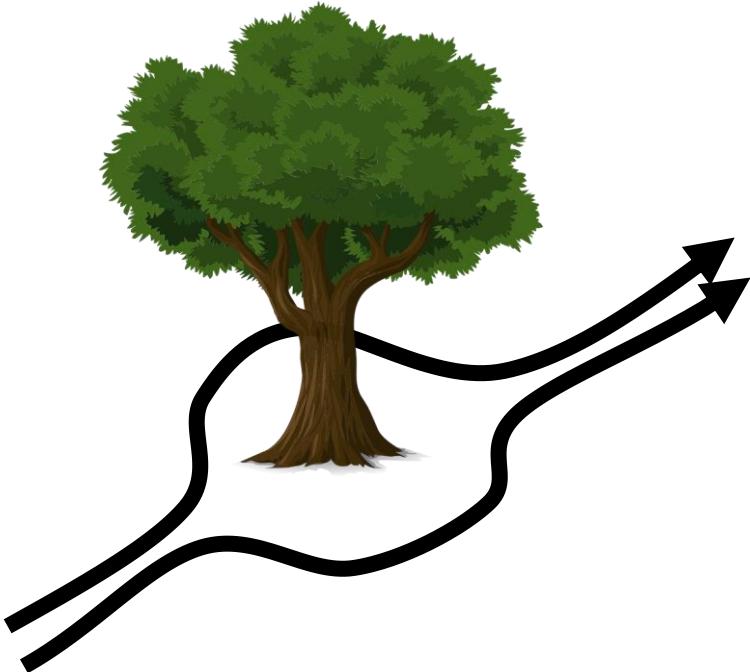
see: de Haan et al., “Causal Confusion in Imitation Learning”

Question 1: Does including history mitigate causal confusion?

Question 2: Can DAgger mitigate causal confusion?

Why might we fail to fit the expert?

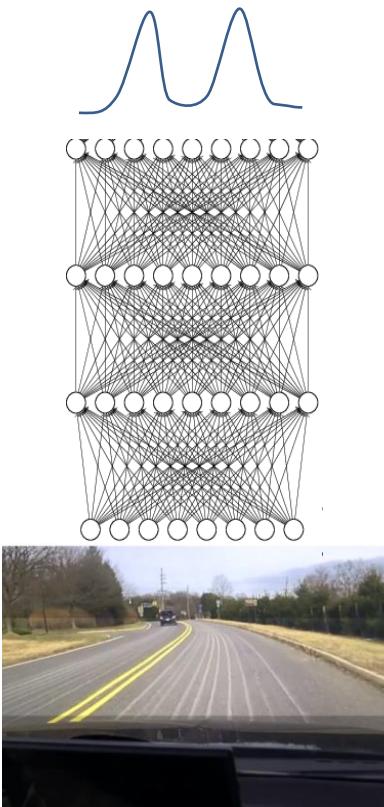
1. Non-Markovian behavior
2. Multimodal behavior



1. More expressive continuous distributions
2. Discretization with high-dimensional action spaces



Expressive continuous distributions



Quite a few options, many ways to make things work:

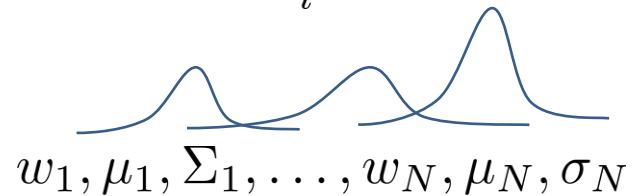
1. mixture of Gaussians
2. latent variable models
3. diffusion models

Expressive continuous distributions

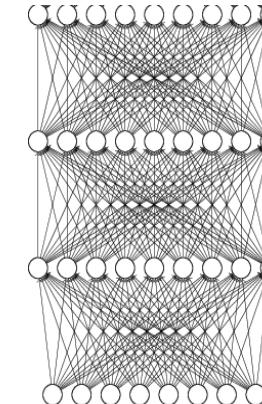


1. mixture of Gaussians
2. latent variable models
3. diffusion models

$$\pi(\mathbf{a}|\mathbf{o}) = \sum_i w_i \mathcal{N}(\mu_i, \Sigma_i)$$



$w_1, \mu_1, \Sigma_1, \dots, w_N, \mu_N, \sigma_N$

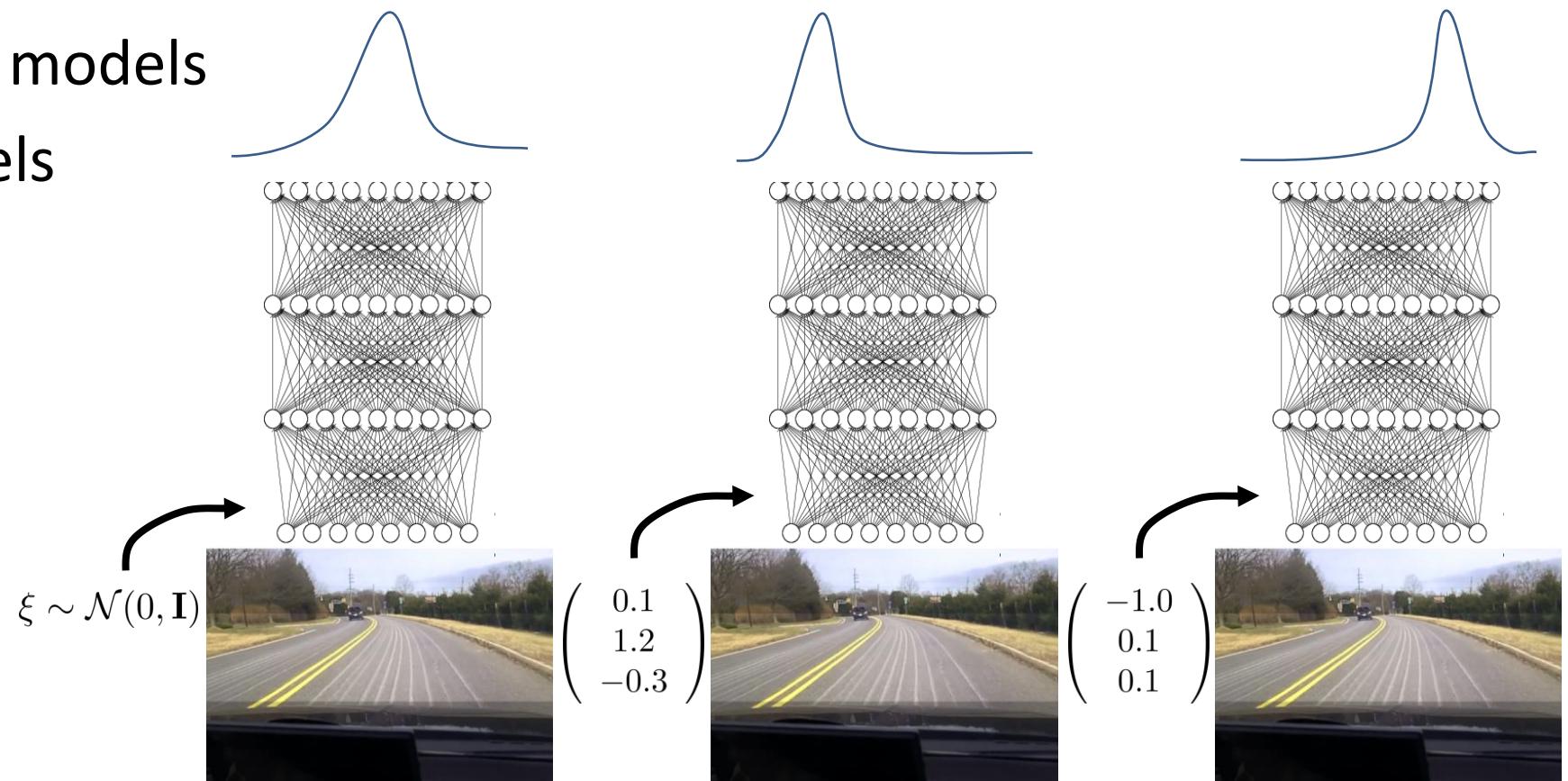


Expressive continuous distributions

- 1. mixture of Gaussians
- 2. latent variable models
- 3. diffusion models

The most widely used type of model of this sort is the (conditional) variational autoencoder

We'll learn about such models later in the course



Expressive continuous distributions

- 1. mixture of Gaussians
- 2. latent variable models
- 3. diffusion models



\mathbf{x}_0 = true image

$\mathbf{x}_{i+1} = \mathbf{x}_i + \text{noise}$

Learned network: $f(\mathbf{x}_i) = \mathbf{x}_{i-1}$

(actually use $f(\mathbf{x}_i) = \text{noise}$)

$\mathbf{x}_{i-1} = \mathbf{x}_i - f(\mathbf{x}_i)$



Expressive continuous distributions

1. mixture of Gaussians
2. latent variable models
3. diffusion models



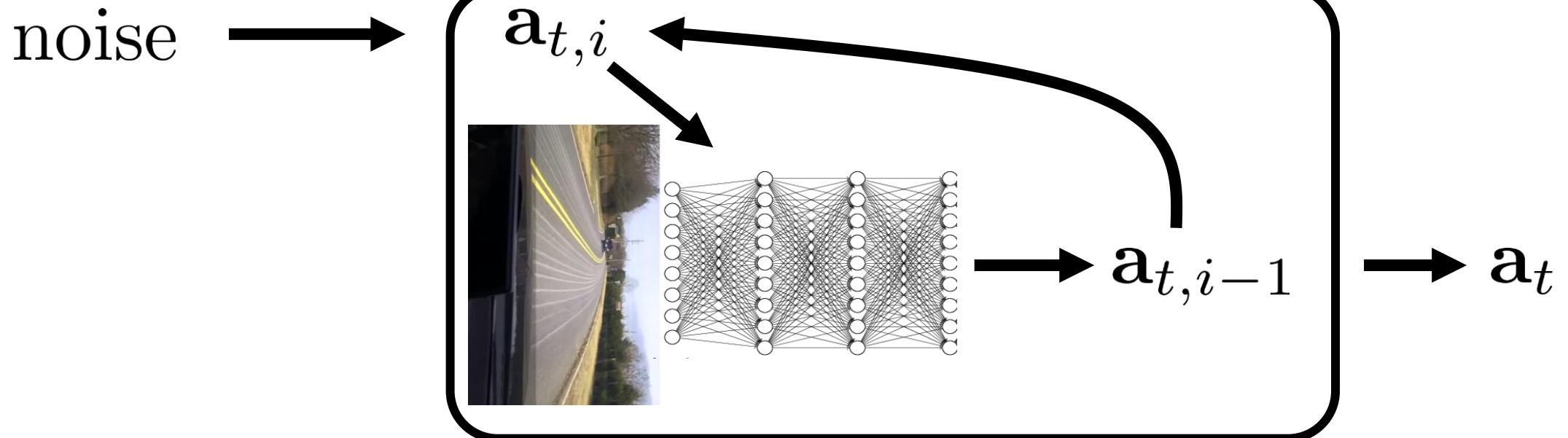
$\mathbf{a}_{t,0}$ = true action

$\mathbf{a}_{t,i+1} = \mathbf{a}_{t,i} + \text{noise}$

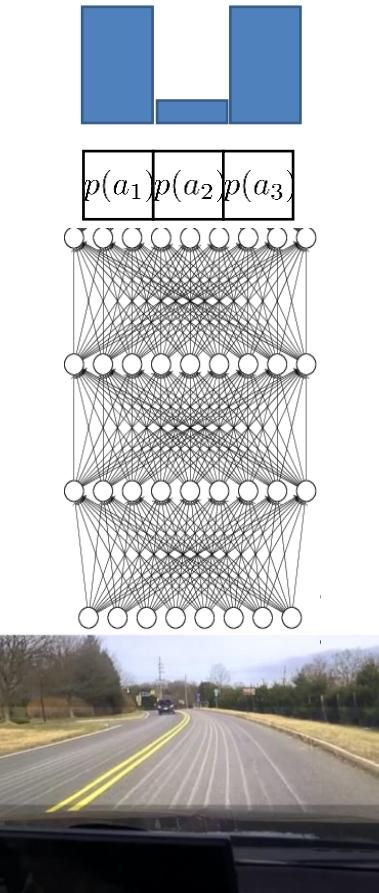
Learned network: $f(\mathbf{s}_t, \mathbf{a}_{t,i}) = \mathbf{a}_{t,i-1}$

(actually use $f(\mathbf{s}_t, \mathbf{a}_{t,i}) = \text{noise}$)

$\mathbf{a}_{t,i-1} = \mathbf{a}_{t,i} - f(\mathbf{s}_t, \mathbf{a}_{t,i})$



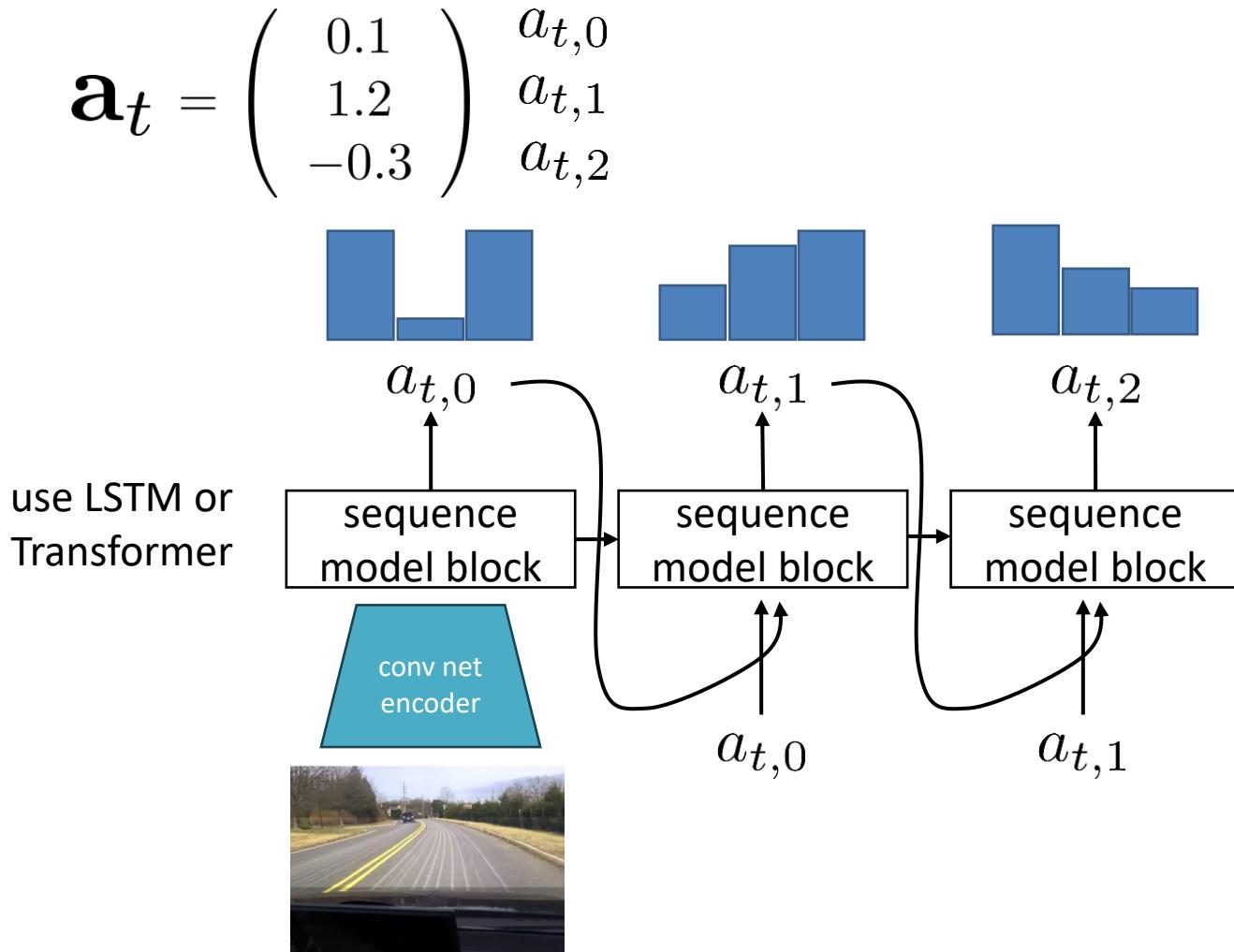
What about discretization?



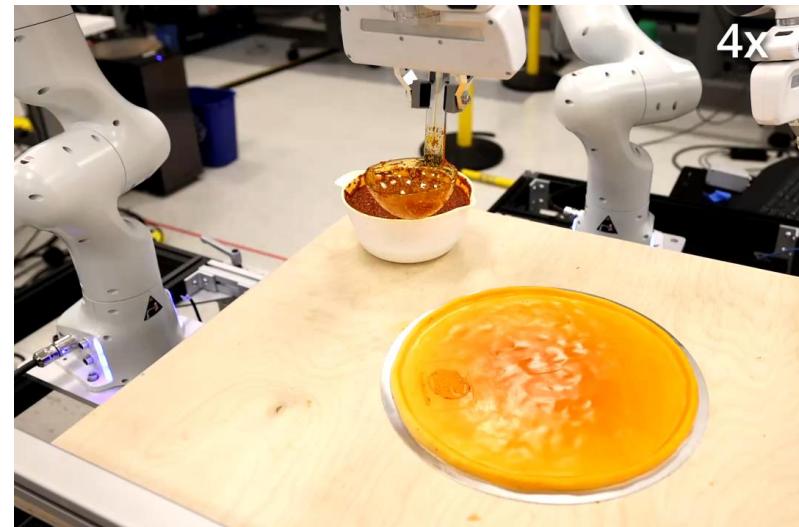
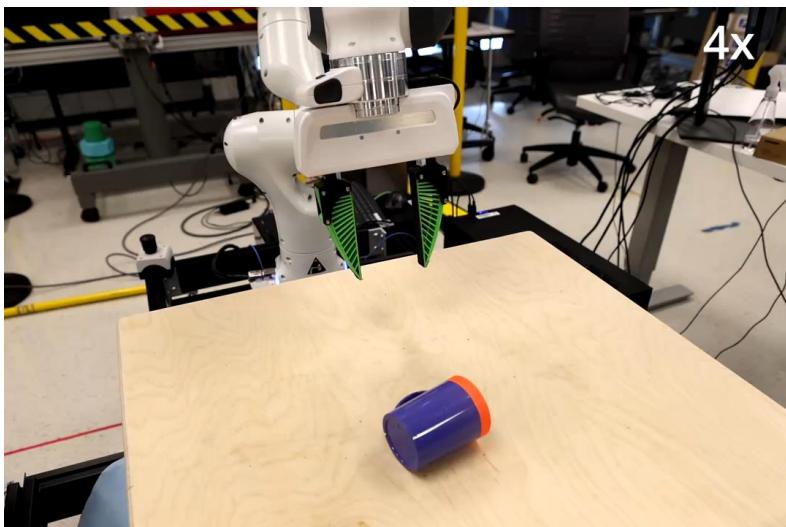
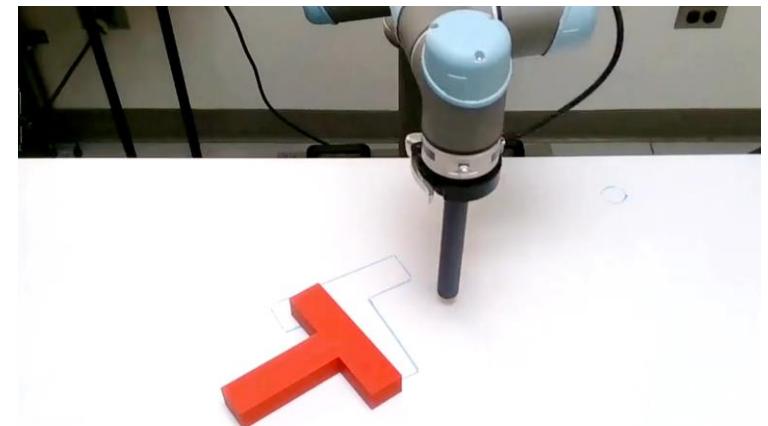
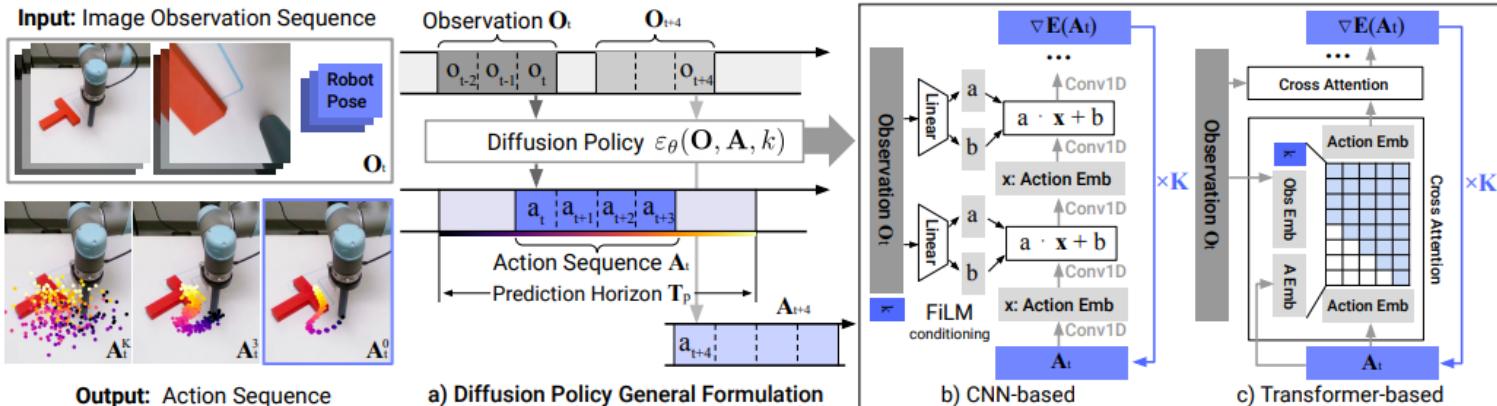
Problem: this is great for 1D actions,
but in higher dimensions, discretizing
the full space is impractical

Solution: discretize one dimension at
a time

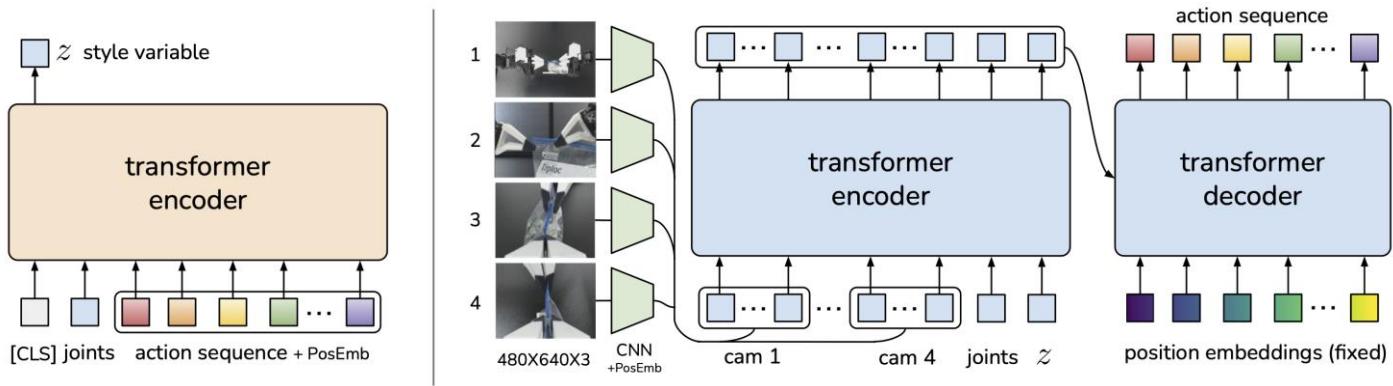
Autoregressive discretization



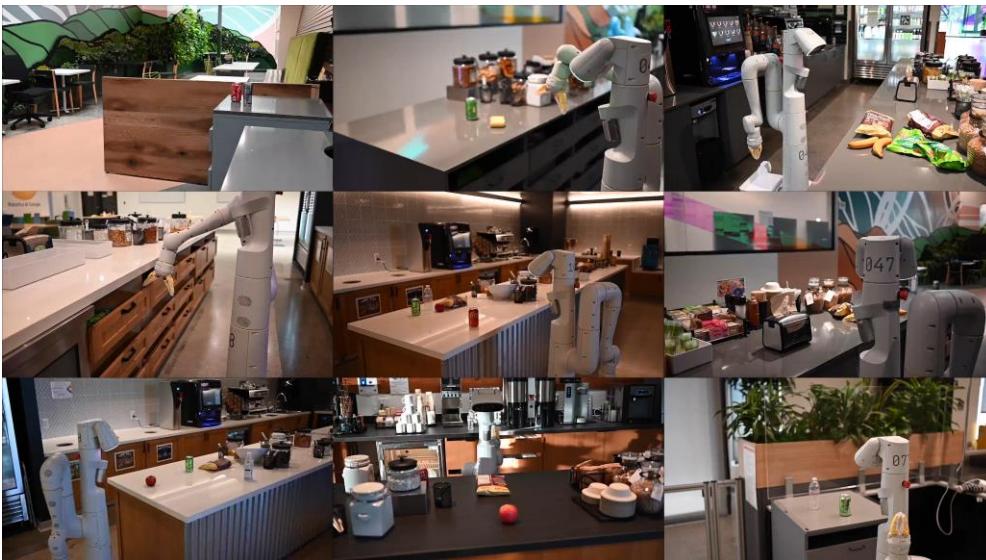
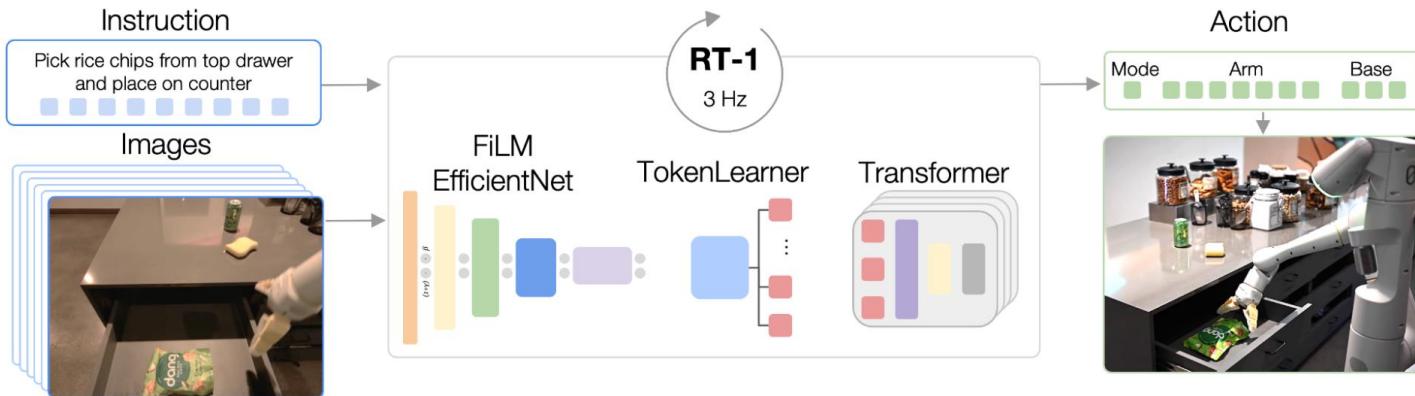
Case study 3: imitation with diffusion models



Case study 4: imitation with latent variables



Case study 5: imitation with Transformers



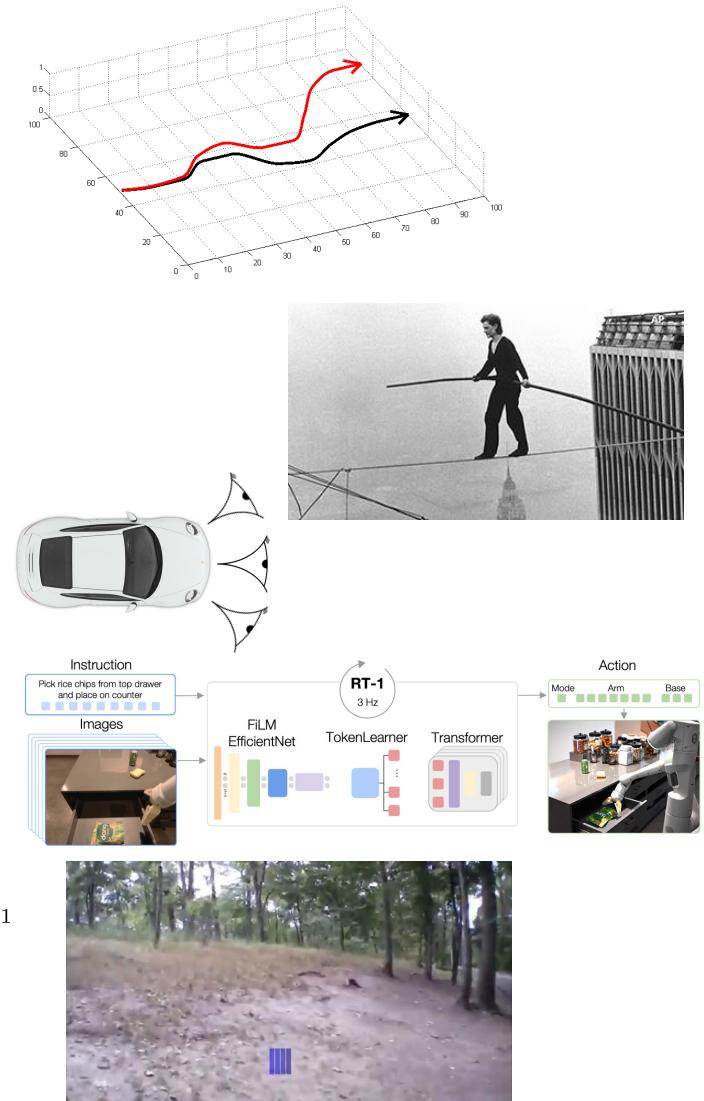
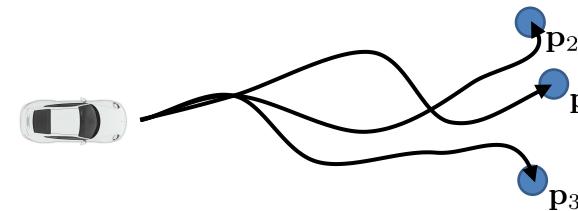
RT-1: Robotics Transformer
for Real-World Control at Scale

 Robotics at Google
 Everyday Robots
Google Research

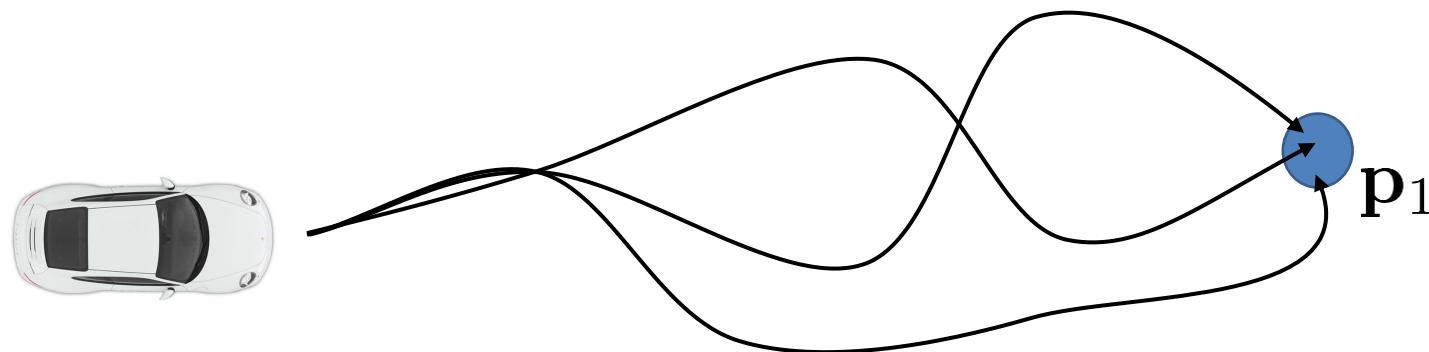
<https://robotics-transformer.github.io/>

Where are we...

- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- **We can address the problem in a few ways:**
 - Be smart about how we collect (and augment) our data
 - Use very powerful models that make very few mistakes
 - **Use multi-task learning**
 - Change the algorithm (Dagger)

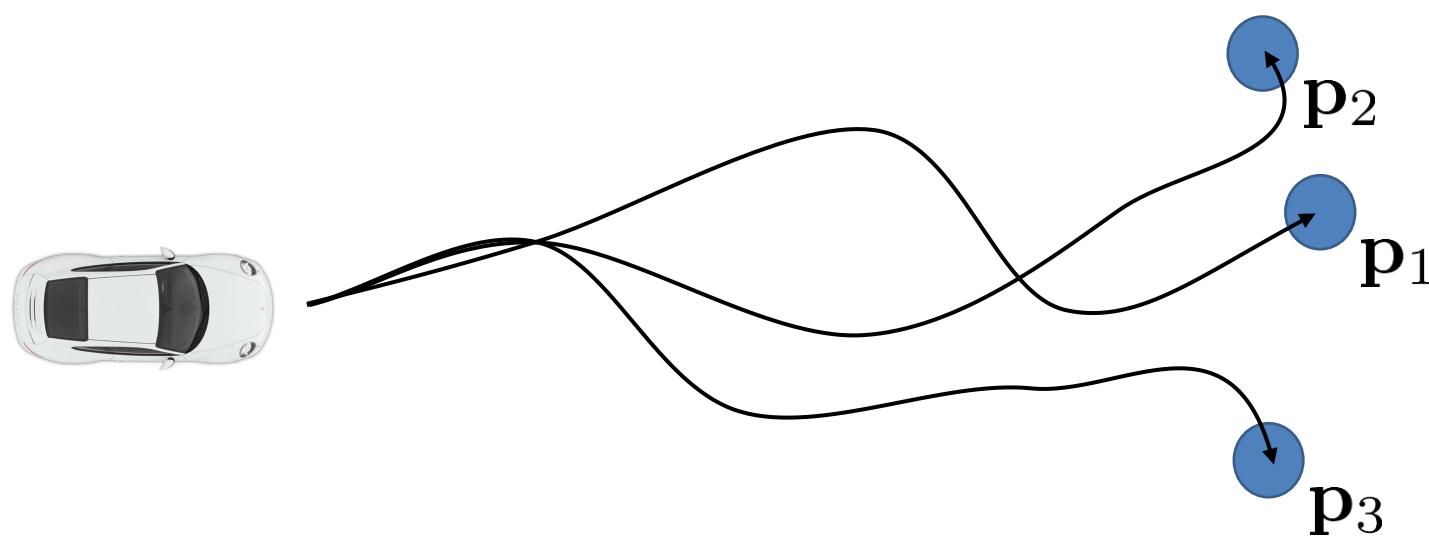


Does learning many tasks become easier?



$$\pi_{\theta}(\mathbf{a}|\mathbf{s})$$

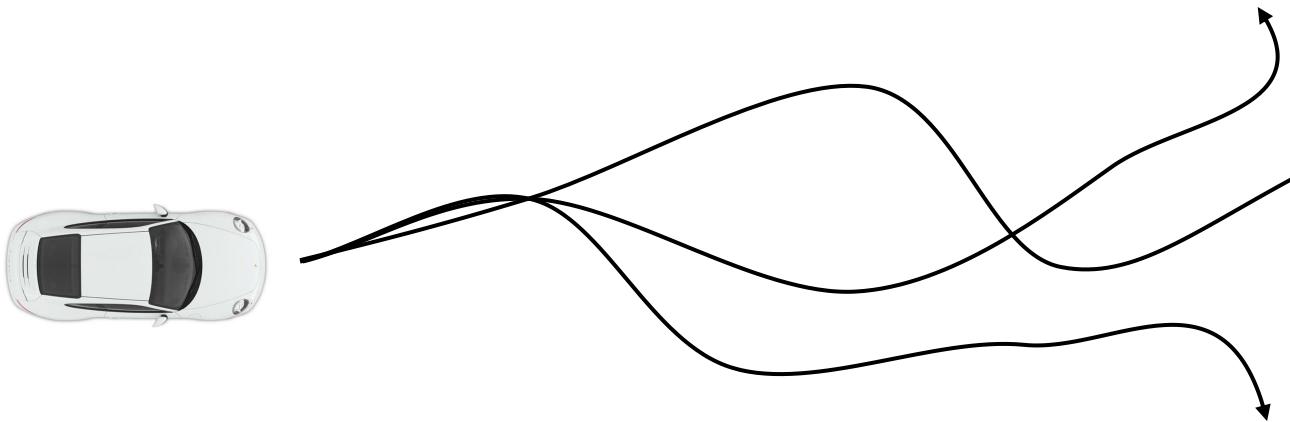
policy for reaching \mathbf{p}_1



$$\pi_{\theta}(\mathbf{a}|\mathbf{s}, \mathbf{p})$$

policy for reaching *any* \mathbf{p}

Goal-conditioned behavioral cloning



training time:

demo 1: $\{s_1, a_t, \dots, s_{T-1}, a_{T-1}, s_T\}$ ← successful demo for reaching s_T

demo 2: $\{s_1, a_t, \dots, s_{T-1}, a_{T-1}, s_T\}$ learn $\pi_\theta(a|s, g)$ ← goal state

demo 3: $\{s_1, a_t, \dots, s_{T-1}, a_{T-1}, s_T\}$

We see distributional shift in **two** places here!

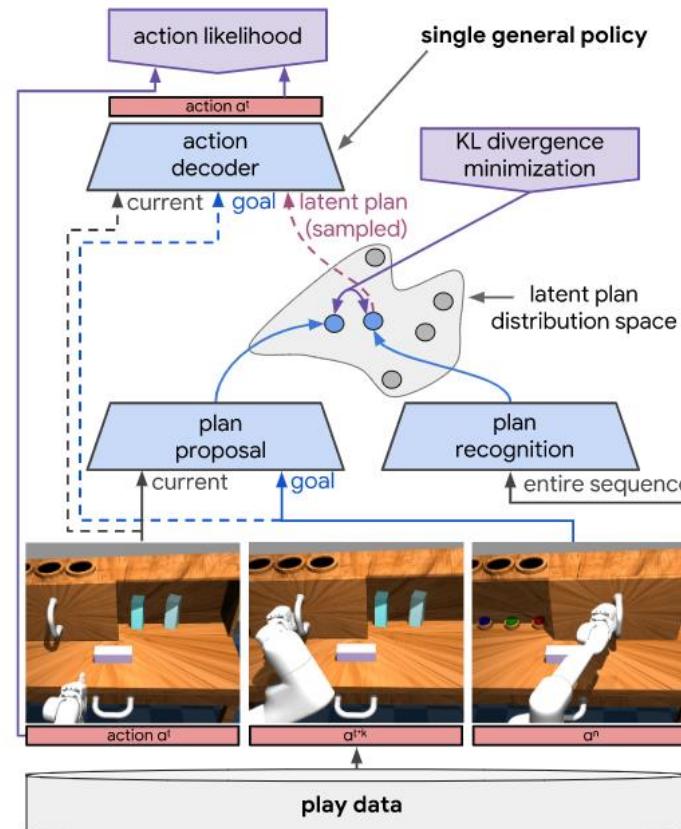
for each demo $\{s_1^i, a_1^i, \dots, s_{T-1}^i, a_{T-1}^i, s_T^i\}$

Can you figure out what the second place is?

maximize $\log \pi_\theta(a_t^i | s_t^i, g = s_T^i)$

Learning Latent Plans from Play

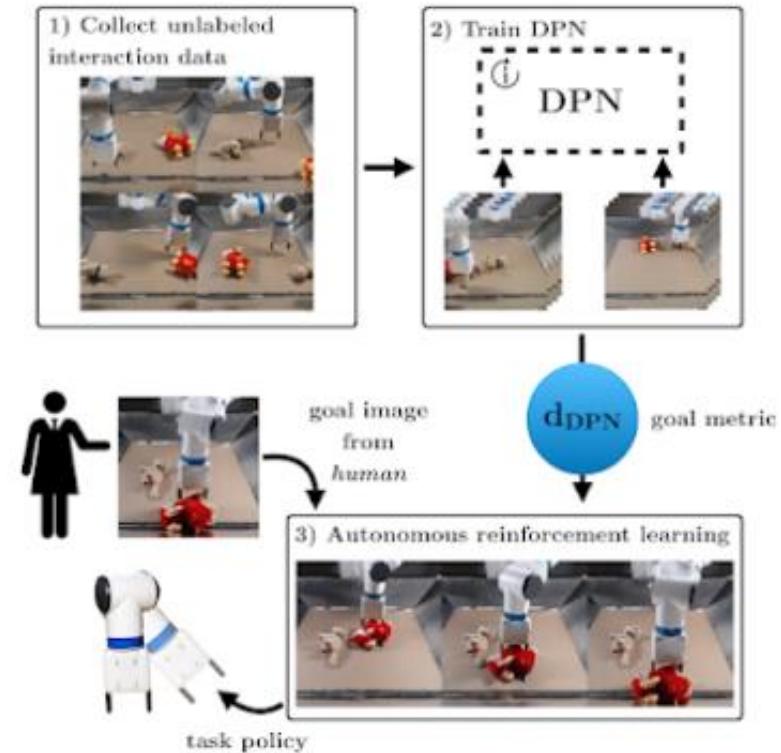
COREY LYNCH MOHI KHANSARI TED XIAO VIKASH KUMAR JONATHAN TOMPSON SERGEY LEVINE PIERRE SERMANET
Google Brain Google X Google Brain Google Brain Google Brain Google Brain Google Brain



Unsupervised Visuomotor Control through Distributional Planning Networks

Tianhe Yu, Gleb Shevchuk, Dorsa Sadigh, Chelsea Finn

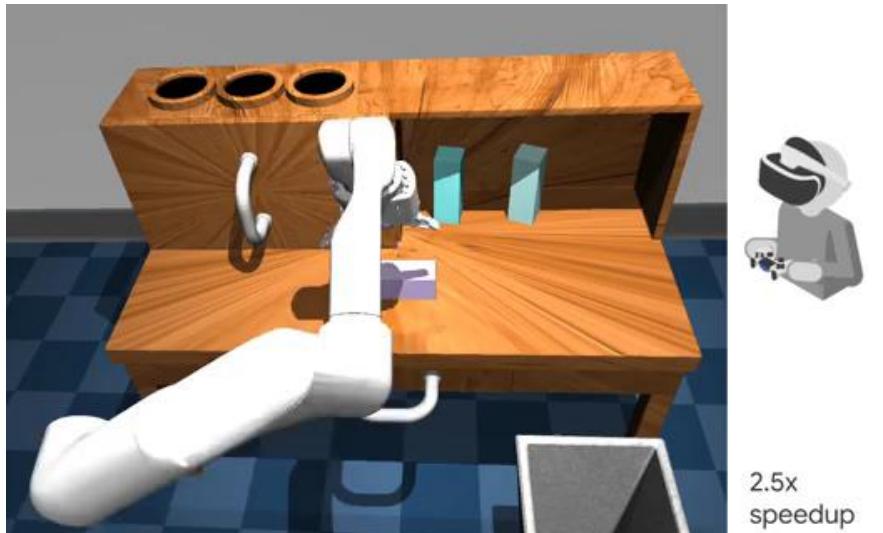
Stanford University



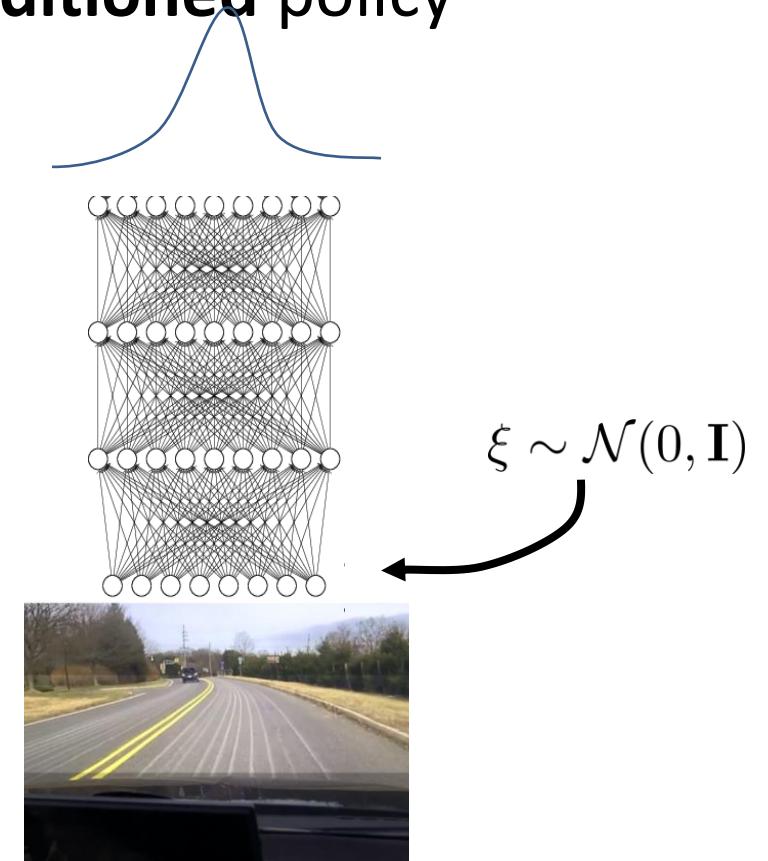
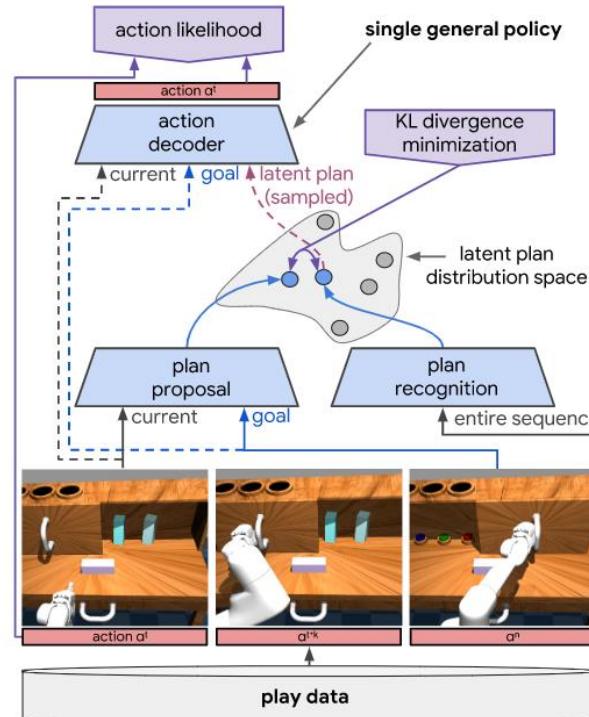
Learning Latent Plans from Play

COREY LYNCH MOHI KHANSARI TED XIAO VIKASH KUMAR JONATHAN TOMPSON SERGEY LEVINE PIERRE SERMANET
Google Brain Google X Google Brain Google Brain Google Brain Google Brain Google Brain

1. Collect data



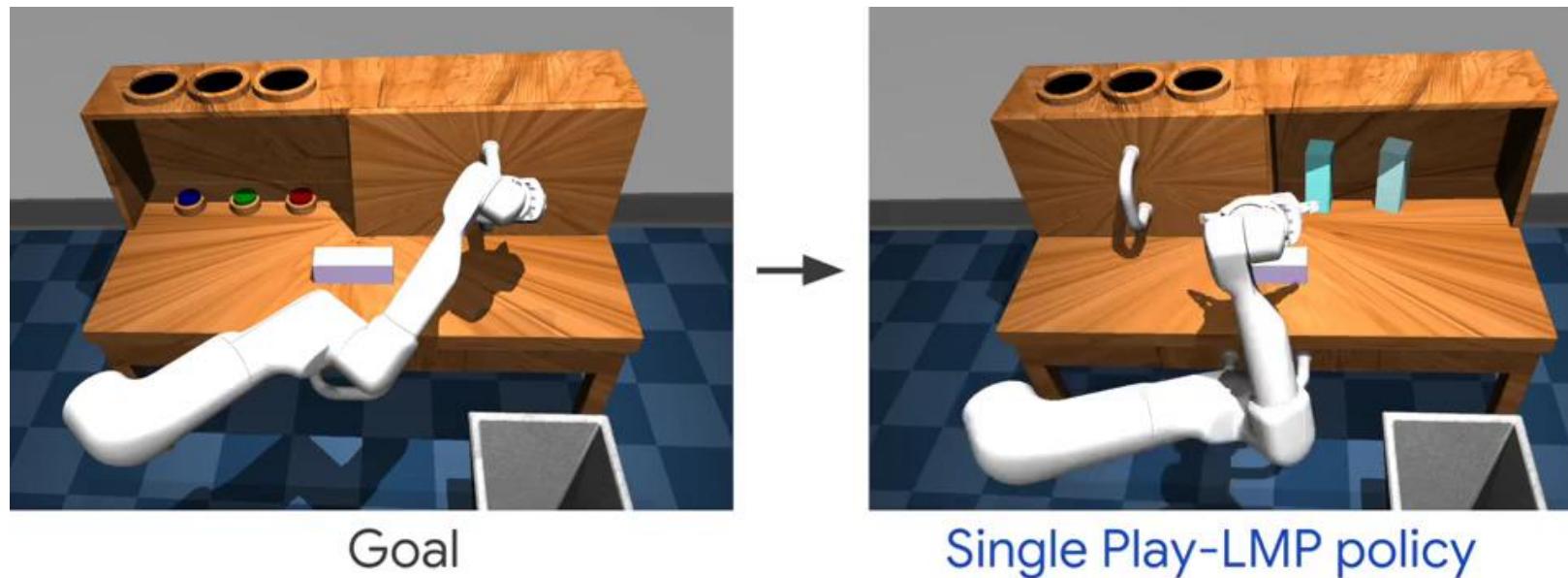
2. Train goal conditioned policy



Learning Latent Plans from Play

COREY LYNCH MOHI KHANSARI TED XIAO VIKASH KUMAR JONATHAN TOMPSON SERGEY LEVINE PIERRE SERMANET
Google Brain Google X Google Brain Google Brain Google Brain Google Brain Google Brain

3. Reach goals



Going beyond just imitation?

Learning to Reach Goals via Iterated Supervised Learning

Dibya Ghosh*
UC Berkeley

Abhishek Gupta*
UC Berkeley

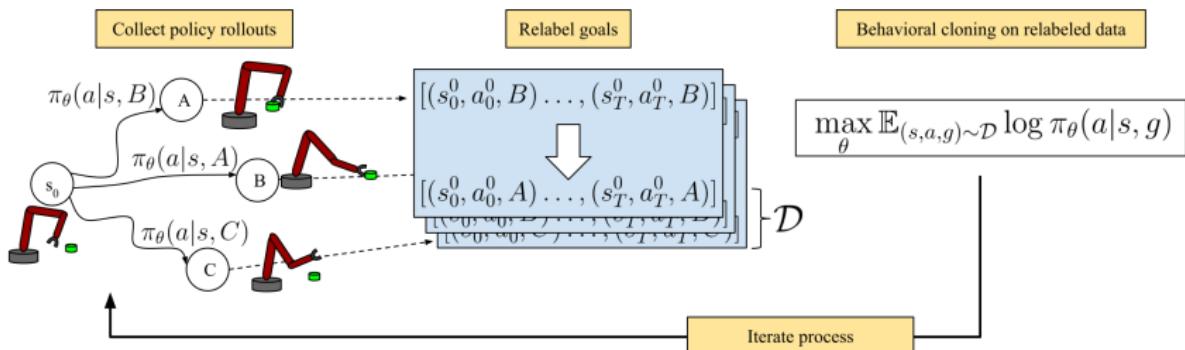
Ashwin Reddy
UC Berkeley

Justin Fu
UC Berkeley

Coline Devin
UC Berkeley

Benjamin Eysenbach
Carnegie Mellon University

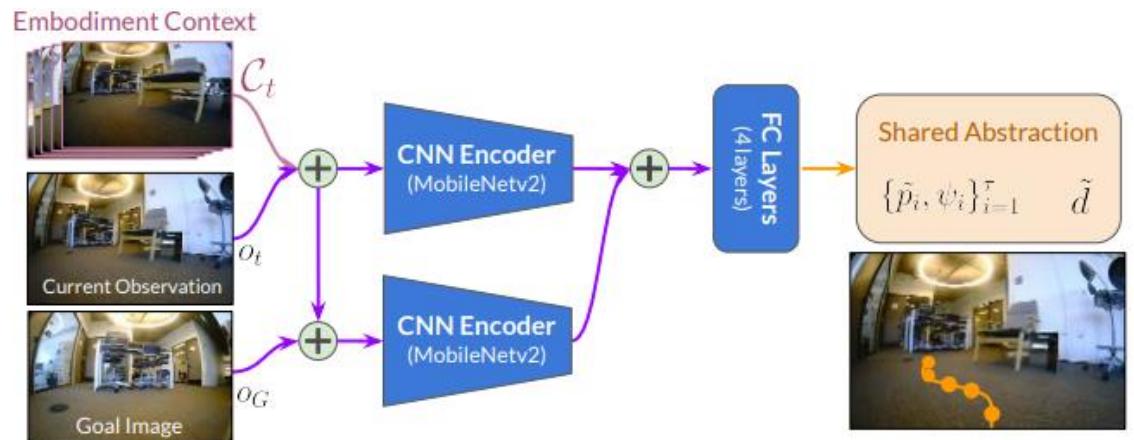
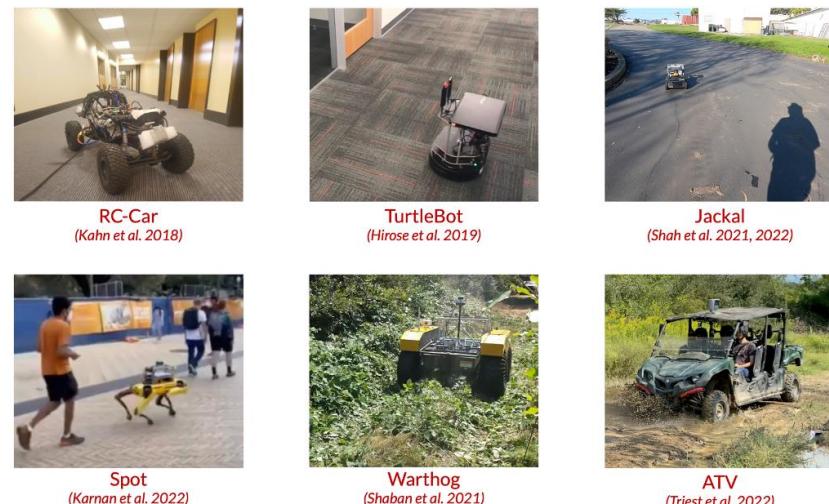
Sergey Levine
UC Berkeley



- Start with a **random** policy
- Collect data with **random** goals
- Treat this data as “demonstrations” for the goals that were reached
- Use this to improve the policy
- Repeat

Goal-conditioned BC at a huge scale

Dataset	Platform	Speed	Amt.	Environment
1 GoStanford [26]	TurtleBot2	0.5m/s	14h	office
2 RECON [32]	Jackal	1m/s	25h	off-road
3 CoryHall [35]	RC Car	1.2m/s	2h	hallways
4 Berkeley [33]	Jackal	2m/s	4h	suburban
5 SCAND-S [36]	Spot	1.5m/s	8h	sidewalks
6 SCAND-J [36]	Jackal	2m/s	1h	sidewalks
7 Seattle [37]	Warthog	5m/s	1h	off-road
8 TartanDrive [38]	ATV	10m/s	5h	off-road
Ours		60h		



Also related (for later...)

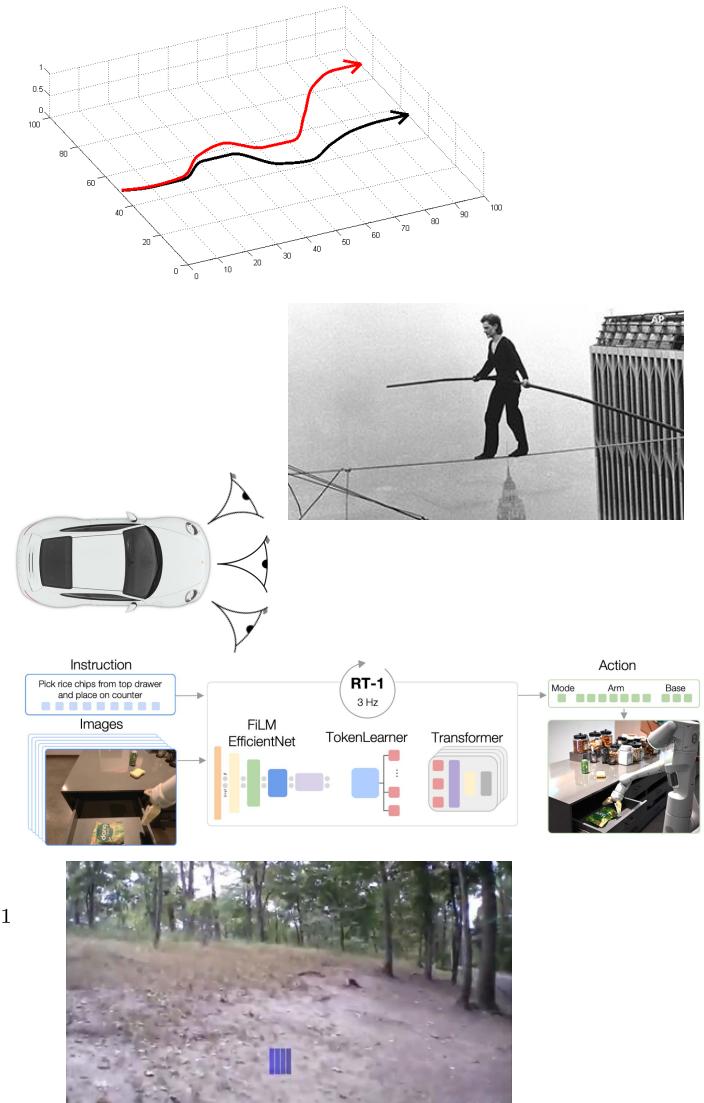
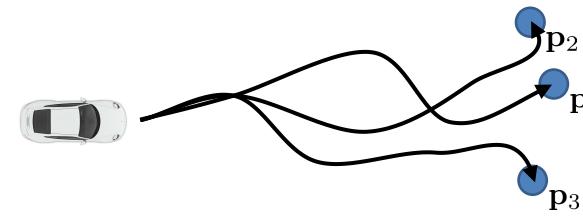
Hindsight Experience Replay

Marcin Andrychowicz*, **Filip Wolski**, **Alex Ray**, **Jonas Schneider**, **Rachel Fong**,
Peter Welinder, **Bob McGrew**, **Josh Tobin**, **Pieter Abbeel†**, **Wojciech Zaremba†**
OpenAI

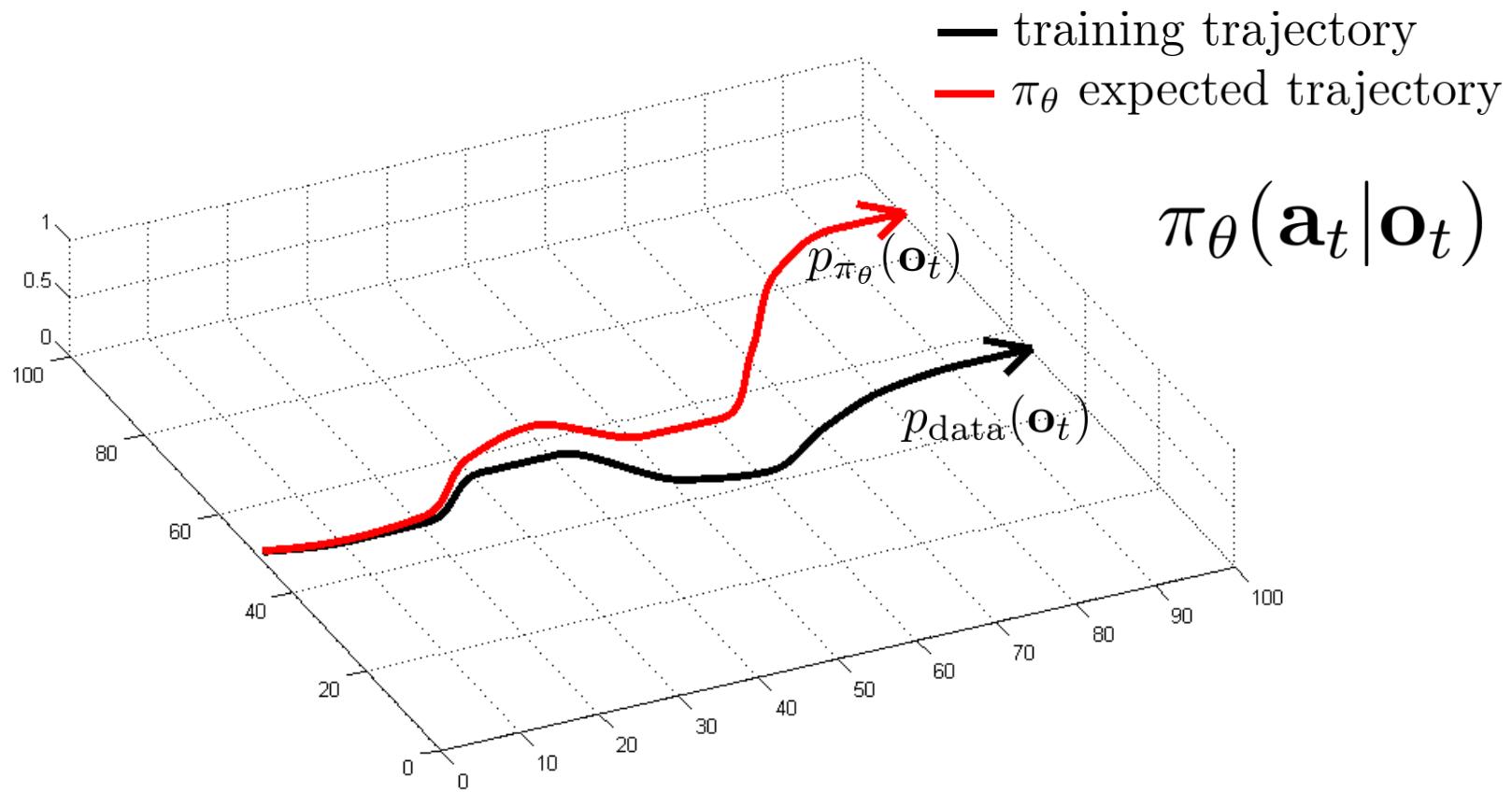
- Similar principle but with reinforcement learning
- This will make more sense later once we cover off-policy value-based RL algorithms
- Worth mentioning because this idea has been used widely outside of imitation (and was arguably first proposed there)

Where are we...

- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- **We can address the problem in a few ways:**
 - Be smart about how we collect (and augment) our data
 - Use very powerful models that make very few mistakes
 - Use multi-task learning
 - Change the algorithm (DAgger)



Can we make it work more often?



can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

Can we make it work more often?

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

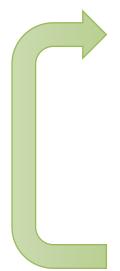
idea: instead of being clever about $p_{\pi_\theta}(\mathbf{o}_t)$, be clever about $p_{\text{data}}(\mathbf{o}_t)$!

DAgger: Dataset Aggregation

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$

but need labels \mathbf{a}_t !

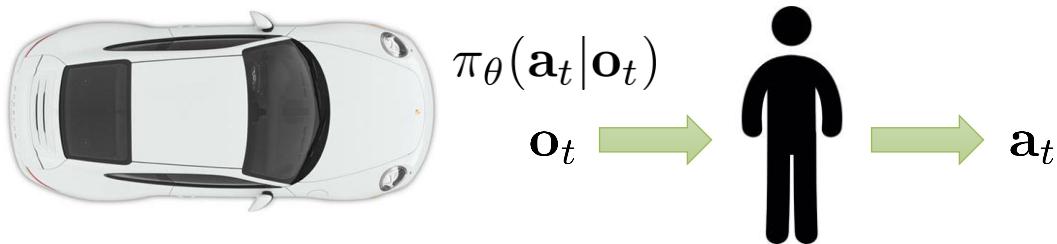
- 
1. train $\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \dots, \mathbf{o}_N, \mathbf{a}_N\}$
 2. run $\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$
 3. Ask human to label \mathcal{D}_π with actions \mathbf{a}_t
 4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

DAgger Example



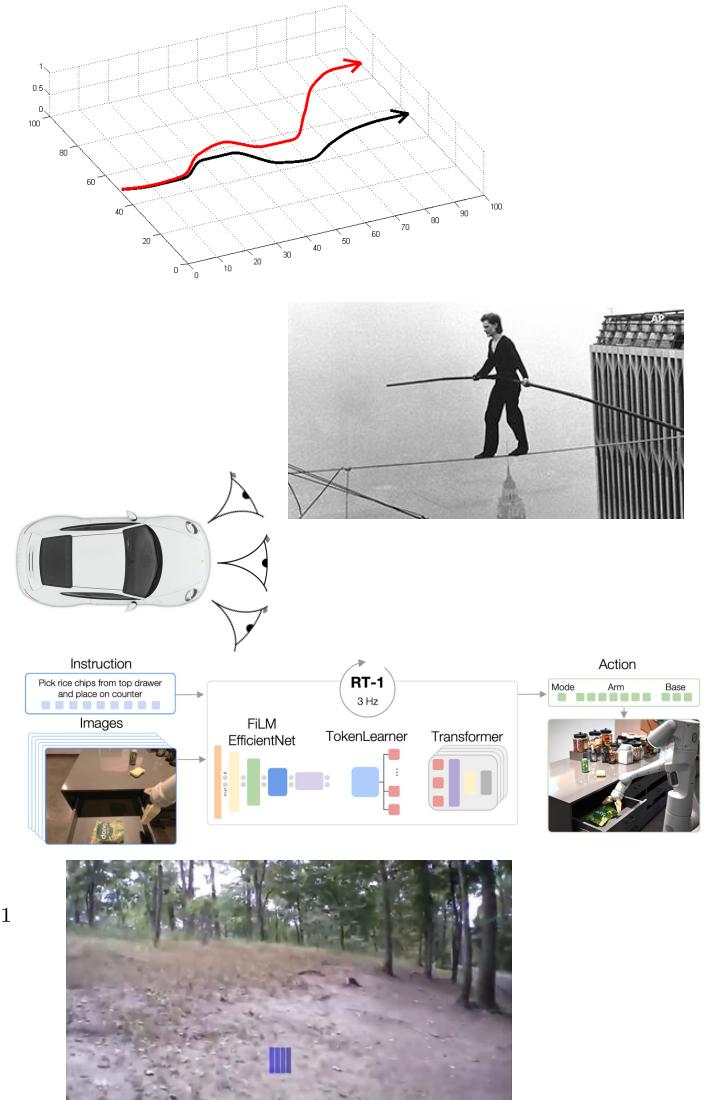
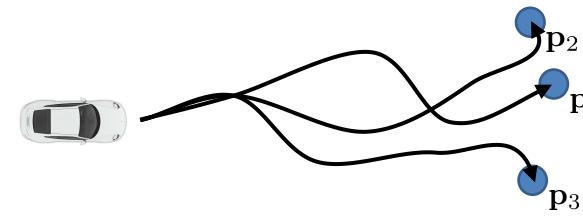
What's the problem?

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \dots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$
3. Ask human to label \mathcal{D}_π with actions \mathbf{a}_t
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$



Recap

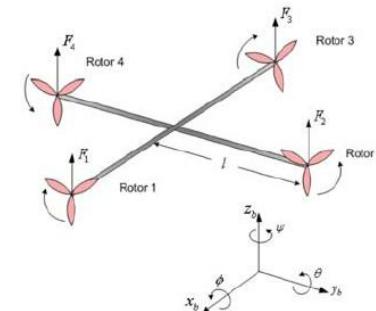
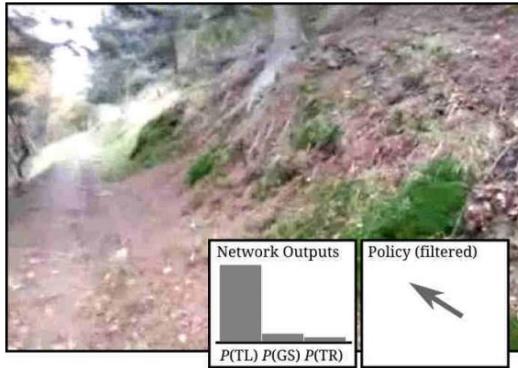
- Imitation learning via behavioral cloning is not guaranteed to work
 - This is **different** from supervised learning
 - The reason: i.i.d. assumption does not hold!
- We can formalize **why** this is and do a bit of theory
- We can address the problem in a few ways:
 - Be smart about how we collect (and augment) our data
 - Use very powerful models that make very few mistakes
 - Use multi-task learning
 - Change the algorithm (DAgger)



Cost functions and reward functions,
a preview of what comes next

Imitation learning: what's the problem?

- Humans need to provide data, which is typically finite
 - Deep learning works best when data is plentiful
- Humans are not good at providing some kinds of actions



- Humans can learn autonomously; can our machines do the same?
 - Unlimited data from own experience
 - Continuous self-improvement

Terminology & notation



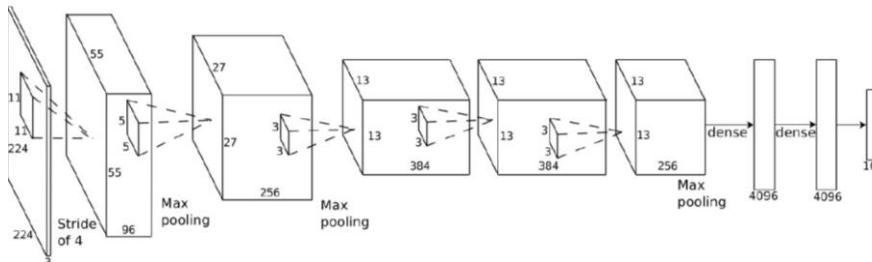
\mathbf{o}_t



s_t – state

\mathbf{o}_t – observation

\mathbf{a}_t – action



$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$



\mathbf{a}_t



$c(s_t, \mathbf{a}_t)$ – cost function

$r(s_t, \mathbf{a}_t)$ – reward function

$$\min_{\theta} E_{\mathbf{a}_T, \mathbf{o}_{[1:T]}, s_{[1:T]}} \left[\sum_t \delta(s_t, \mathbf{a}_t) \text{ [delta function by tiger]} \right]$$

Aside: notation

s_t – state

a_t – action

$r(s, a)$ – reward function



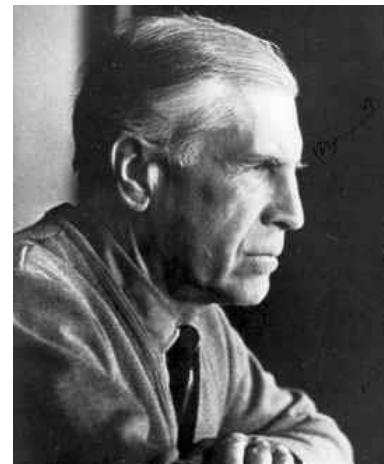
Richard Bellman

x_t – state

u_t – action

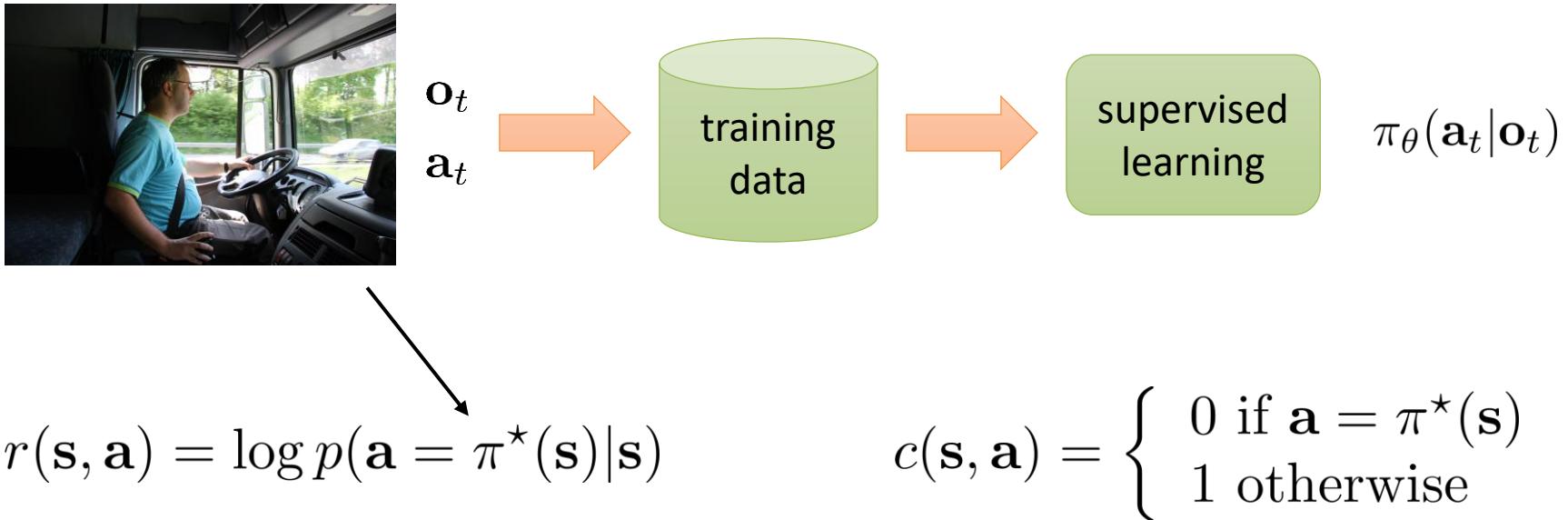
$c(x, u)$ – cost function

$$r(s, a) = -c(x, u)$$



Lev Pontryagin

A cost function for imitation?



Goal: minimize $E_{\mathbf{s}_t \sim p_{\pi_\theta}(\mathbf{s}_t)}[c(\mathbf{s}_t, \mathbf{a}_t)]$

Goal: maximize $E_{\mathbf{s}_t \sim p_{\pi_\theta}(\mathbf{s}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)]$

Imitation learning algorithms **do** maximize reward when they work well!

For a very particular choice of reward

PyTorch and Neural Nets

CS285 Deep RL

Instructor: Kyle Stachowicz



[Adapted from Marwa Abdulhai's CS285 Fa22 Slides]

PyTorch Tutorial (Colab)

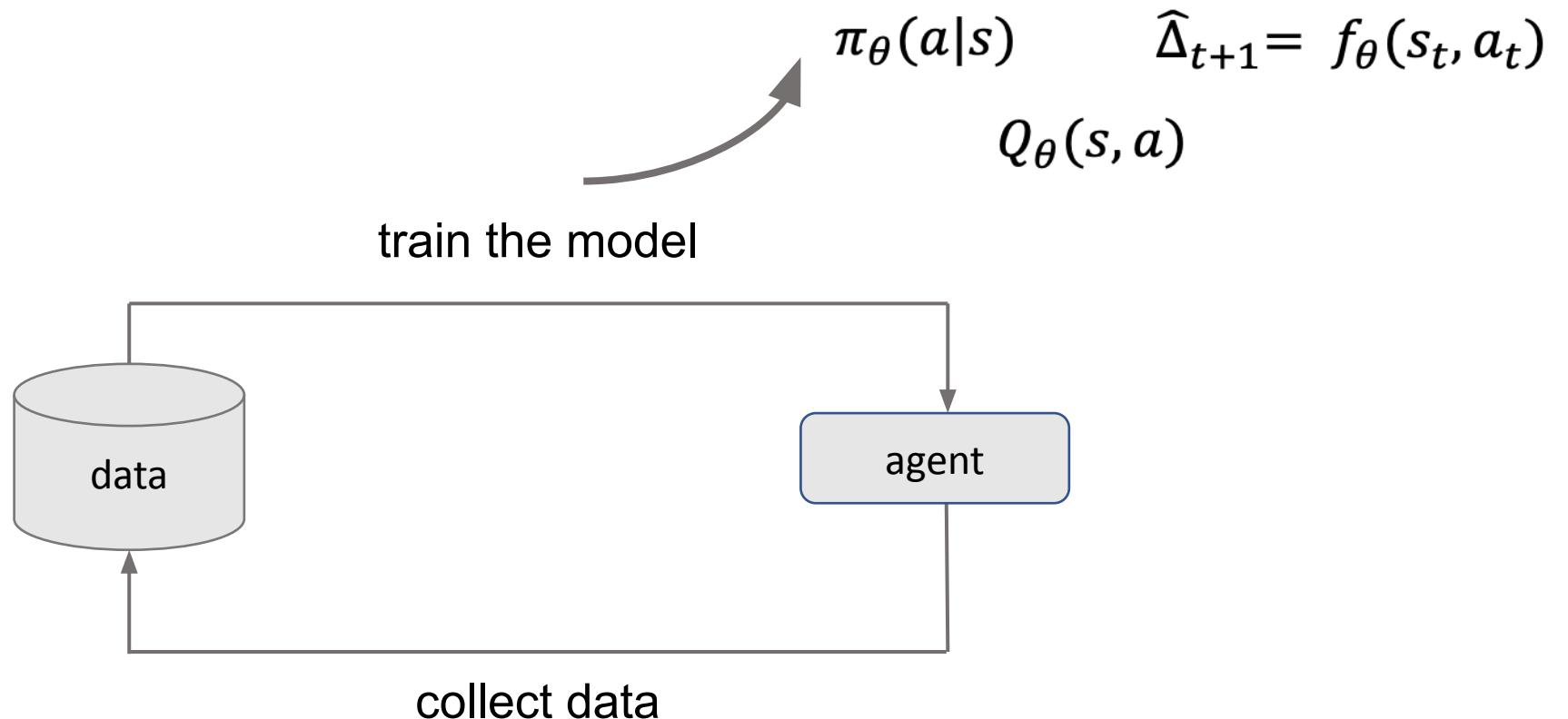


<https://colab.research.google.com/drive/12nQiv6aZHNUcFAAuTjJenDWKQbIt2Mz>

<http://bit.ly/cs285-pytorch-2023>

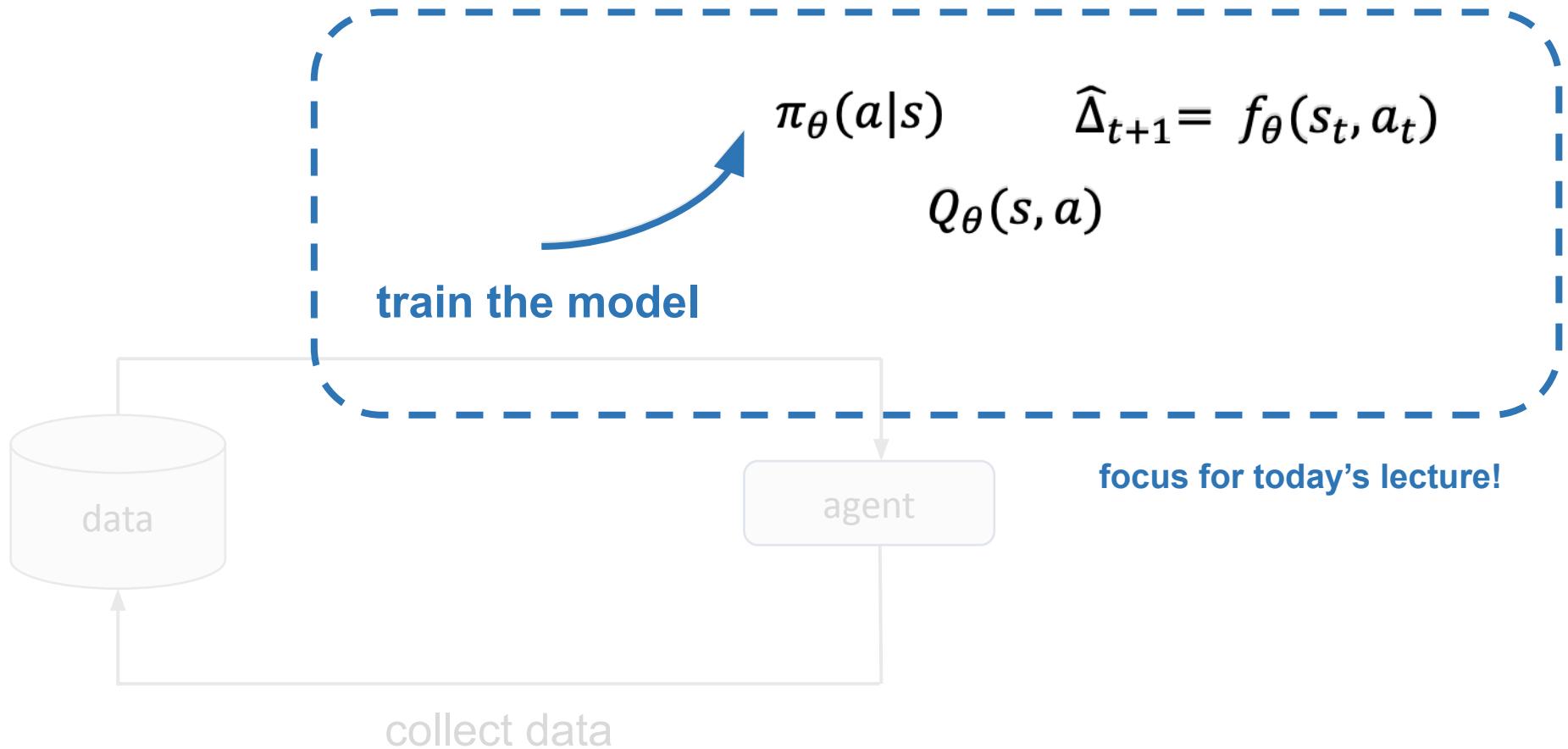
Goal of this course

Train an agent to perform useful tasks

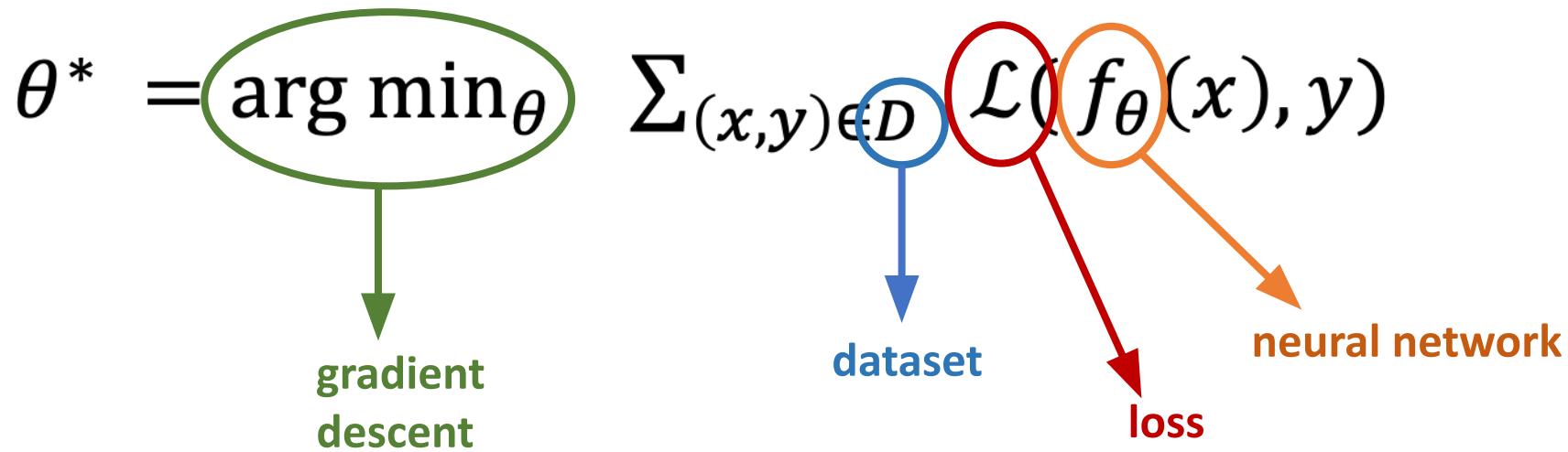


Goal of this course

Train an agent to perform useful tasks



How do train a model?



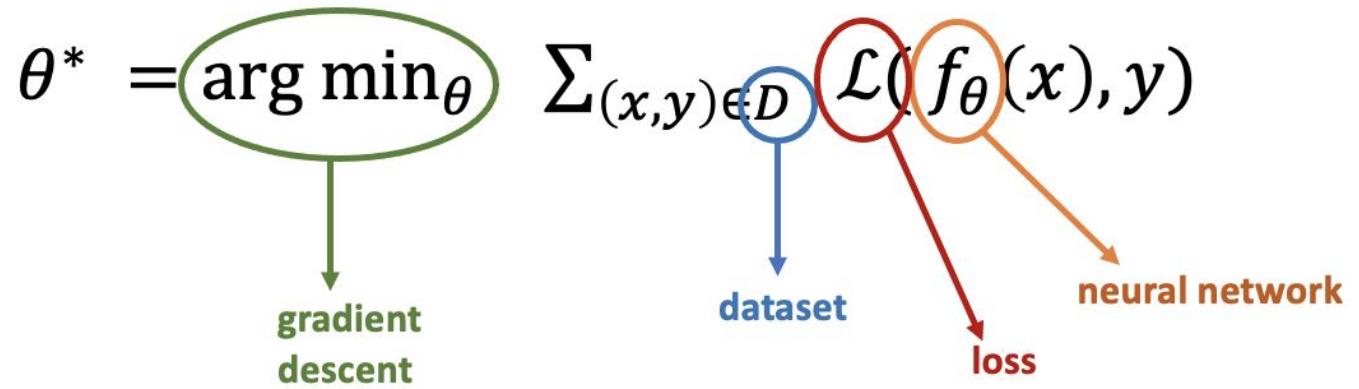
PyTorch does all of these!

What is PyTorch?

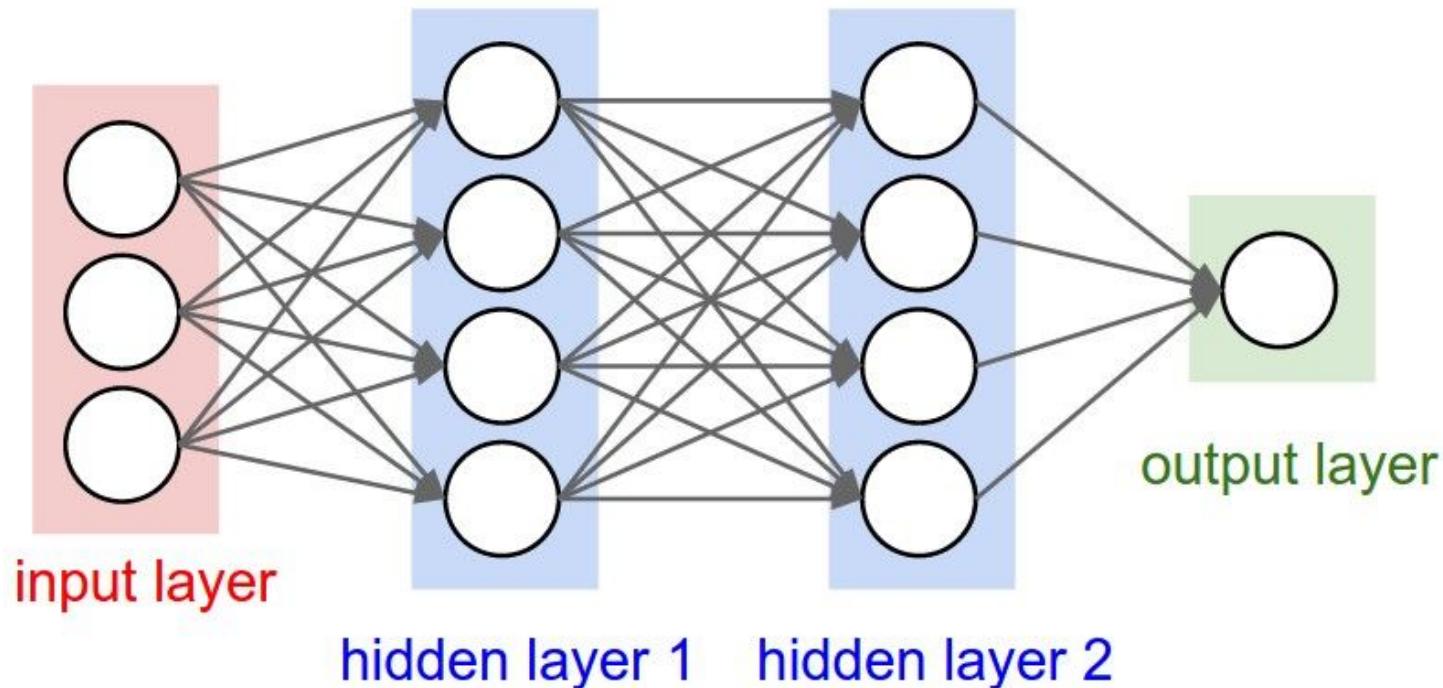
Python library for:

- Defining neural networks
- Automating computing gradients
- And more! (datasets, optimizers,

GPUs, etc.)



How does PyTorch work?



You define:

$$h_1 = \sigma(W_1 x) \quad h_2 = \sigma(W_2 h_1) \quad y = \sigma(W_3 h_2)$$

PyTorch computes:

$$\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_1} \quad \frac{\partial y}{\partial W_2} = \frac{\partial y}{\partial h_2} \frac{\partial h_2}{\partial W_1} \quad \frac{\partial y}{\partial W_3}$$



- Fast CPU implementations
- **CPU-only**
- **No autodiff**
- **Imperative**



- Fast CPU implementations
- **Allows GPU**
- **Supports autodiff**
- **Imperative**

Other features include:

- Datasets and dataloading
- Common neural network operations
- Built-in optimizers (Adam, SGD, ...)

The Basics



```
arr_a = [1, 3, 4, 5, 9]
arr_b = [9, 5, 7, 2, 5]

# Element-wise operations
list_sum = [a + b for a, b in zip(list_a, list_b)]
list_prod = [a * b for a, b in zip(list_a, list_b)]
list_doubled = [2 * a for a in list_a]

# Indexing
value = list_a[3]
list_slice = list_a[2:3]

arr_idx = [3, 2, 1]
arr_indexed = [arr_a[i] for i in arr_idx]
```



```
import numpy as np

arr_a = np.array([1, 3, 4, 5, 9])
arr_b = np.array([9, 5, 7, 2, 5])

# Element-wise operations
arr_sum = a + b
arr_prod = a * b
arr_doubled = 2 * a

# Indexing
value = arr_a[3]
arr_slice = arr_a[2:3]

arr_idx = np.array([3, 2, 1])
arr_indexed = arr_a[arr_idx]
```



```
import torch

tensor_a = torch.tensor([1, 3, 4, 5, 9])
tensor_b = torch.tensor([9, 5, 7, 2, 5])

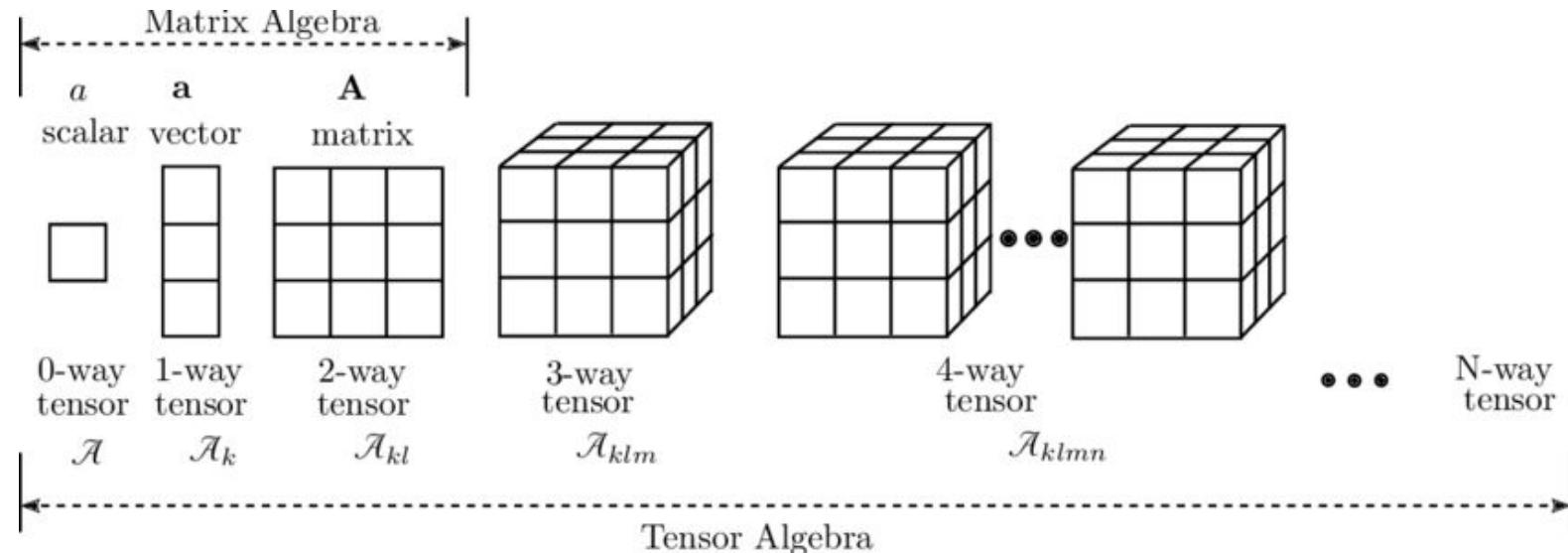
# Element-wise operations
tensor_sum = tensor_a + tensor_b
tensor_prod = tensor_a * tensor_b
tensor_doubled = 2 * tensor_a

# Indexing
value = tensor_a[3]
tensor_slice = tensor_a[2:3]

tensor_idx = torch.tensor([3, 2, 1])
tensor_indexed = tensor_a[tensor_idx]
```

100x faster!

Multidimensional Arrays



Scalar



Vector

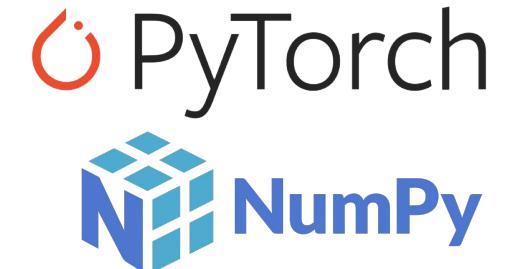


Matrix

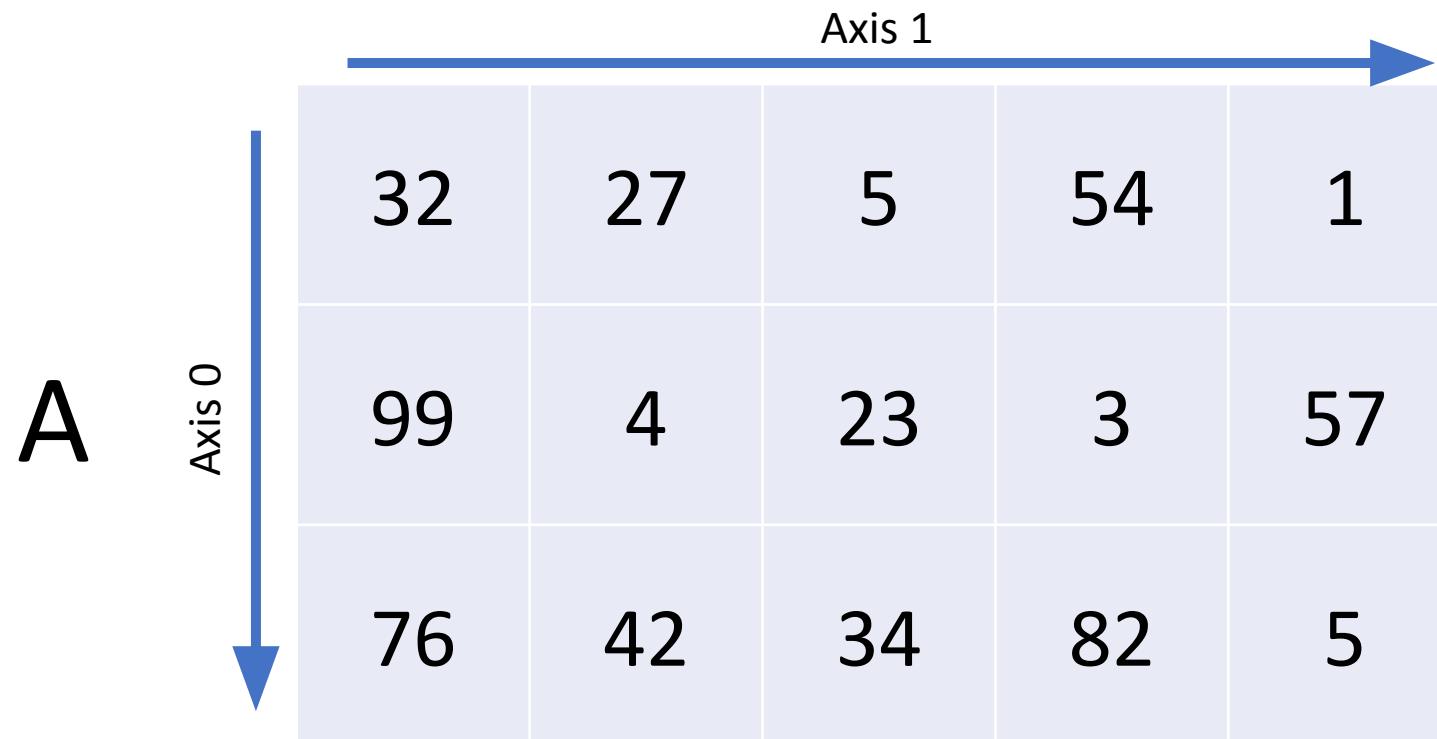


Tensor

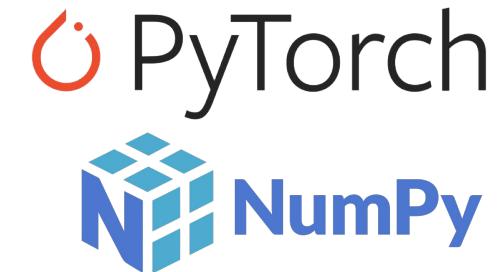




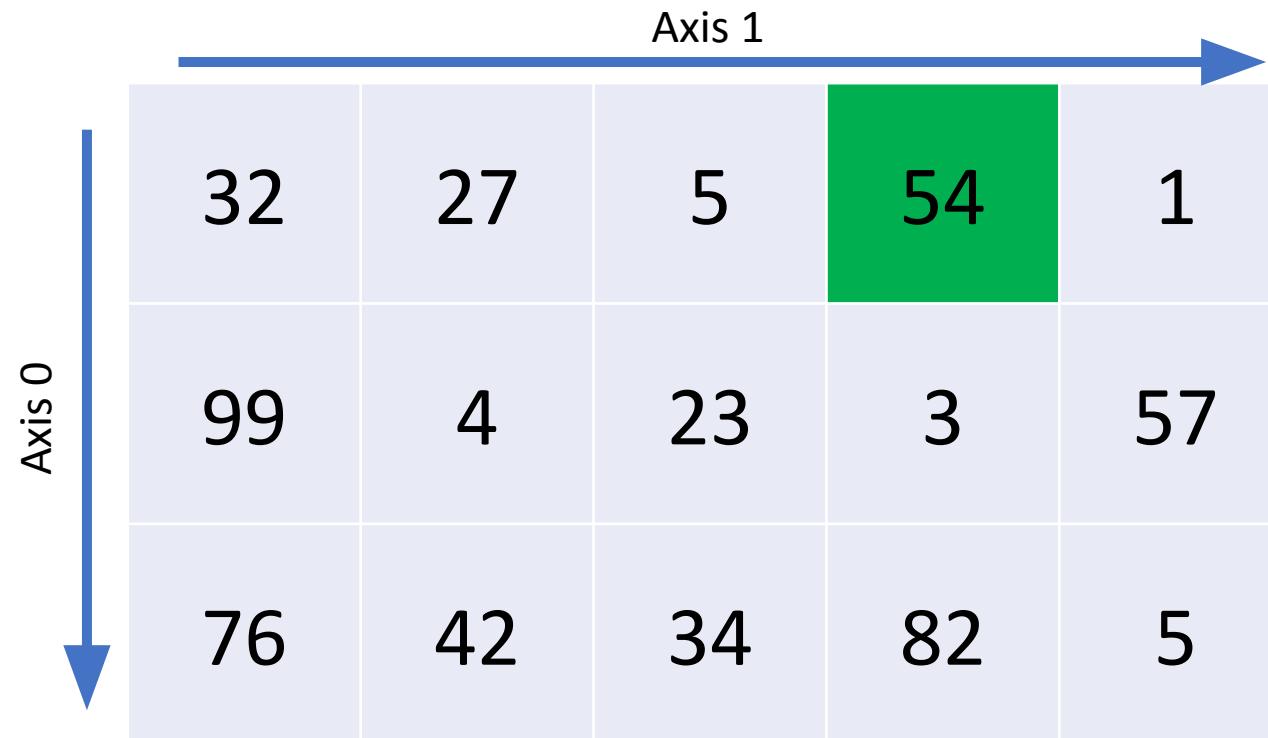
Multidimensional Indexing



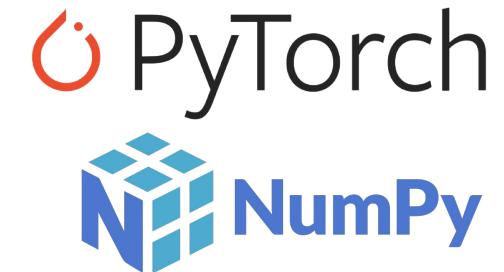
`A.shape == (3, 5)`



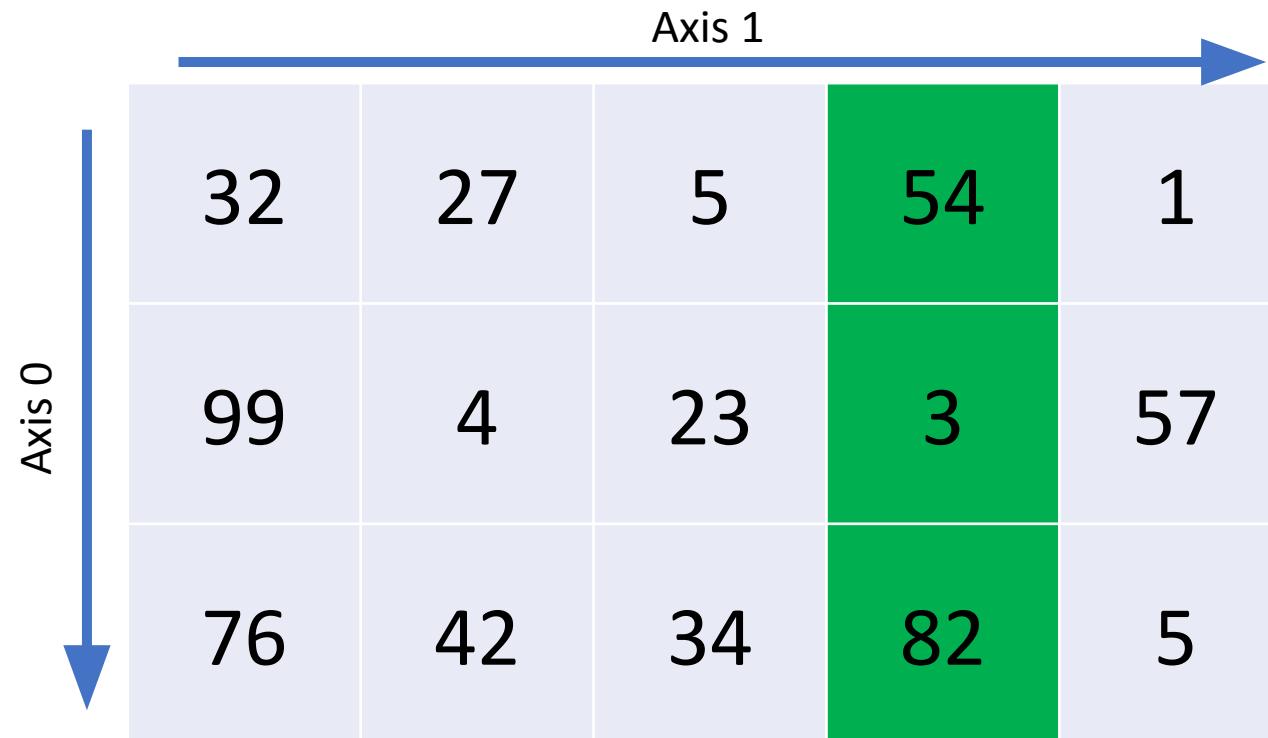
Multidimensional Indexing



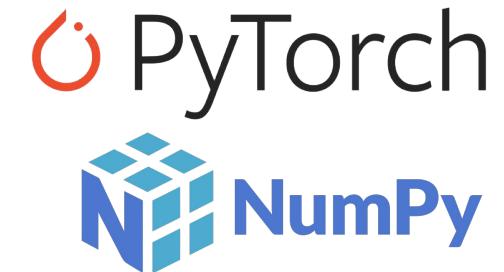
$A[0, 3]$



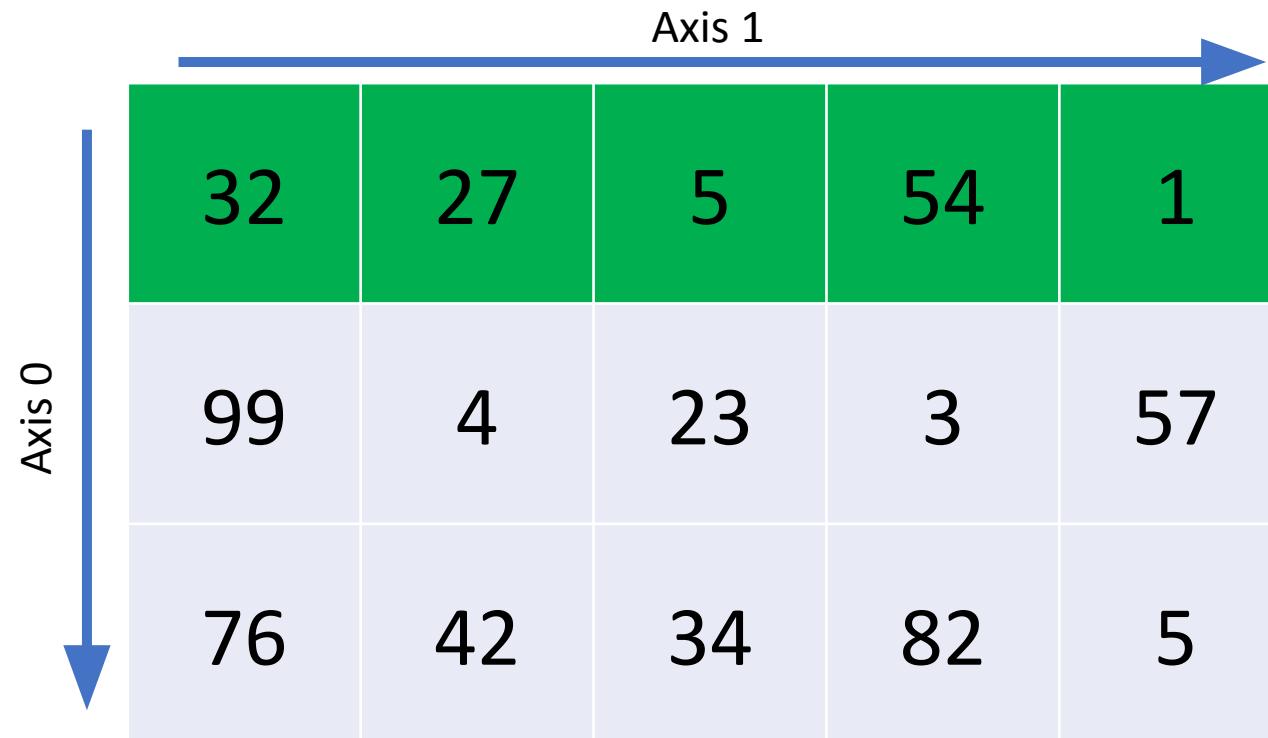
Multidimensional Indexing



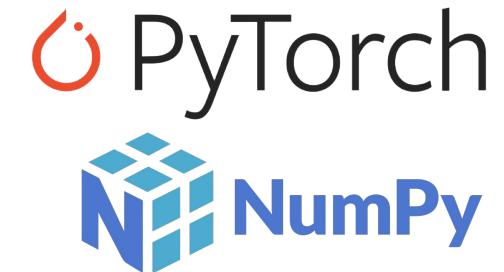
$A[:, 3]$



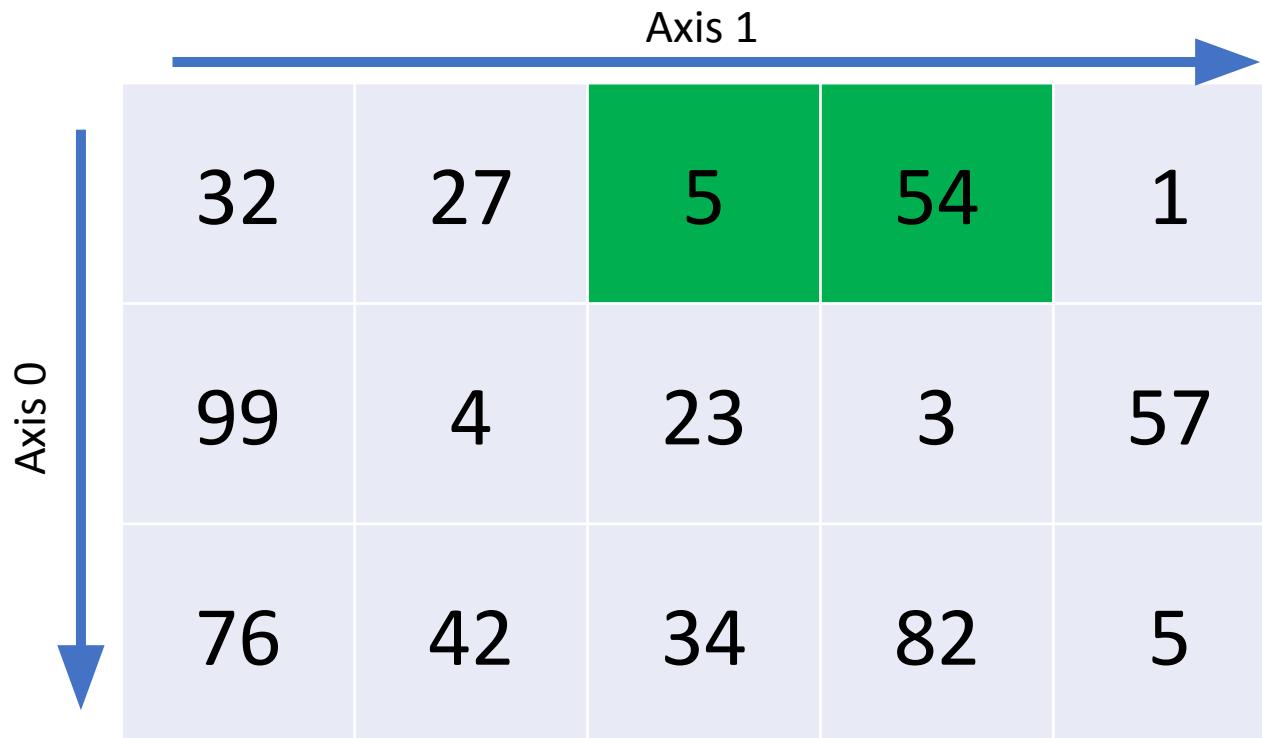
Multidimensional Indexing



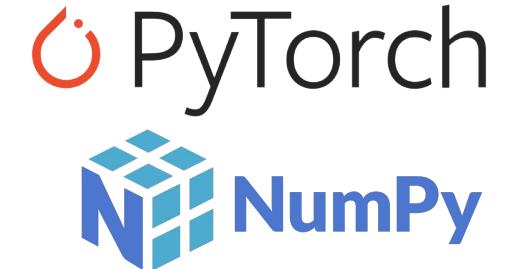
$A[0, :]$



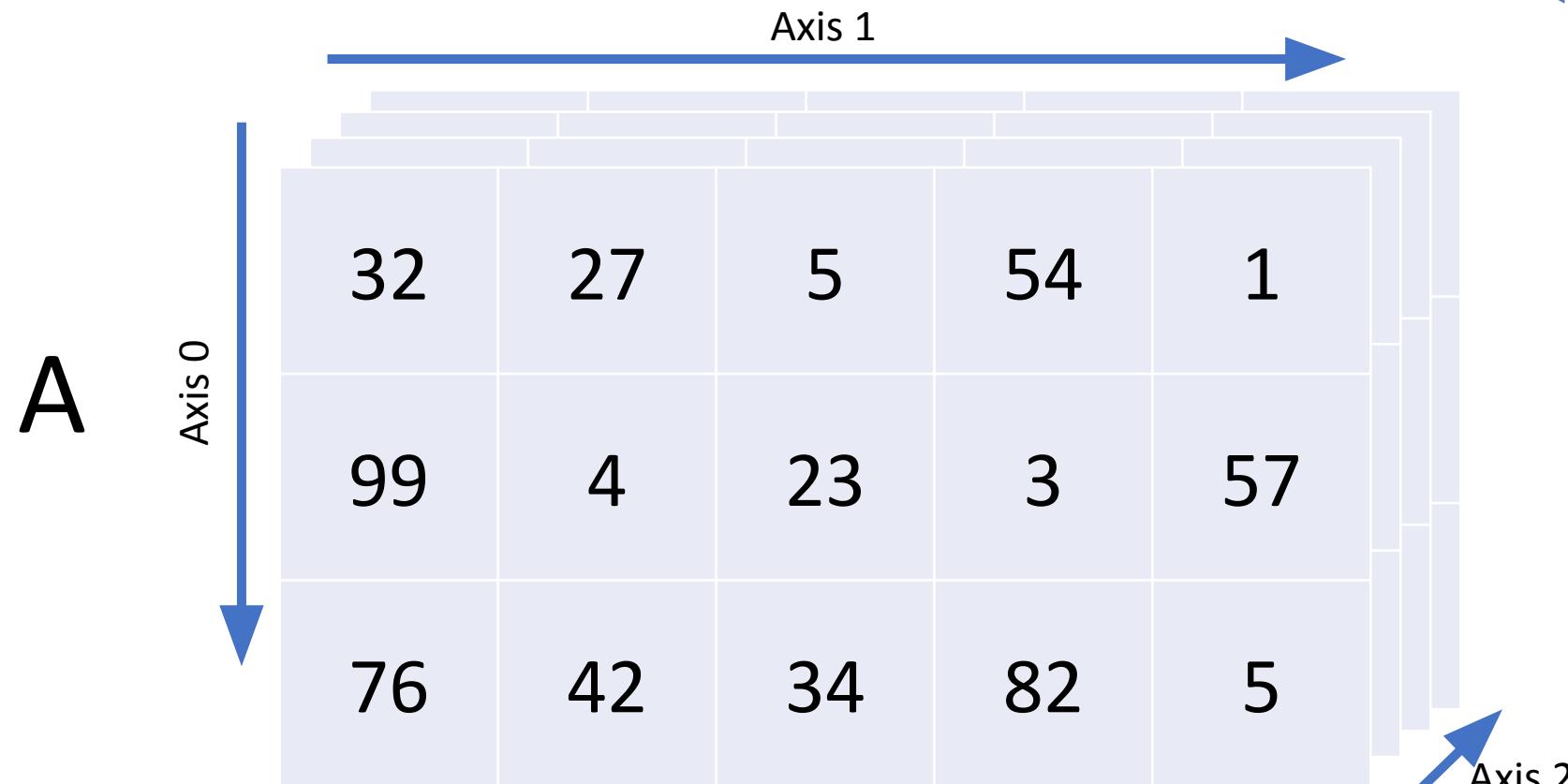
Multidimensional Indexing



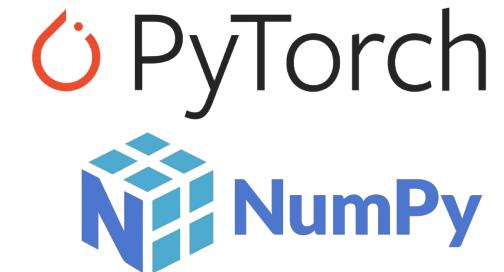
$A[0, 2:4]$



Multidimensional Indexing



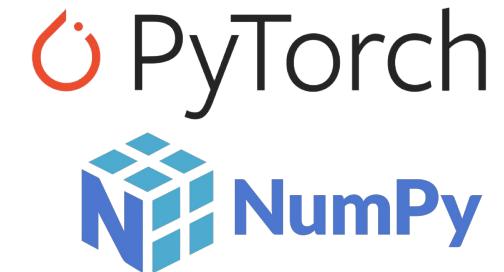
$A.\text{shape} == (3, 5, 4)$



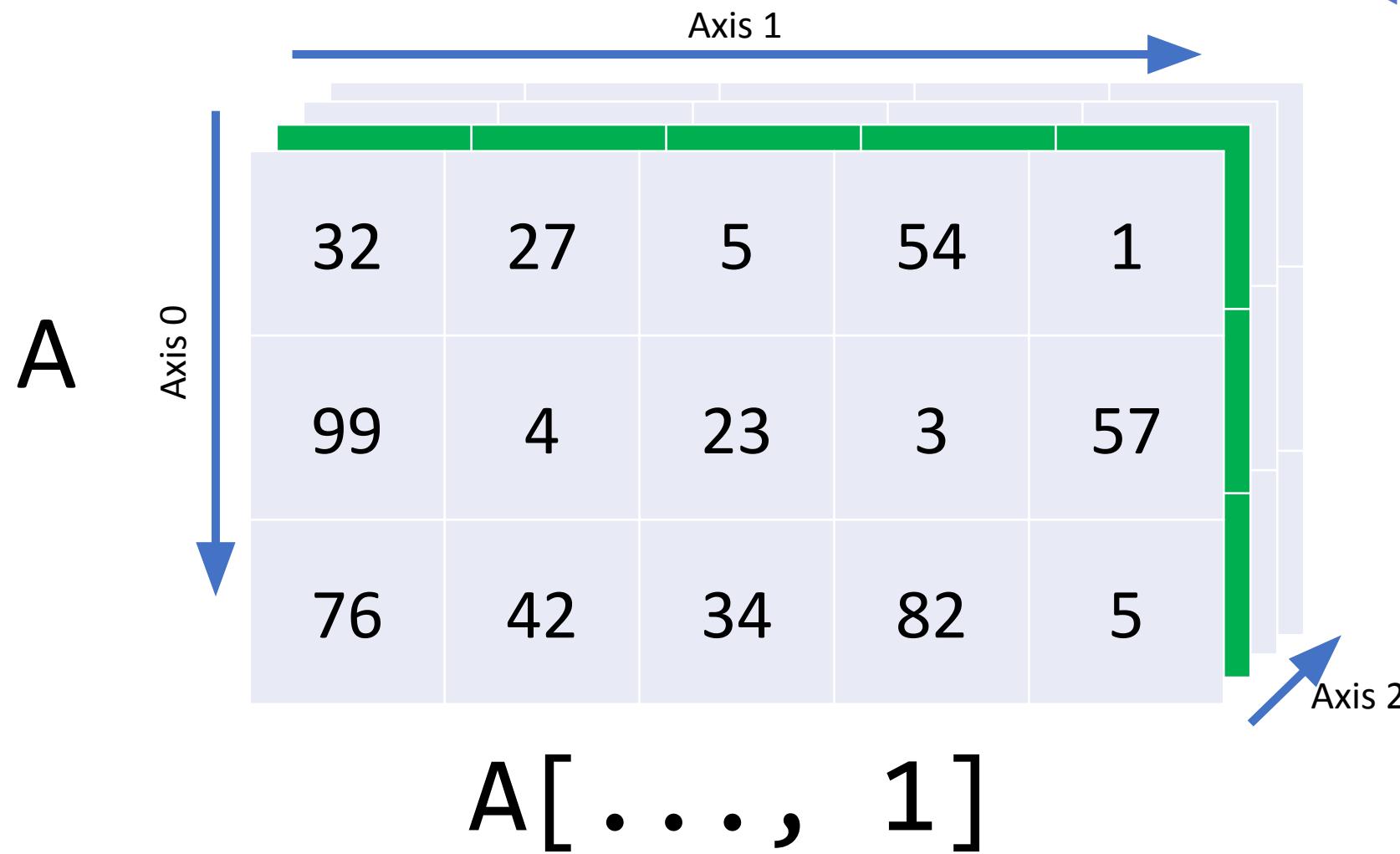
Multidimensional Indexing



$A[0, \dots]$



Multidimensional Indexing

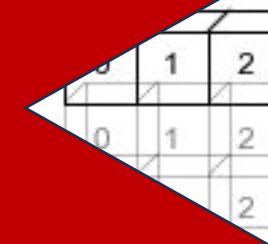


Broadcasting

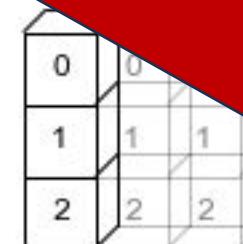
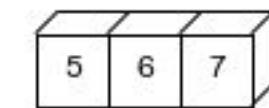
TL;DR: Shape
like $(6, 5, 4, 3, 1)$
to shape $(6, 5, 4, 3, 2)$

(Trailing dimensions
matched, array
repeated along
dimensions)

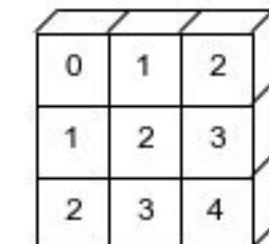
`np.arange(3)`



=



=



Shape Operations



```
A = np.random.normal(size=(10, 15))

# Indexing with newaxis/None
# adds an axis with size 1
A[np.newaxis] # -> shape (1, 10, 15)

# Squeeze removes a axis with size 1
A[np.newaxis].squeeze(0) # -> shape (10, 15)

# Transpose switches out axes.
A.transpose((1, 0)) # -> shape (15, 10)

# !!! BE CAREFUL WITH RESHAPE !!!
A.reshape(15, 10)    # -> shape (15, 10)
A.reshape(3, 25, -1) # -> shape (3, 25, 2)
```



```
A = torch.randn((10, 15))

# Indexing with None
# adds an axis with size 1
A[None] # -> shape (1, 10, 15)

# Squeeze removes a axis with size 1
A[None].squeeze(0) # -> shape (10, 15)

# Permute switches out axes.
A.permute((1, 0)) # -> shape (15, 10)

# !!! BE CAREFUL WITH VIEW !!!
A.view(15, 10)    # -> shape (15, 10)
A.view(3, 25, -1) # -> shape (3, 25, 2)
```

Device Management

- Numpy: all arrays live on the CPU's RAM
- Torch: tensors can either live on CPU or GPU memory
 - Move to GPU with `.to("cuda")/.cuda()`
 - Move to CPU with `.to("cpu")/.cpu()`

**YOU CANNOT PERFORM OPERATIONS BETWEEN
TENSORS ON DIFFERENT DEVICES!**

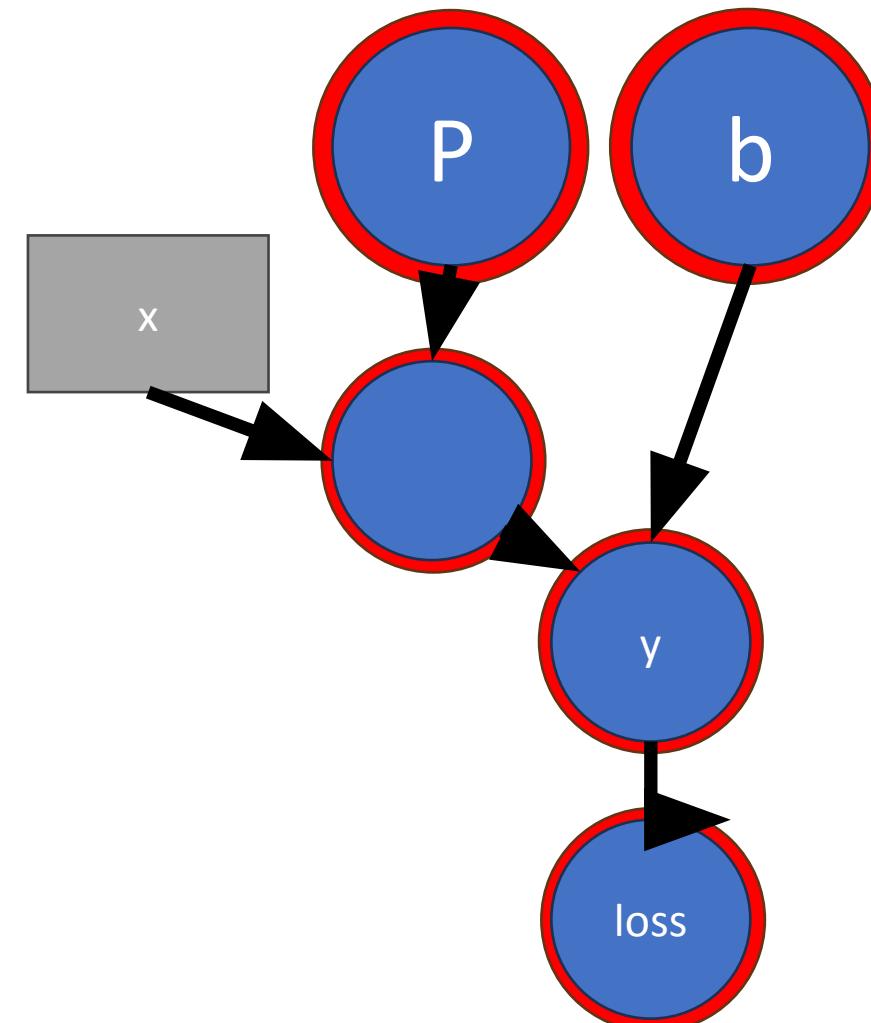
```
[ ] device = torch.device("cuda")
x = torch.zeros((2, 3))
y = torch.ones((2, 3), device=device)
z = x + y
```

```
-----
RuntimeError                                Traceback (most recent call last)
<ipython-input-71-565d7b7035e6> in <module>
      2 x = torch.zeros((2, 3))
      3 y = torch.ones((2, 3), device=device)
----> 4 z = x + y
```

```
RuntimeError: Expected all tensors to be on the same device, but found at least two
devices, cuda:0 and cpu!
```

Computing Gradients

```
P = torch.randn((1024, 1024))
print(P.requires_grad) # -> False
P = torch.randn((1024, 1024), requires_grad=True)
b = torch.randn((1024,), requires_grad=True)
print(P.grad) # -> None
```

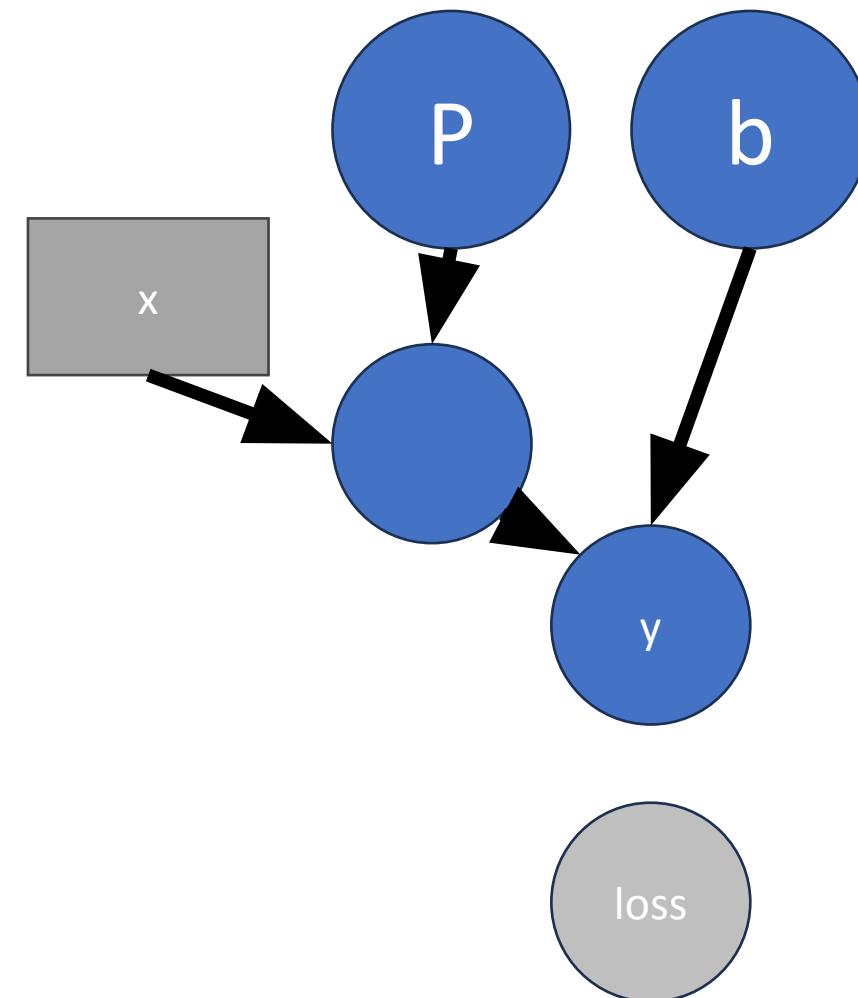


Computing Gradients

```
P = torch.randn((1024, 1024))
print(P.requires_grad) # -> False
P = torch.randn((1024, 1024), requires_grad=True)
b = torch.randn((1024,), requires_grad=True)
print(P.grad) # -> None

x = torch.randn((32, 1024))
y = torch.nn.relu(x @ P + b)

target = 3
loss = torch.mean((y - target) ** 2).detach()
```



Training Loop

REMEMBER THIS!

```
net = (...).to("cuda")
dataset = ...
dataloader = ..
optimizer = ...
loss_fn = ...
for epoch in range(num_epochs):
    # Training..
    net.train()
    for data, target in dataloader:
        data = torch.from_numpy(data).float().cuda()
        target = torch.from_numpy(target).float().cuda()

        prediction = net(data)
        loss = loss_fn(prediction, target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    net.eval()
    # Do evaluation..
```

Converting Numpy / PyTorch

Numpy -> PyTorch:

```
torch.from_numpy(numpy_array).float()
```

PyTorch -> Numpy:

- (If `requires_grad`) Get a copy without graph with `.detach()`
- (If on GPU) Move to CPU with `.to("cpu")/.cpu()`
- Convert to numpy with `.numpy`

All together:

```
torch_tensor.detach().cpu().numpy()
```

Custom networks

```
import torch.nn as nn

class SingleLayerNetwork(nn.Module):
    def __init__(self, in_dim: int, out_dim: int, hidden_dim: int):
        super().__init__() # <- Don't forget this!
        self.net = nn.Sequential(
            nn.Module(in_dim, hidden_dim),
            nn.ReLU(),
            nn.Module(hidden_dim, out_dim),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.net(x)

batch_size = 256
my_net = SingleLayerNetwork(2, 32, 1).to("cuda")
output = my_net(torch.randn(size=(batch_size, 2)).cuda())
```

- Prefer `net()` over `net.forward()`
- Everything (network and its inputs) on the same device!!!

Torch Best Practices

- When in doubt, **assert** is your friend

```
assert x.shape == (B, N), \
    f"Expected shape ({B, N}) but got {x.shape}"
```

- Be extra careful with `.reshape/.view`

- If you use it, assert before and after
- Only use it to collapse/expand a single dim
- In Torch, prefer `.flatten()/ .permute()/ .unflatten()`

- If you do some complicated operation, test it!

- Compare to a pure Python implementation

Torch Best Practices (continued)

- Don't mix numpy and Torch code
 - Understand the boundaries between the two
 - Make sure to cast 64-bit numpy arrays to 32 bits
 - `torch.Tensor` only in `nn.Module`!
- Training loop will always look the same
 - Load batch, compute loss
 - `.zero_grad()`, `.backward()`, `.step()`

PyTorch Tutorial (Colab)



<https://colab.research.google.com/drive/12nQiv6aZHNUcFAAuTjJenDWKQbIt2Mz>

<http://bit.ly/cs285-pytorch-2023>

Introduction to Reinforcement Learning

CS 285

Instructor: Sergey Levine
UC Berkeley

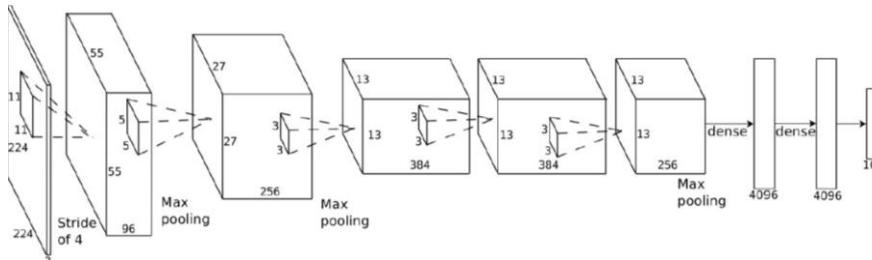


Definitions

Terminology & notation



\mathbf{o}_t



$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$



\mathbf{a}_t

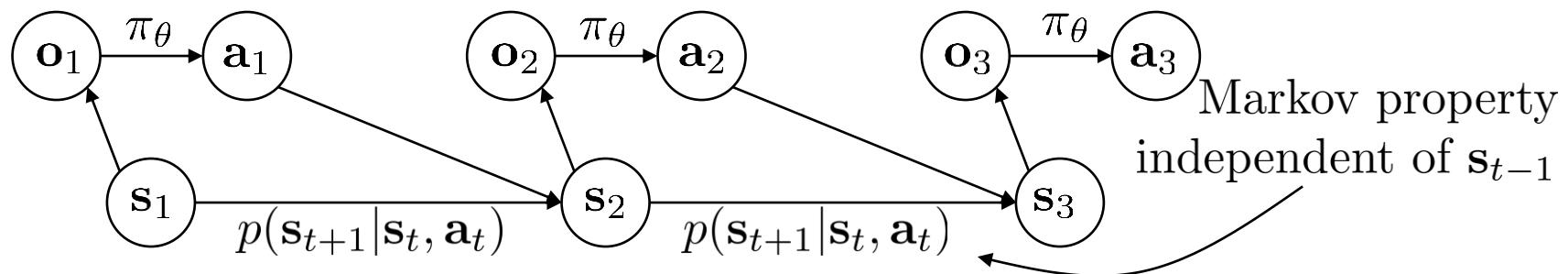
\mathbf{s}_t – state

\mathbf{o}_t – observation

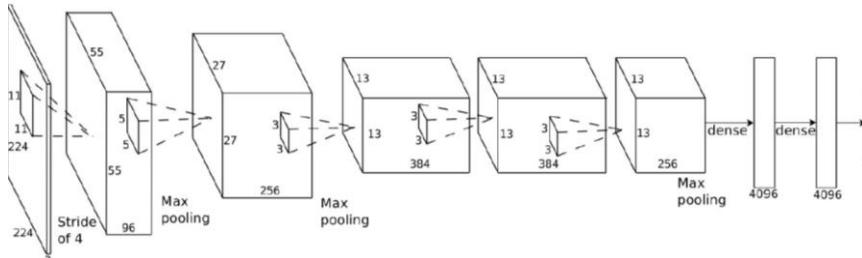
\mathbf{a}_t – action

$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$ – policy

$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ – policy (fully observed)



Imitation Learning



$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$$



$$\mathbf{o}_t$$



training
data



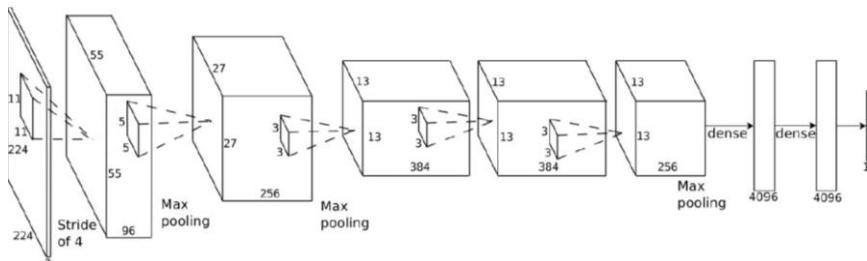
supervised
learning

$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$$

Reward functions



\mathbf{o}_t



$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$



\mathbf{a}_t

which action is better or worse?

$r(\mathbf{s}, \mathbf{a})$: reward function

tells us which states and actions are better



high reward



low reward

\mathbf{s} , \mathbf{a} , $r(\mathbf{s}, \mathbf{a})$, and $p(\mathbf{s}' | \mathbf{s}, \mathbf{a})$ define
Markov decision process

Definitions

Markov chain

$$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$$

\mathcal{S} – state space

states $s \in \mathcal{S}$ (discrete or continuous)

\mathcal{T} – transition operator

$$p(s_{t+1}|s_t)$$

Andrey Markov

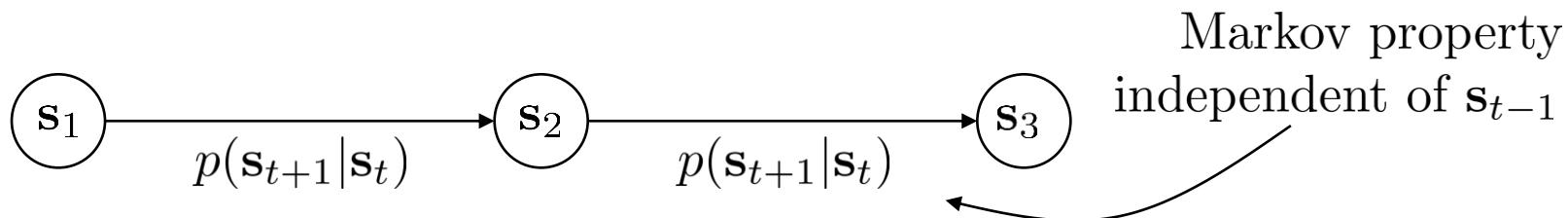
why “operator”?

$$\text{let } \mu_{t,i} = p(s_t = i)$$

$\vec{\mu}_t$ is a vector of probabilities

$$\text{let } \mathcal{T}_{i,j} = p(s_{t+1} = i | s_t = j)$$

$$\text{then } \vec{\mu}_{t+1} = \mathcal{T} \vec{\mu}_t$$



Definitions

Markov decision process

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$$

\mathcal{S} – state space

states $s \in \mathcal{S}$ (discrete or continuous)

\mathcal{A} – action space

actions $a \in \mathcal{A}$ (discrete or continuous)

\mathcal{T} – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

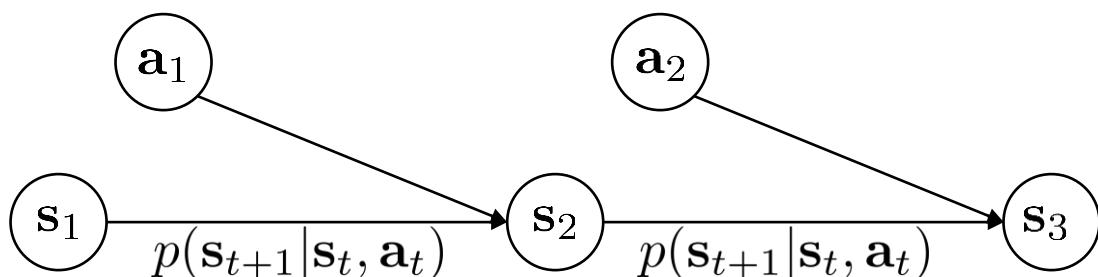
$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$

let $\xi_{t,k} = p(a_t = k)$

let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$



Richard Bellman



Definitions

Markov decision process

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$$

\mathcal{S} – state space

states $s \in \mathcal{S}$ (discrete or continuous)

\mathcal{A} – action space

actions $a \in \mathcal{A}$ (discrete or continuous)

\mathcal{T} – transition operator (now a tensor!)

r – reward function

$$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

$r(s_t, a_t)$ – reward



Richard Bellman

Definitions

partially observed Markov decision process

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$$

\mathcal{S} – state space

states $s \in \mathcal{S}$ (discrete or continuous)

\mathcal{A} – action space

actions $a \in \mathcal{A}$ (discrete or continuous)

\mathcal{O} – observation space

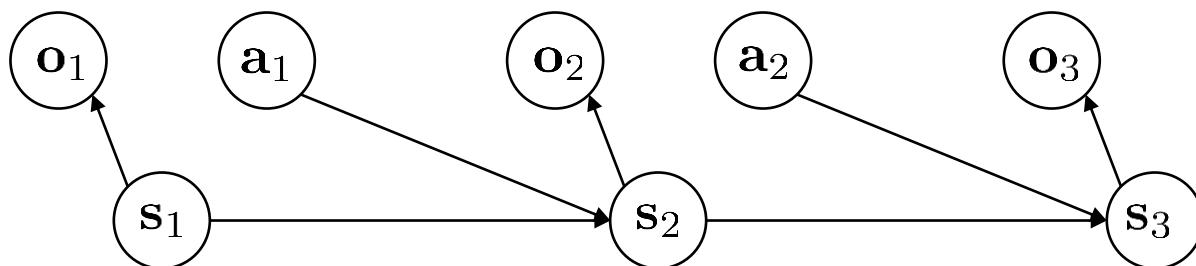
observations $o \in \mathcal{O}$ (discrete or continuous)

\mathcal{T} – transition operator (like before)

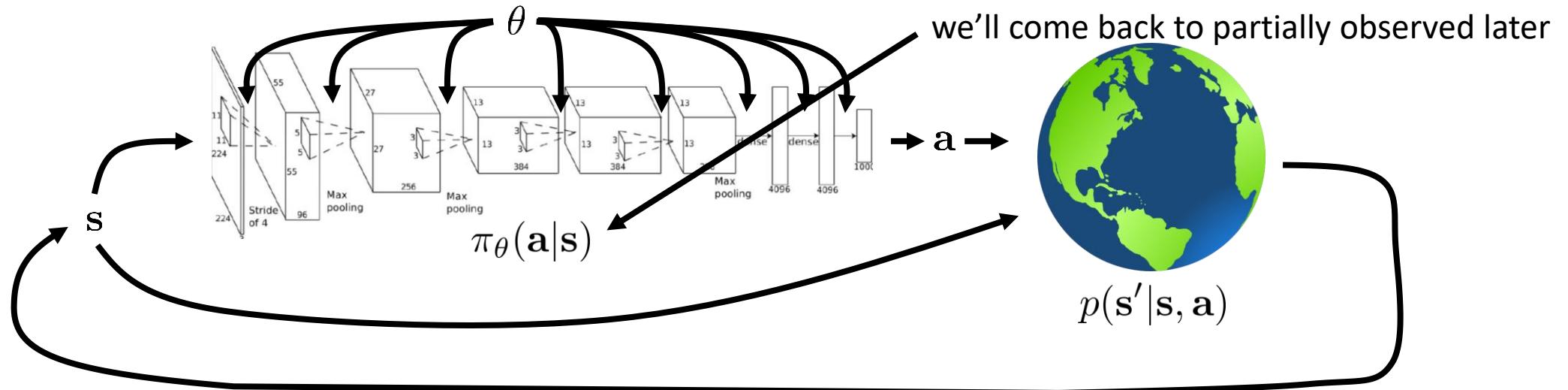
\mathcal{E} – emission probability $p(o_t|s_t)$

r – reward function

$$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$



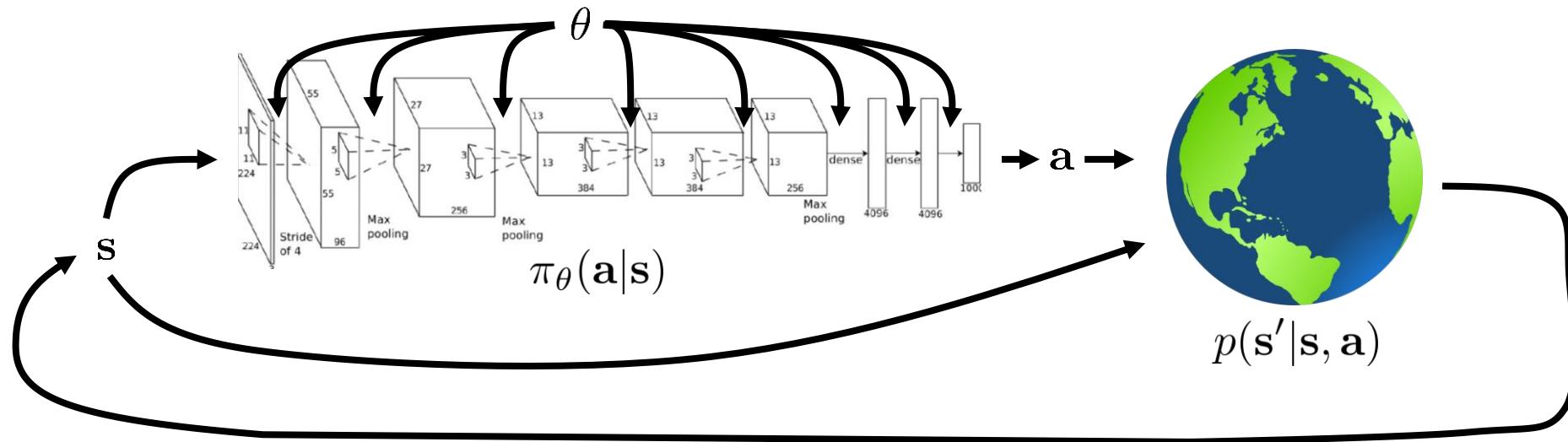
The goal of reinforcement learning



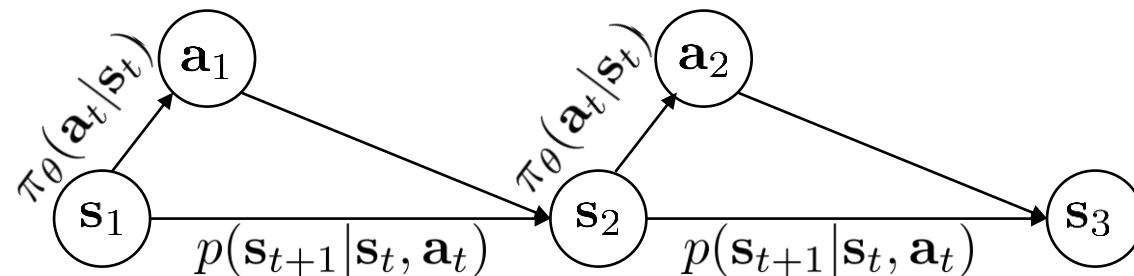
$$p_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \underbrace{\prod_{t=1}^T \pi_\theta(a_t | s_t)}_{p_\theta(\tau)} p(s_{t+1} | s_t, a_t)$$

$$\theta^\star = \arg \max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

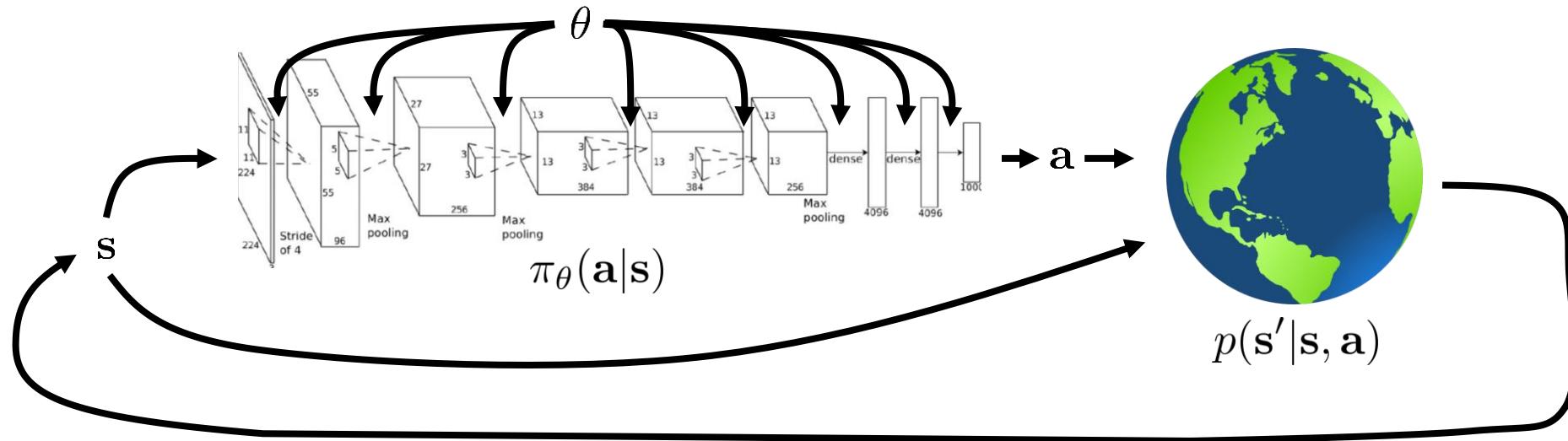
The goal of reinforcement learning



$$p_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \underbrace{\prod_{t=1}^T \pi_\theta(a_t | s_t)}_{p_\theta(\tau)} \underbrace{p(s_{t+1} | s_t, a_t)}_{\text{Markov chain on } (s, a)}$$

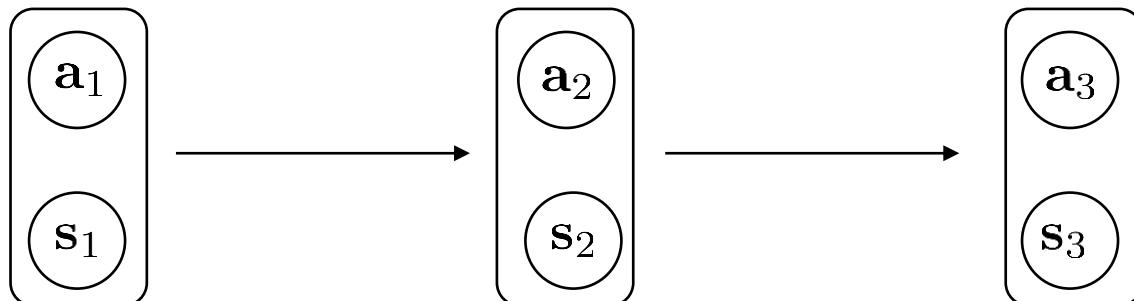


The goal of reinforcement learning



$$p_\theta(s_1, a_1, \dots, s_T, a_T) = \underbrace{p(s_1)}_{p_\theta(\tau)} \prod_{t=1}^T \underbrace{\pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)}_{\text{Markov chain on } (s, a)}$$

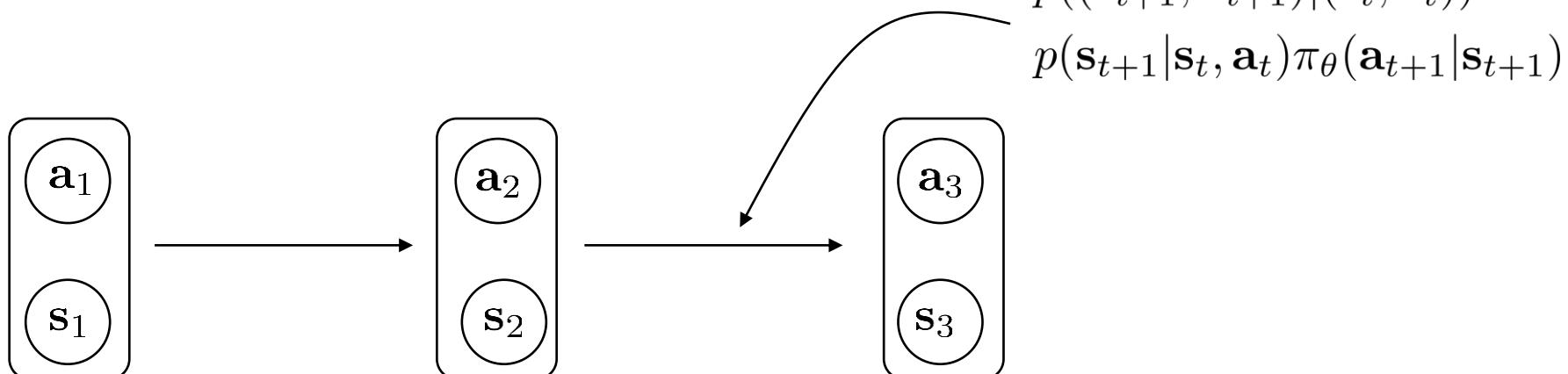
$$\begin{aligned} p((s_{t+1}, a_{t+1}) | (s_t, a_t)) &= \\ p(s_{t+1} | s_t, a_t) \pi_\theta(a_{t+1} | s_{t+1}) \end{aligned}$$



Finite horizon case: state-action marginal

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$= \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)] \quad p_{\theta}(\mathbf{s}_t, \mathbf{a}_t) \quad \text{state-action marginal}$$



Infinite horizon case: stationary distribution

$$\theta^* = \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

what if $T = \infty$?

does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution?

$$\mu = \mathcal{T}\mu$$

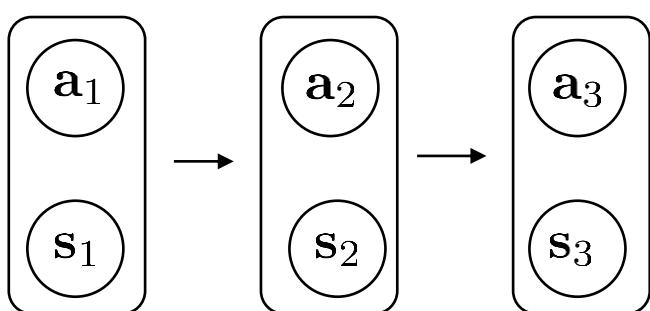
↑
stationary = the
same before and
after transition

$$(\mathcal{T} - \mathbf{I})\mu = 0$$

μ is eigenvector of \mathcal{T} with eigenvalue 1!

(always exists under some regularity conditions)

$$\mu = p_{\theta}(\mathbf{s}, \mathbf{a}) \quad \text{stationary distribution}$$



state-action transition operator

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} \quad \begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

Infinite horizon case: stationary distribution

$$\theta^* = \arg \max_{\theta} \frac{1}{T} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)] \rightarrow E_{(\mathbf{s}, \mathbf{a}) \sim p_\theta(\mathbf{s}, \mathbf{a})} [r(\mathbf{s}, \mathbf{a})]$$

(in the limit as $T \rightarrow \infty$)

what if $T = \infty$?

does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution?

$$\mu = \mathcal{T}\mu$$

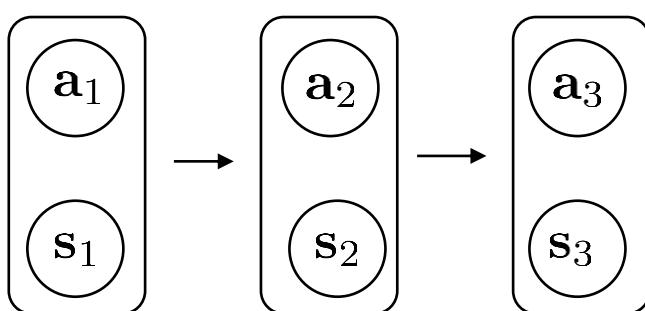
↑
stationary = the
same before and
after transition

$$(\mathcal{T} - \mathbf{I})\mu = 0$$

μ is eigenvector of \mathcal{T} with eigenvalue 1!

(always exists under some regularity conditions)

$$\mu = p_\theta(\mathbf{s}, \mathbf{a}) \quad \text{stationary distribution}$$



state-action transition operator

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} \quad \begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

Expectations and stochastic systems

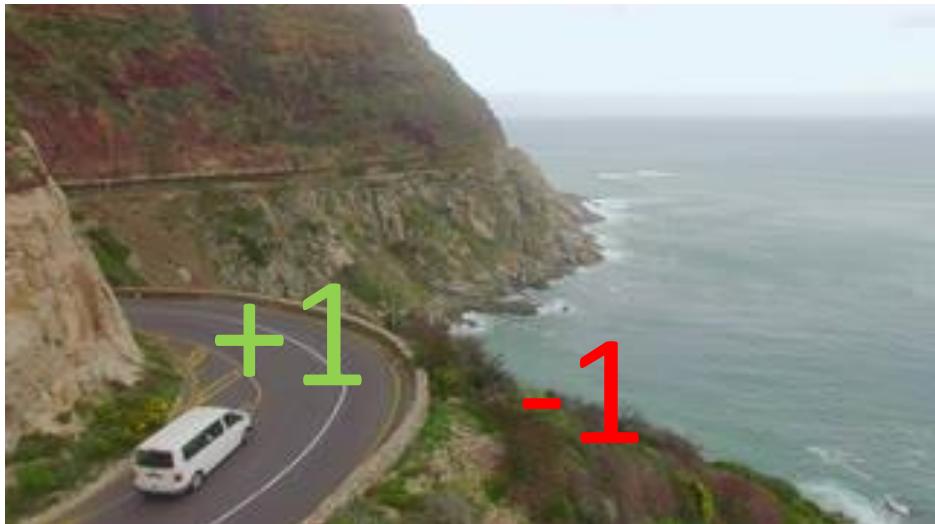
$$\theta^* = \arg \max_{\theta} E_{(\mathbf{s}, \mathbf{a}) \sim p_{\theta}(\mathbf{s}, \mathbf{a})}[r(\mathbf{s}, \mathbf{a})]$$

infinite horizon case

$$\theta^* = \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)]$$

finite horizon case

In RL, we almost always care about *expectations*



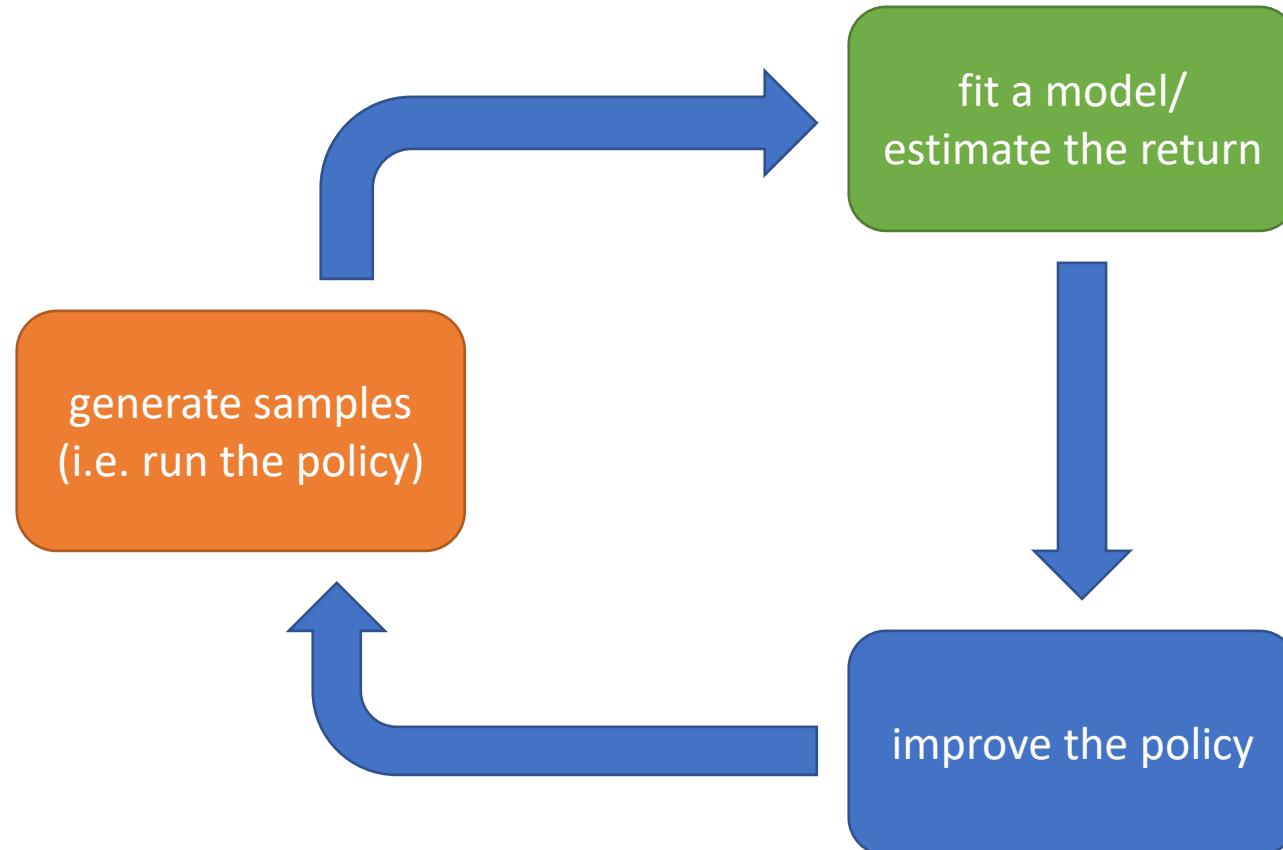
$r(\mathbf{x})$ – not smooth

$\pi_{\theta}(\mathbf{a} = \text{fall}) = \theta$

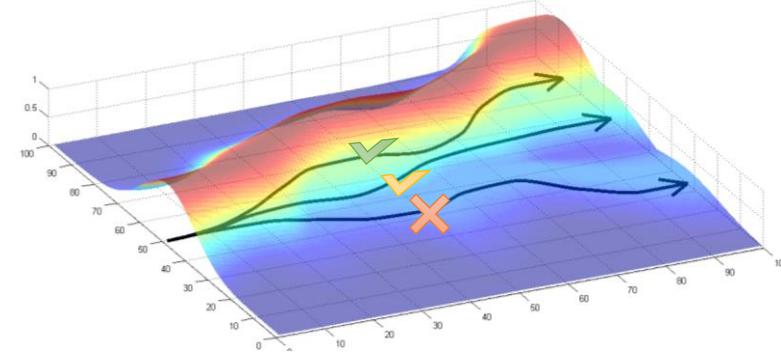
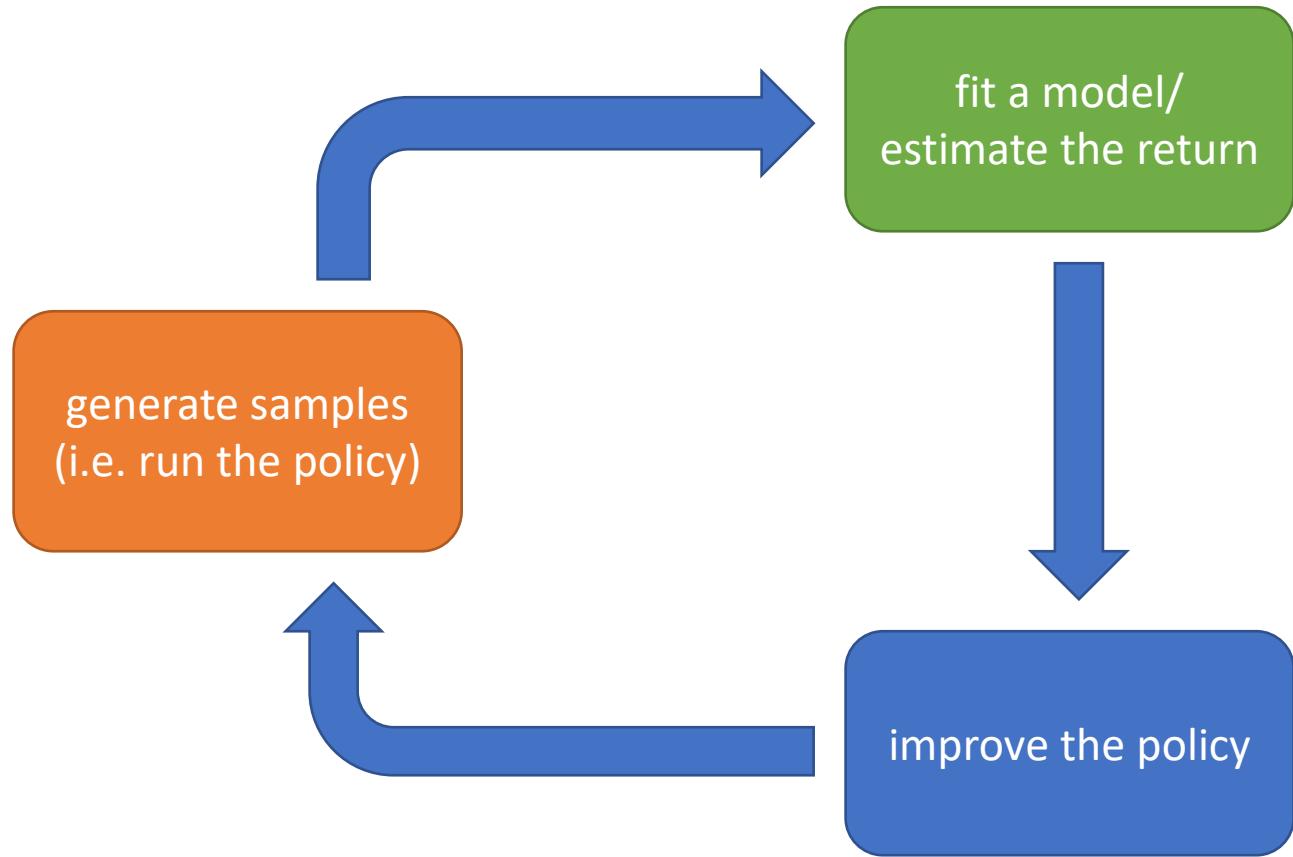
$E_{\pi_{\theta}}[r(\mathbf{x})]$ – smooth in θ !

Algorithms

The anatomy of a reinforcement learning algorithm



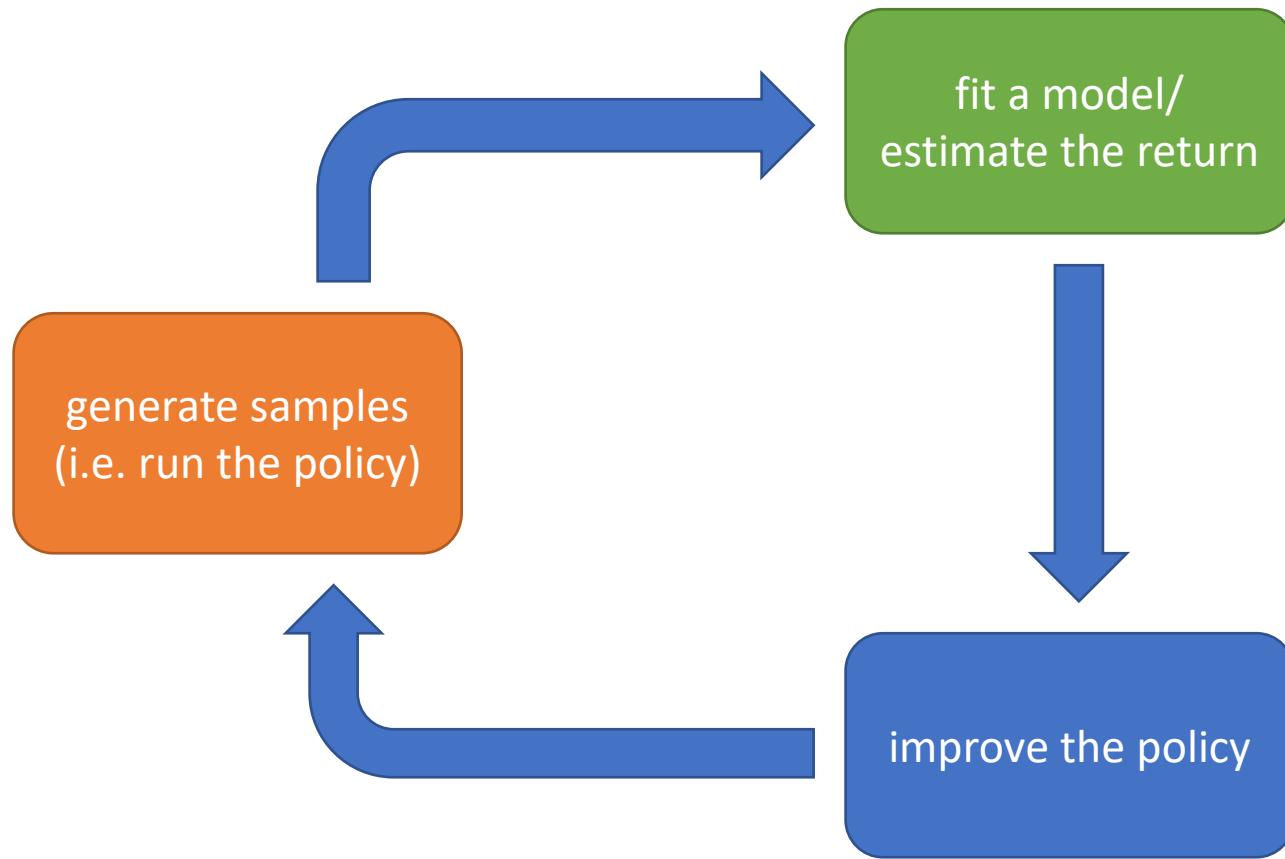
A simple example



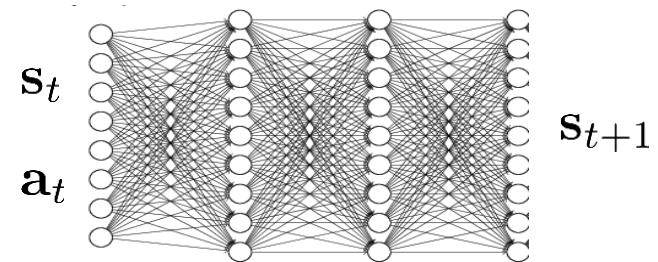
$$J(\theta) = E_{\pi} \left[\sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_t r_t^i$$

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

Another example: RL by backprop



learn f_ϕ such that $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$

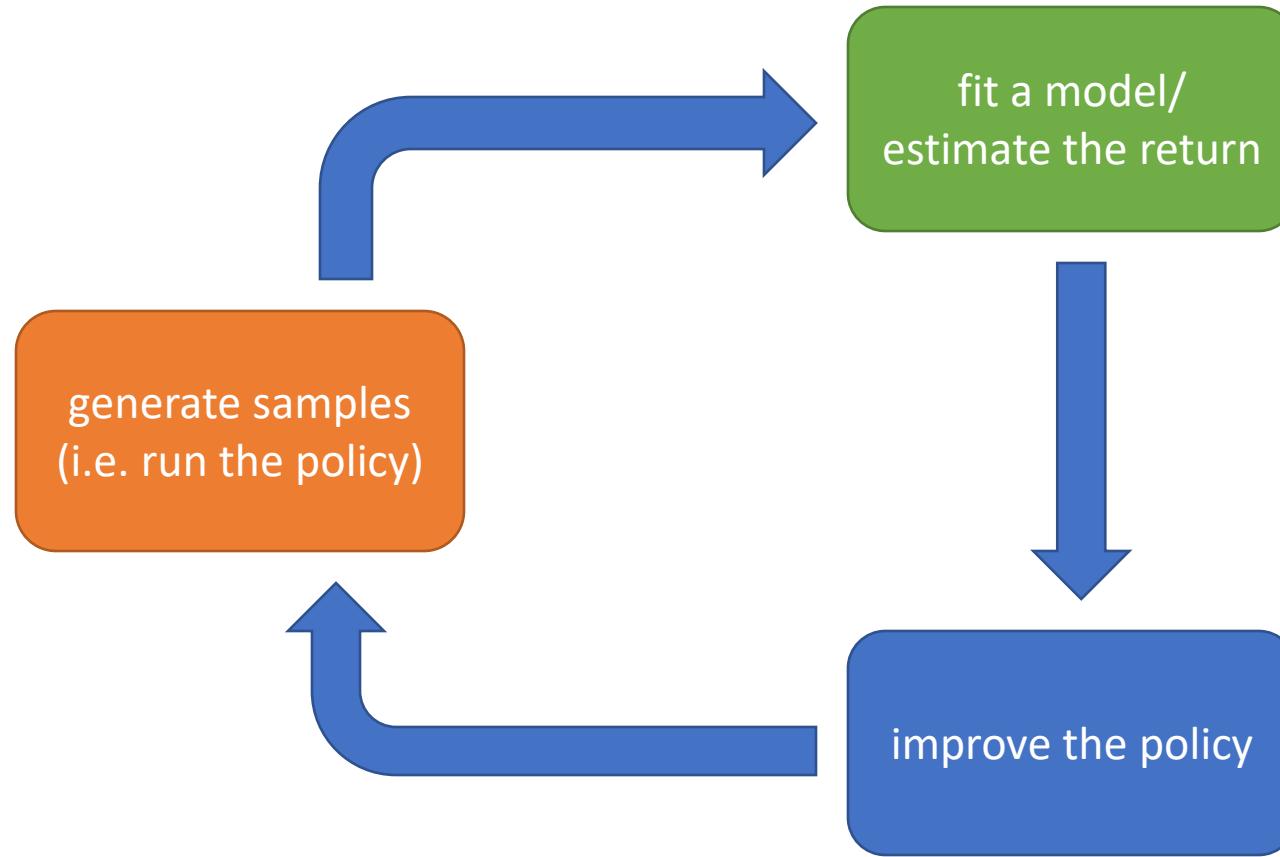


backprop through f_ϕ and r to
train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Which parts are expensive?

real robot/car/power grid/whatever:
1x real time, until we invent time travel

MuJoCo simulator:
up to 10000x real time



$$J(\theta) = E_\pi \left[\sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_t r_t^i$$

trivial, fast

$$\text{learn } \mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$$

expensive

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

backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Value Functions

How do we deal with all these expectations?

$$E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} \left[E_{\mathbf{a}_1 \sim \pi(\mathbf{a}_1 | \mathbf{s}_1)} [r(\mathbf{s}_1, \mathbf{a}_1) + E_{\mathbf{s}_2 \sim p(\mathbf{s}_2 | \mathbf{s}_1, \mathbf{a}_1)} \underbrace{\left[E_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2 | \mathbf{s}_2)} [r(\mathbf{s}_2, \mathbf{a}_2) + \dots | \mathbf{s}_2] | \mathbf{s}_1, \mathbf{a}_1 \right] | \mathbf{s}_1]} \right]$$

what if we knew this part?

$$Q(\mathbf{s}_1, \mathbf{a}_1) = r(\mathbf{s}_1, \mathbf{a}_1) + E_{\mathbf{s}_2 \sim p(\mathbf{s}_2 | \mathbf{s}_1, \mathbf{a}_1)} \left[E_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2 | \mathbf{s}_2)} [r(\mathbf{s}_2, \mathbf{a}_2) + \dots | \mathbf{s}_2] | \mathbf{s}_1, \mathbf{a}_1 \right]$$

$$E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right] = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} \left[E_{\mathbf{a}_1 \sim \pi(\mathbf{a}_1 | \mathbf{s}_1)} [Q(\mathbf{s}_1, \mathbf{a}_1) | \mathbf{s}_1] \right]$$

easy to modify $\pi_\theta(\mathbf{a}_1 | \mathbf{s}_1)$ if $Q(\mathbf{s}_1, \mathbf{a}_1)$ is known!

example: $\pi(\mathbf{a}_1 | \mathbf{s}_1) = 1$ if $\mathbf{a}_1 = \arg \max_{\mathbf{a}_1} Q(\mathbf{s}_1, \mathbf{a}_1)$

Definition: Q-function

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

Definition: value function

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

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

$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [V^\pi(\mathbf{s}_1)]$ is the RL objective!

Using Q-functions and value functions

Idea 1: if we have policy π , and we know $Q^\pi(\mathbf{s}, \mathbf{a})$, then we can *improve* π :

set $\pi'(\mathbf{a}|\mathbf{s}) = 1$ if $\mathbf{a} = \arg \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$

this policy is at least as good as π (and probably better)!

and it doesn't matter what π is

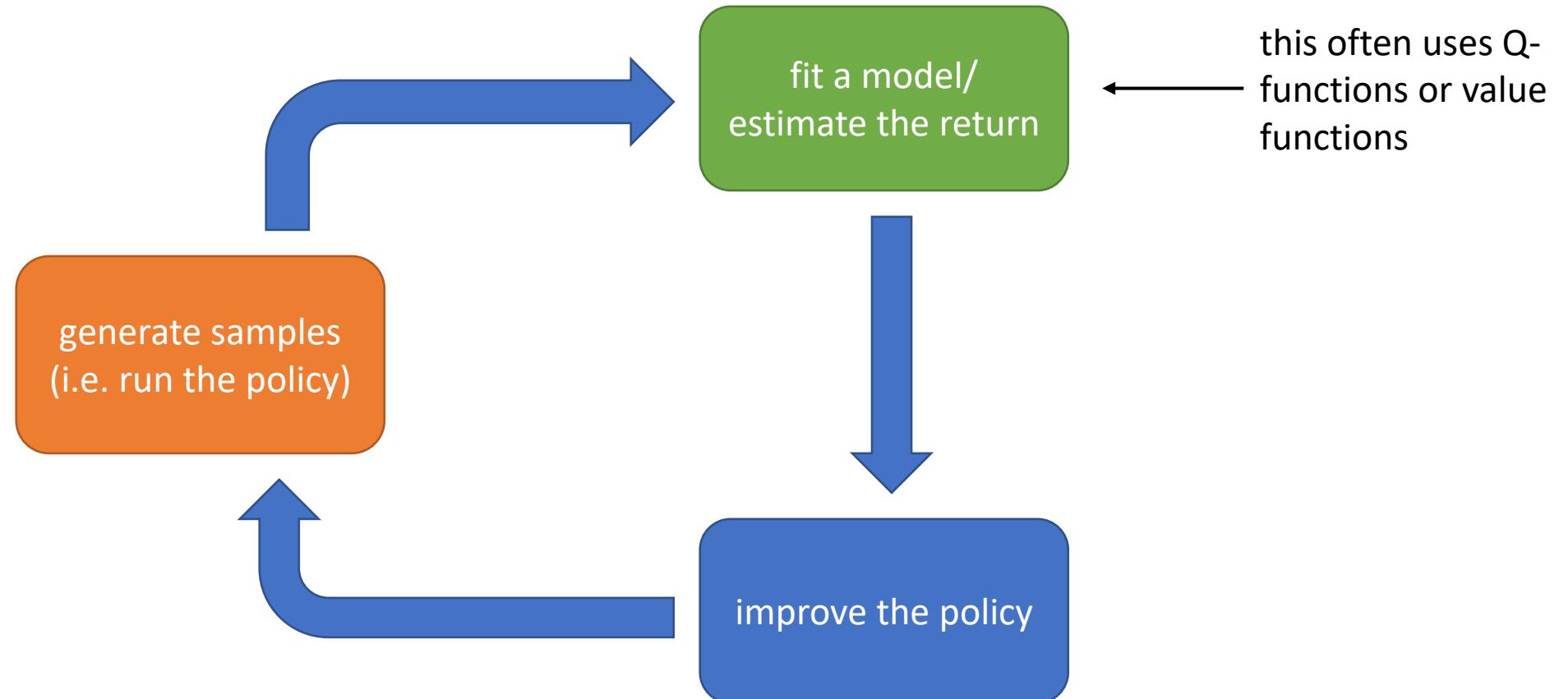
Idea 2: compute gradient to increase probability of good actions \mathbf{a} :

if $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$, then \mathbf{a} is *better than average* (recall that $V^\pi(\mathbf{s}) = E[Q^\pi(\mathbf{s}, \mathbf{a})]$ under $\pi(\mathbf{a}|\mathbf{s})$)

modify $\pi(\mathbf{a}|\mathbf{s})$ to increase probability of \mathbf{a} if $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$

These ideas are *very* important in RL; we'll revisit them again and again!

The anatomy of a reinforcement learning algorithm



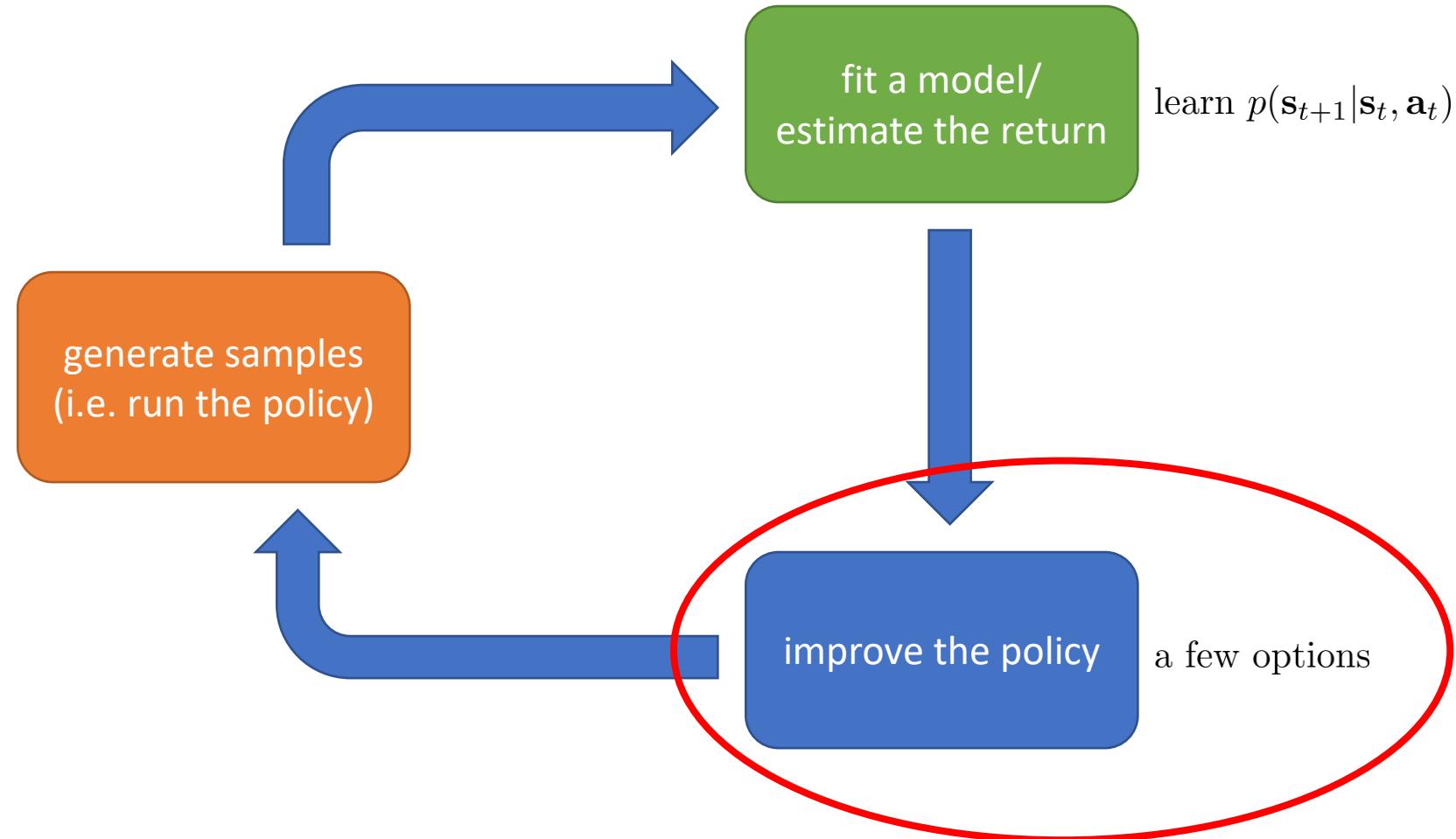
Types of Algorithms

Types of RL algorithms

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then...
 - Use it for planning (no explicit policy)
 - Use it to improve a policy
 - Something else

Model-based RL algorithms



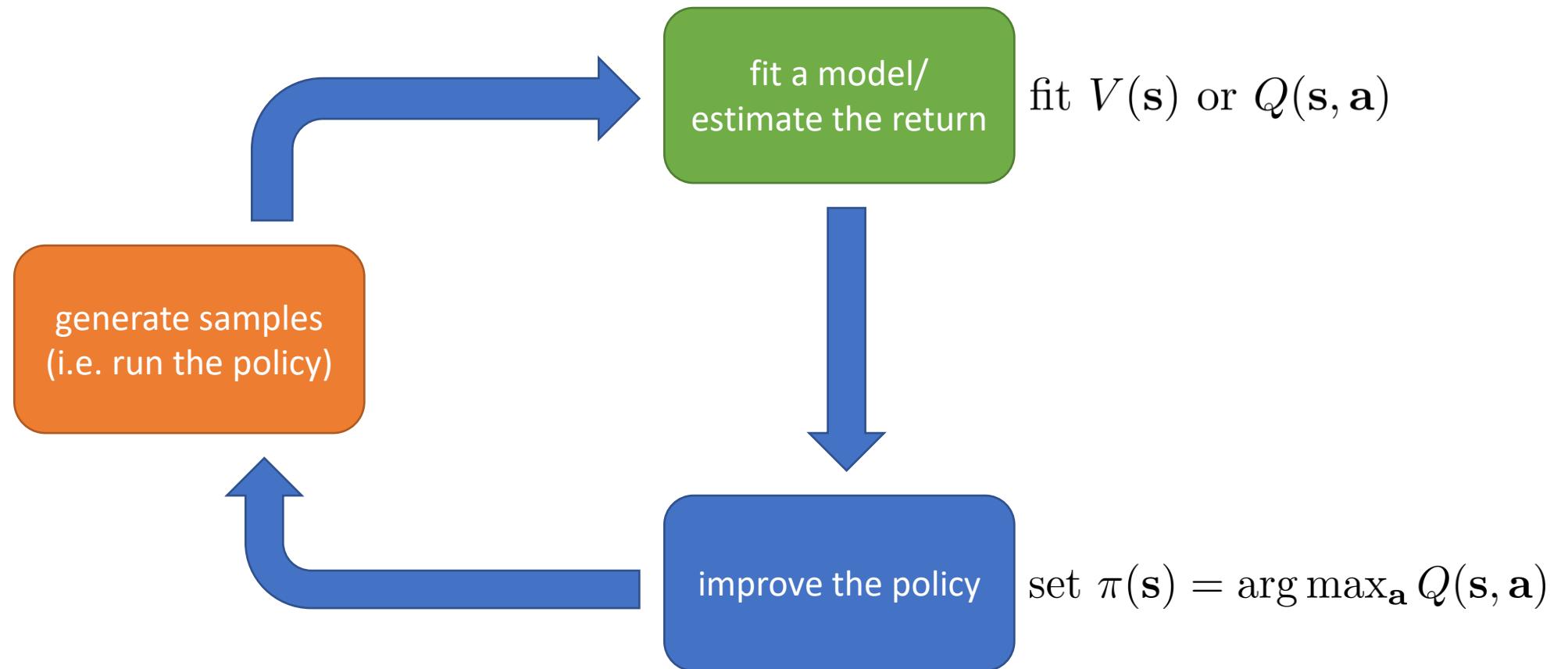
Model-based RL algorithms

improve the policy

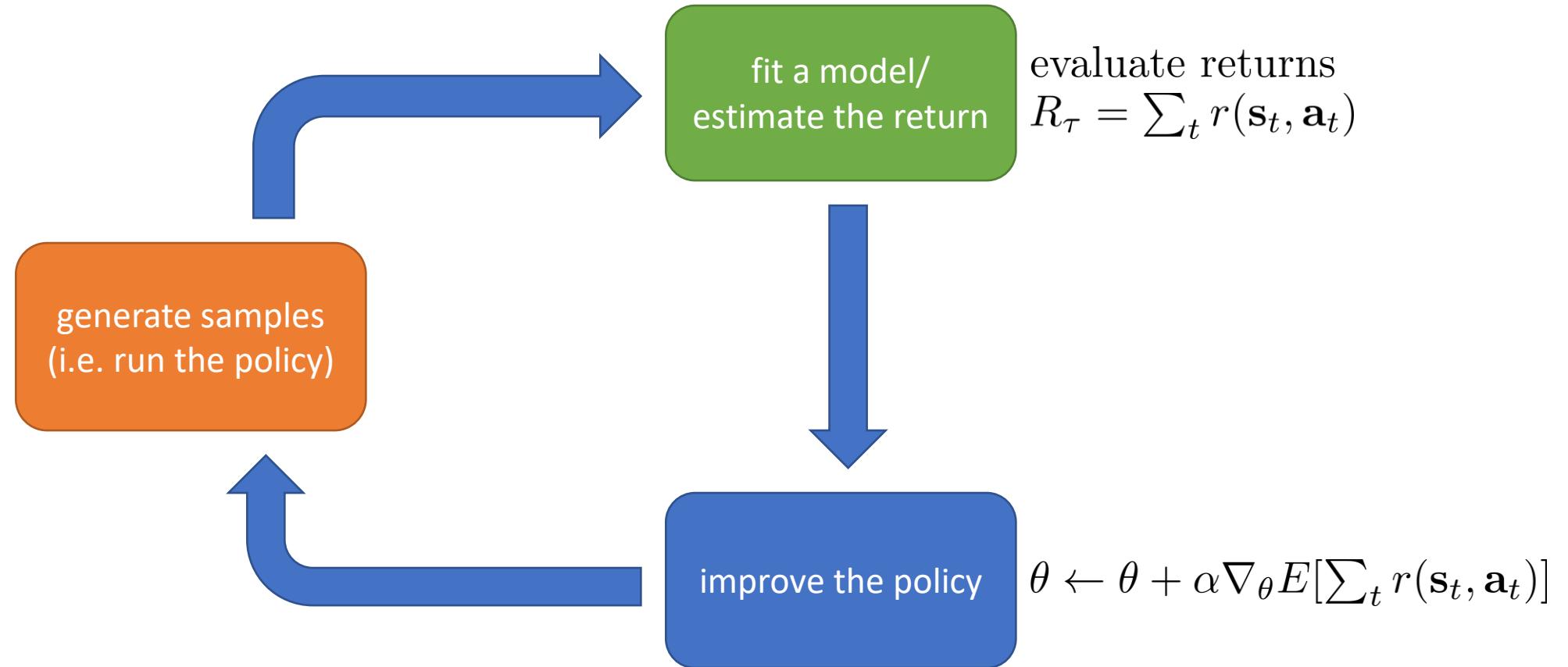
a few options

1. Just use the model to plan (no policy)
 - Trajectory optimization/optimal control (primarily in continuous spaces) – essentially backpropagation to optimize over actions
 - Discrete planning in discrete action spaces – e.g., Monte Carlo tree search
2. Backpropagate gradients into the policy
 - Requires some tricks to make it work
3. Use the model to learn a value function
 - Dynamic programming
 - Generate simulated experience for model-free learner

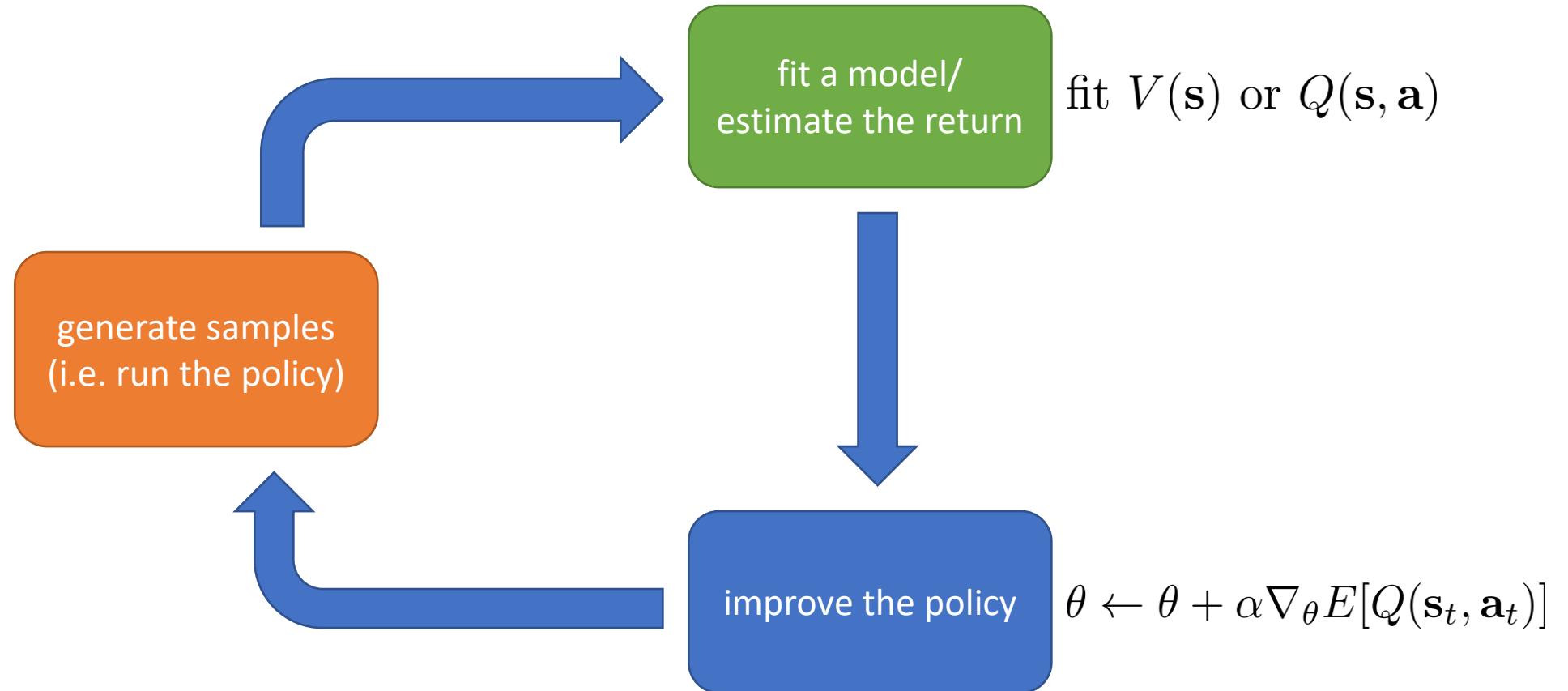
Value function based algorithms



Direct policy gradients



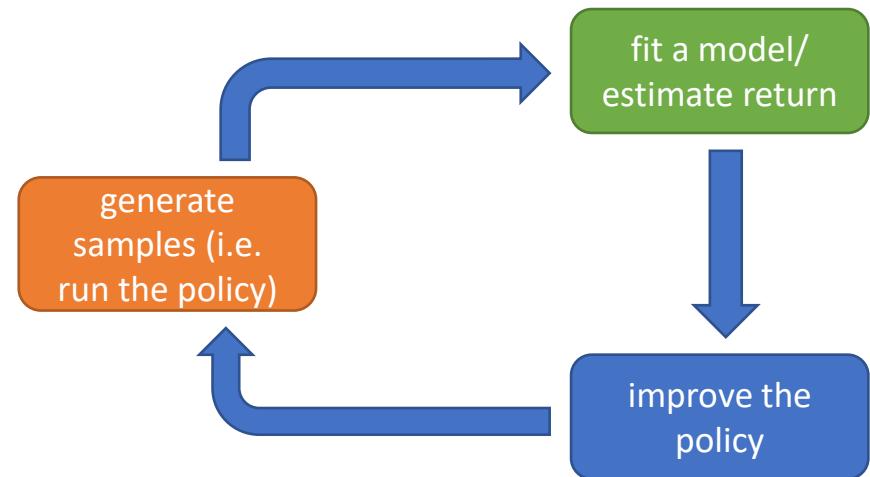
Actor-critic: value functions + policy gradients



Tradeoffs Between Algorithms

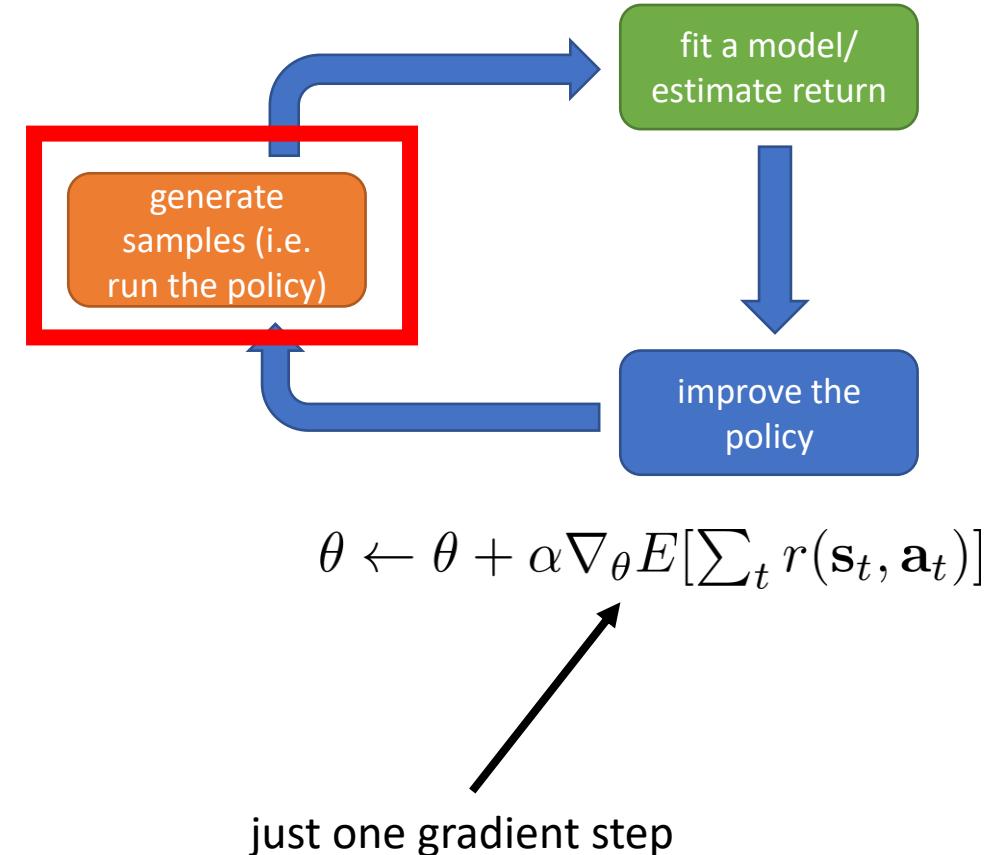
Why so many RL algorithms?

- Different tradeoffs
 - Sample efficiency
 - Stability & ease of use
- Different assumptions
 - Stochastic or deterministic?
 - Continuous or discrete?
 - Episodic or infinite horizon?
- Different things are easy or hard in different settings
 - Easier to represent the policy?
 - Easier to represent the model?

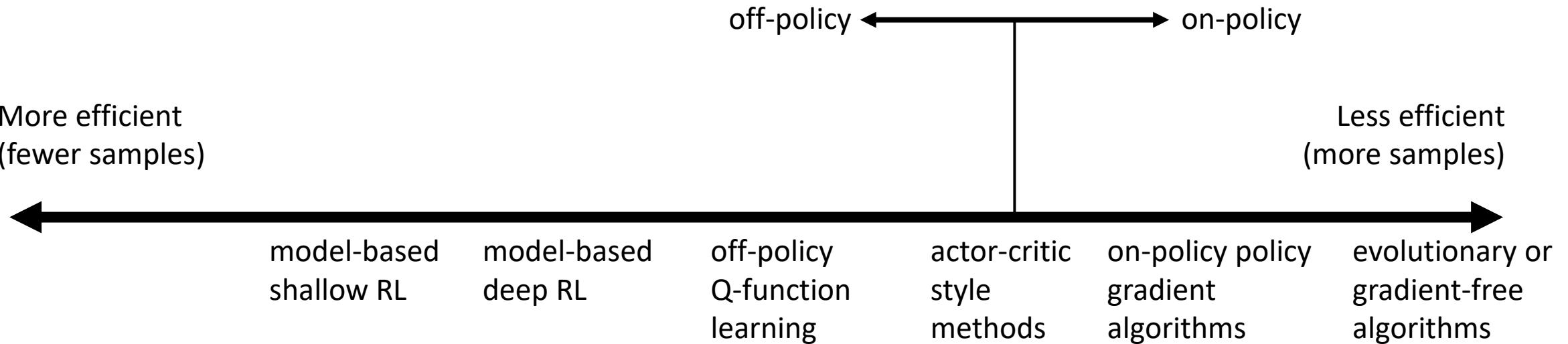


Comparison: sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?
- Most important question: is the algorithm *off policy*?
 - Off policy: able to improve the policy without generating new samples from that policy
 - On policy: each time the policy is changed, even a little bit, we need to generate new samples



Comparison: sample efficiency



Why would we use a *less* efficient algorithm?

Wall clock time is not the same as efficiency!

Comparison: stability and ease of use

- Does it converge?
- And if it converges, to what?
- And does it converge every time?

Why is any of this even a question???

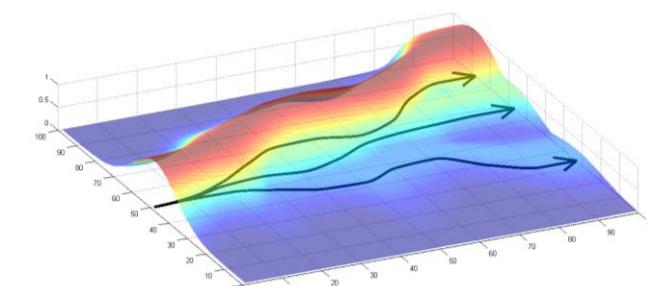
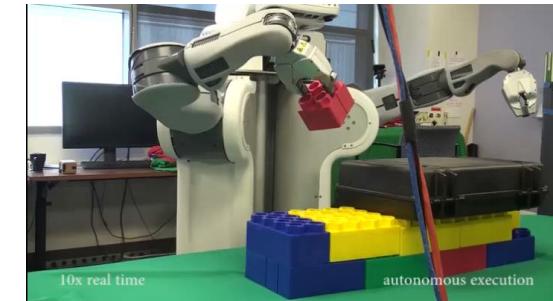
- Supervised learning: almost *always* gradient descent
- Reinforcement learning: often *not* gradient descent
 - Q-learning: fixed point iteration
 - Model-based RL: model is not optimized for expected reward
 - Policy gradient: *is* gradient descent, but also often the least efficient!

Comparison: stability and ease of use

- Value function fitting
 - At best, minimizes error of fit (“Bellman error”)
 - Not the same as expected reward
 - At worst, doesn’t optimize anything
 - Many popular deep RL value fitting algorithms are not guaranteed to converge to *anything* in the nonlinear case
- Model-based RL
 - Model minimizes error of fit
 - This will converge
 - No guarantee that better model = better policy
- Policy gradient
 - The only one that actually performs gradient descent (ascent) on the true objective

Comparison: assumptions

- Common assumption #1: full observability
 - Generally assumed by value function fitting methods
 - Can be mitigated by adding recurrence
- Common assumption #2: episodic learning
 - Often assumed by pure policy gradient methods
 - Assumed by some model-based RL methods
- Common assumption #3: continuity or smoothness
 - Assumed by some continuous value function learning methods
 - Often assumed by some model-based RL methods



Examples of Algorithms

Examples of specific algorithms

- Value function fitting methods
 - Q-learning, DQN
 - Temporal difference learning
 - Fitted value iteration
- Policy gradient methods
 - REINFORCE
 - Natural policy gradient
 - Trust region policy optimization
- Actor-critic algorithms
 - Asynchronous advantage actor-critic (A3C)
 - Soft actor-critic (SAC)
- Model-based RL algorithms
 - Dyna
 - Guided policy search

We'll learn about most of these in the next few weeks!

Example 1: Atari games with Q-functions

- Playing Atari with deep reinforcement learning, Mnih et al. '13
- Q-learning with convolutional neural networks



Example 2: robots and model-based RL

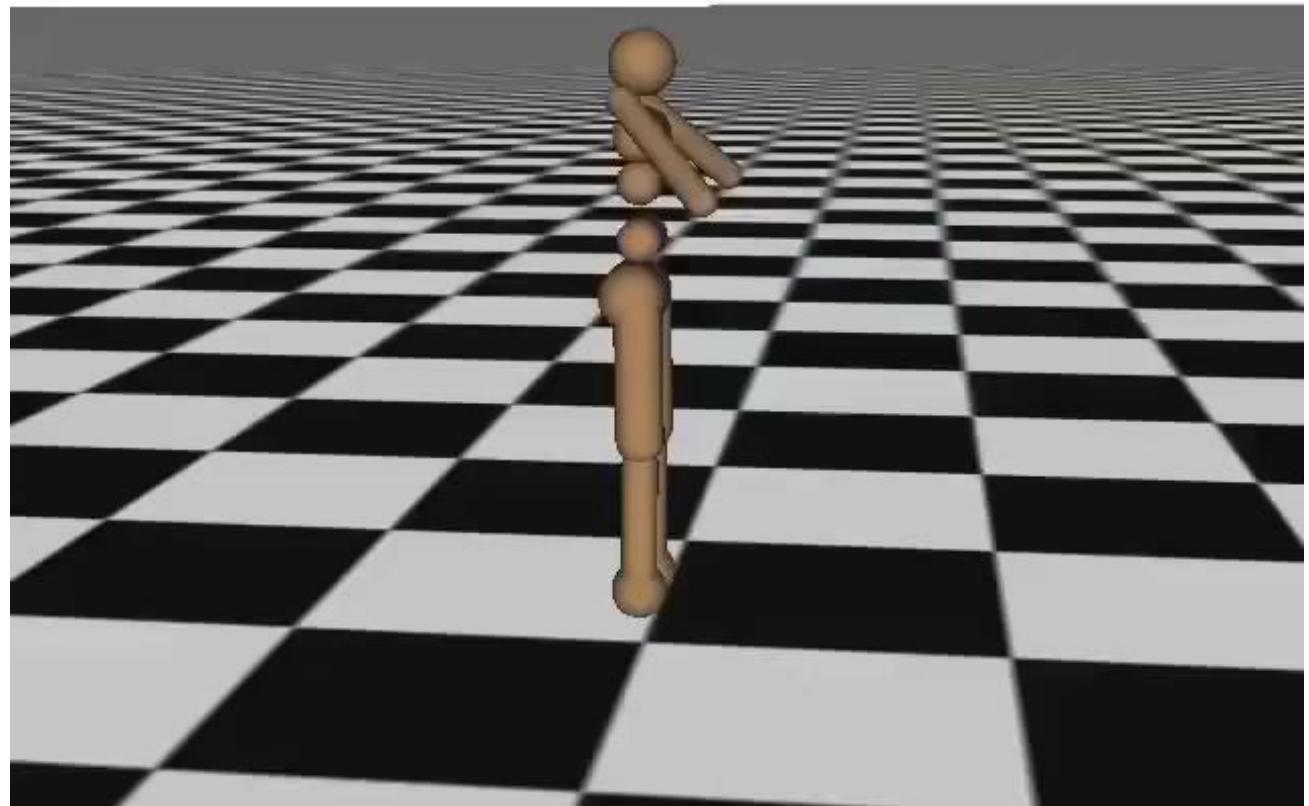
- End-to-end training of deep visuomotor policies, L.* , Finn* '16
- Guided policy search (model-based RL) for image-based robotic manipulation

Various Experiments
Including the policy input

Example 3: walking with policy gradients

- High-dimensional continuous control with generalized advantage estimation, Schulman et al. '16
- Trust region policy optimization with value function approximation

Iteration 0



Example 4: robotic grasping with Q-functions

- QT-Opt, Kalashnikov et al. '18
- Q-learning from images for real-world robotic grasping



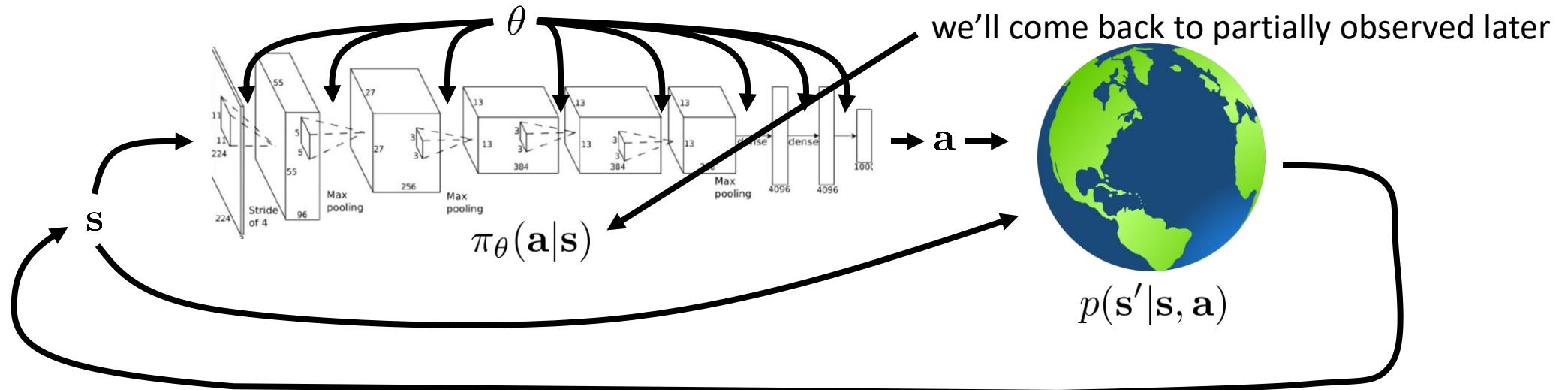
Policy Gradients

CS 285

Instructor: Sergey Levine
UC Berkeley



The goal of reinforcement learning



$$p_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \underbrace{\prod_{t=1}^T \pi_\theta(a_t | s_t)}_{p_\theta(\tau)} p(s_{t+1} | s_t, a_t)$$

$$\theta^\star = \arg \max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

The goal of reinforcement learning

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\theta^* = \arg \max_{\theta} E_{(\mathbf{s}, \mathbf{a}) \sim p_{\theta}(\mathbf{s}, \mathbf{a})} [r(\mathbf{s}, \mathbf{a})]$$

infinite horizon case

$$\theta^* = \arg \max_{\theta} \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

finite horizon case

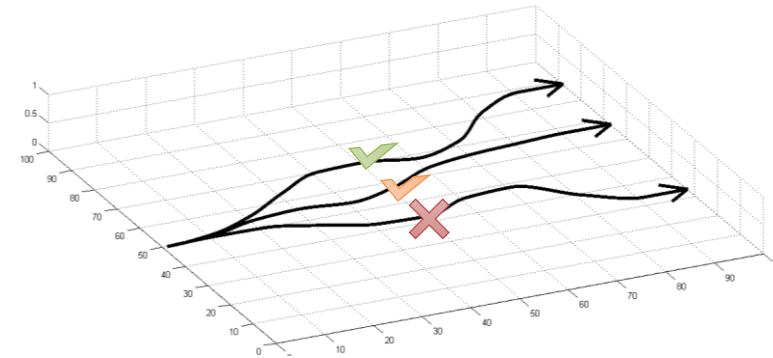
Evaluating the objective

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$J(\theta)$

$$J(\theta) = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

sum over samples from π_{θ}



Direct policy differentiation

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\underbrace{\sum_t r(\mathbf{s}_t, \mathbf{a}_t)}_{J(\theta)} \right]$$

a convenient identity

$$\underbrace{p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau)}_{\text{yellow}} = p_{\theta}(\tau) \frac{\nabla_{\theta} p_{\theta}(\tau)}{p_{\theta}(\tau)} = \underbrace{\nabla_{\theta} p_{\theta}(\tau)}_{\text{blue}}$$

$$J(\theta) = E_{\tau \sim p_{\theta}(\tau)} [r(\tau)] = \int p_{\theta}(\tau) r(\tau) d\tau$$
$$\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_{\theta} J(\theta) = \int \underbrace{\nabla_{\theta} p_{\theta}(\tau)}_{\text{blue}} r(\tau) d\tau = \int \underbrace{p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau)}_{\text{yellow}} r(\tau) d\tau = E_{\tau \sim p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

Direct policy differentiation

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

$$\nabla_{\theta} \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} \right]$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

log of both sides

$$p_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\log p_{\theta}(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

Evaluating the policy gradient

recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

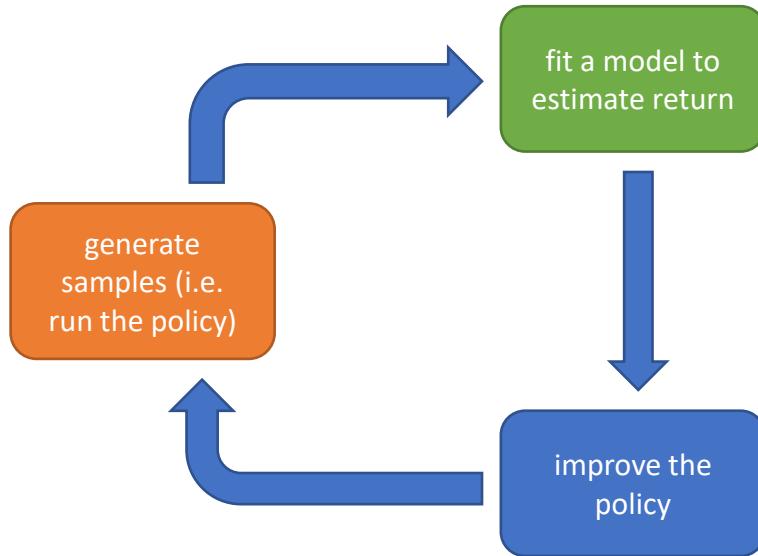
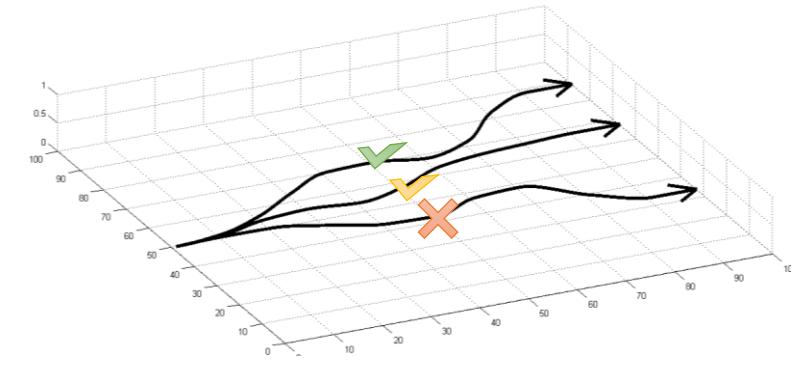
$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

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

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

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



Understanding Policy Gradients

Evaluating the policy gradient

recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

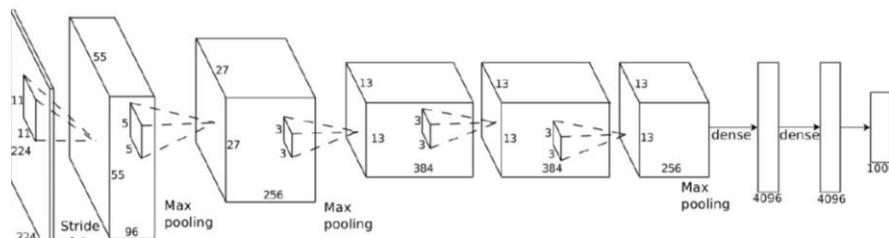
$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

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

what is this?



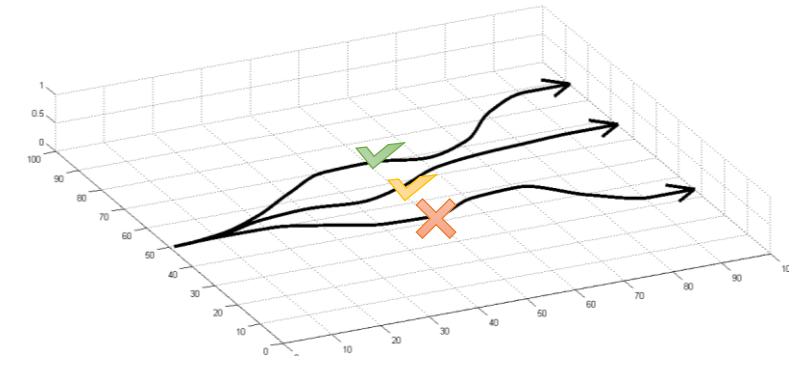
\mathbf{s}_t



$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$



\mathbf{a}_t



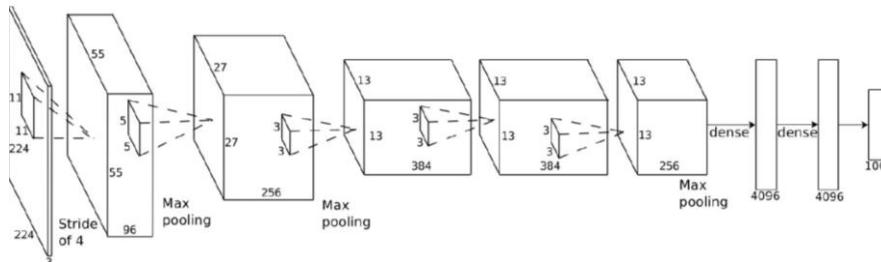
Comparison to maximum likelihood

policy gradient: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$

maximum likelihood: $\nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right)$



\mathbf{s}_t



$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$



\mathbf{a}_t



\mathbf{s}_t
 \mathbf{a}_t



supervised
learning

$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$

Example: Gaussian policies

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

example: $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

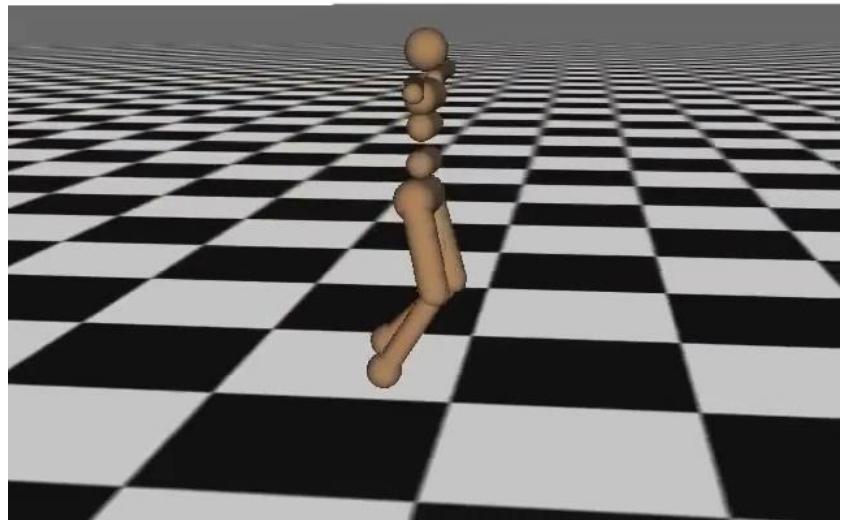
$$\log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \|f(\mathbf{s}_t) - \mathbf{a}_t\|_{\Sigma}^2 + \text{const}$$

$$\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \Sigma^{-1} (f(\mathbf{s}_t) - \mathbf{a}_t) \frac{df}{d\theta}$$

REINFORCE algorithm:

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

Iteration 2000



What did we just do?

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

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

maximum likelihood: $\nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau_i)$

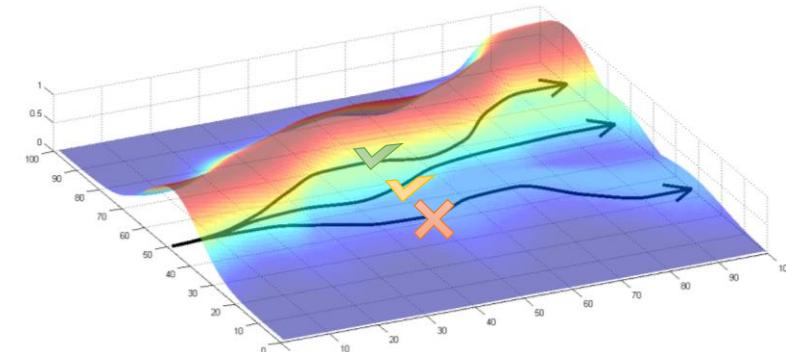
good stuff is made more likely

bad stuff is made less likely

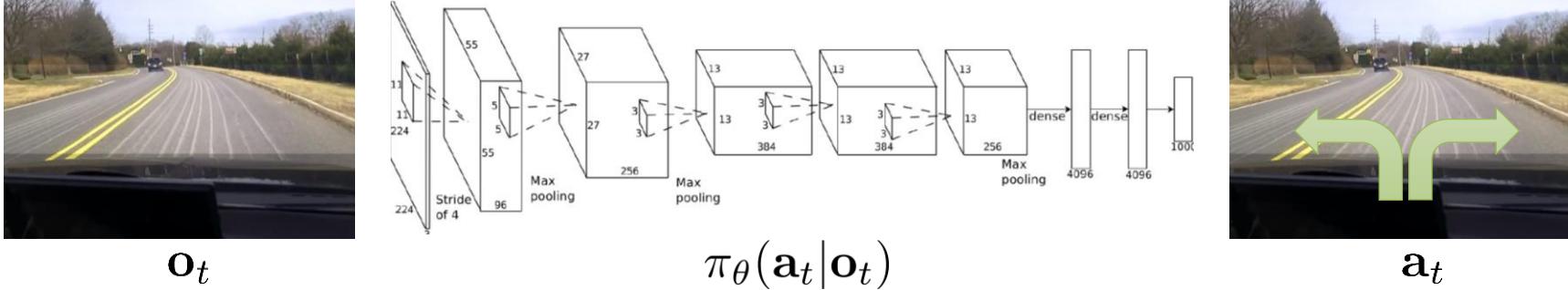
simply formalizes the notion of “trial and error”!

REINFORCE algorithm:

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



Partial observability



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

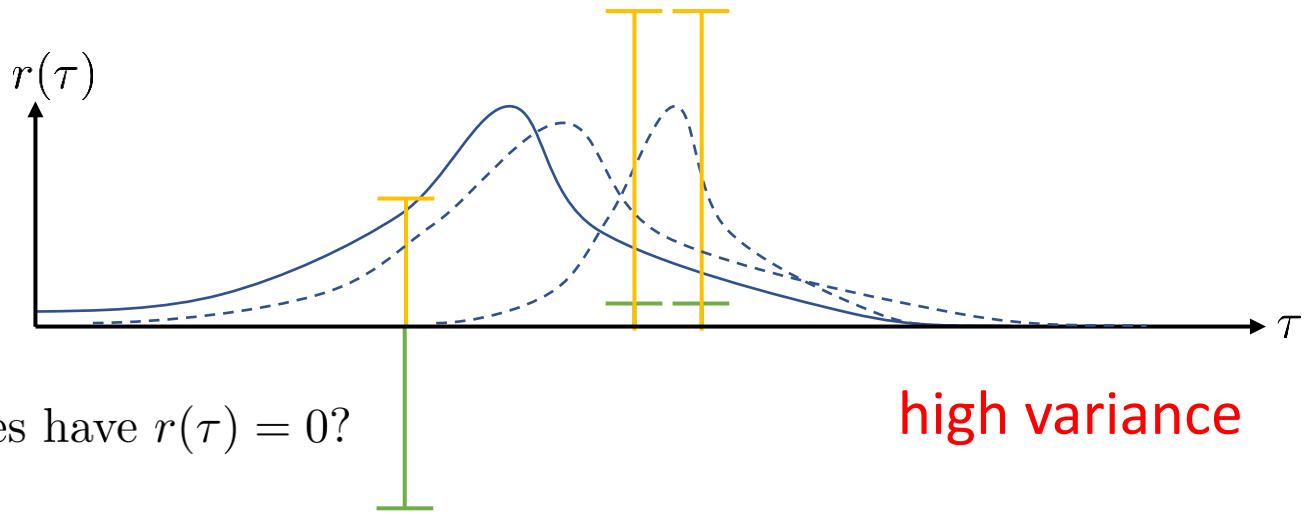
Markov property is not actually used!

Can use policy gradient in partially observed MDPs without modification

What is wrong with the policy gradient?

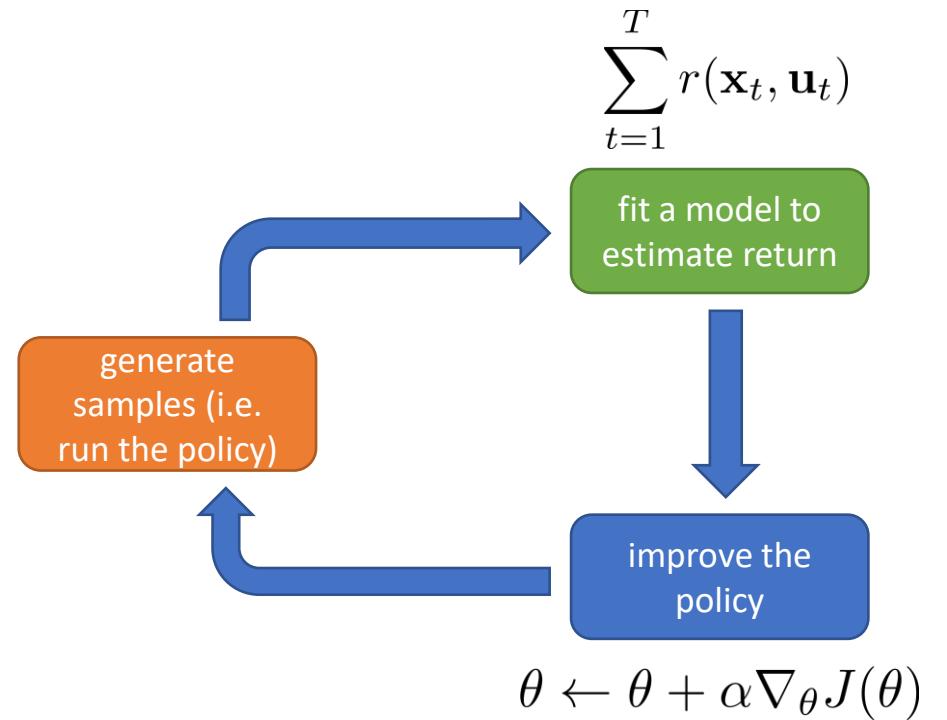
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)$$

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



Review

- Evaluating the RL objective
 - Generate samples
- Evaluating the policy gradient
 - Log-gradient trick
 - Generate samples
- Understanding the policy gradient
 - Formalization of trial-and-error
- Partial observability
 - Works just fine
- What is wrong with policy gradient?



Reducing Variance

Reducing variance

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

Causality: policy at time t' cannot affect reward at time t when $t < t'$

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

“reward to go”

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

Baselines

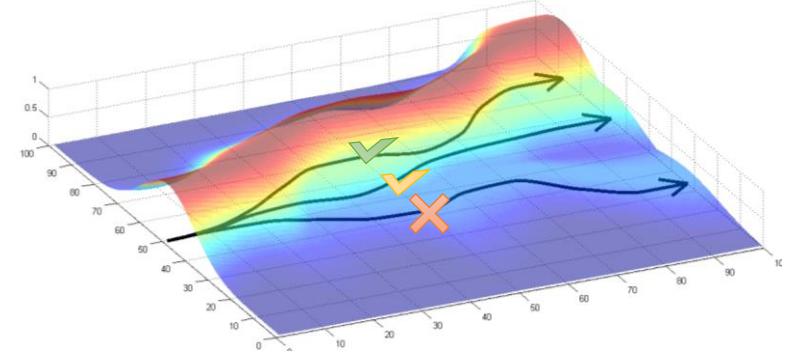
a convenient identity

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

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

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

but... are we *allowed* to do that??



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

subtracting a baseline is *unbiased* in expectation!

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

Analyzing variance

can we write down the variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) (r(\tau) - b)]$$

$$\text{Var} = E_{\tau \sim p_\theta(\tau)} [(\nabla_\theta \log p_\theta(\tau) (r(\tau) - b))^2] - E_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) (r(\tau) - b)]^2$$

this bit is just $E_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) r(\tau)]$
(baselines are unbiased in expectation)

$$\begin{aligned} \frac{d\text{Var}}{db} &= \frac{d}{db} E[g(\tau)^2 (r(\tau) - b)^2] = \frac{d}{db} (E[g(\tau)^2 r(\tau)^2] - 2E[g(\tau)^2 r(\tau)b] + b^2 E[g(\tau)^2]) \\ &= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0 \end{aligned}$$

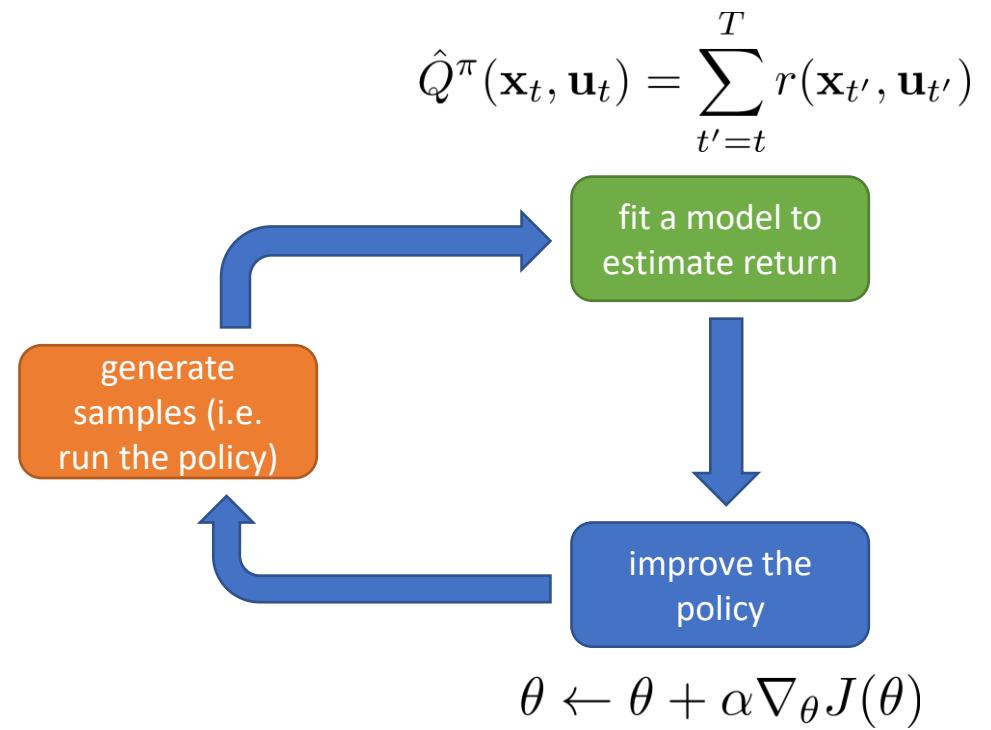
$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$



This is just expected reward, but weighted
by gradient magnitudes!

Review

- The high variance of policy gradient
- Exploiting causality
 - Future doesn't affect the past
- Baselines
 - Unbiased!
- Analyzing variance
 - Can derive optimal baselines



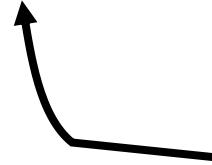
Off-Policy Policy Gradients

Policy gradient is on-policy

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

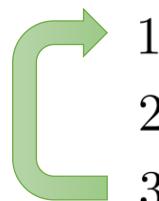
$$\nabla_{\theta} J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$



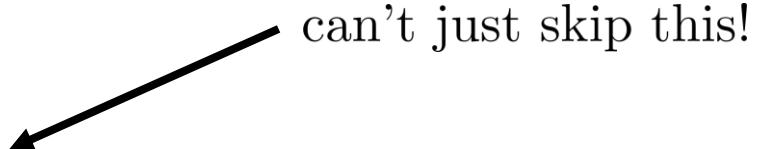
this is trouble...

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

REINFORCE algorithm:



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



can't just skip this!

Off-policy learning & importance sampling

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

what if we don't have samples from $p_\theta(\tau)$?

(we have samples from some $\bar{p}(\tau)$ instead)

$$J(\theta) = E_{\tau \sim \bar{p}(\tau)} \left[\frac{p_\theta(\tau)}{\bar{p}(\tau)} r(\tau) \right]$$

$$p_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{p(\mathbf{s}_1) \prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}{p(\mathbf{s}_1) \prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} = \frac{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t)}$$

importance sampling

$$E_{x \sim p(x)}[f(x)] = \int p(x)f(x)dx$$

$$= \int \frac{q(x)}{q(x)} p(x)f(x)dx$$

$$= \int q(x) \frac{p(x)}{q(x)} f(x)dx$$

$$= E_{x \sim q(x)} \left[\frac{p(x)}{q(x)} f(x) \right]$$

Deriving the policy gradient with IS

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

a convenient identity

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

can we estimate the value of some *new* parameters θ' ?

$$J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} r(\tau) \right]$$

the only bit that depends on θ'

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\frac{\nabla_{\theta'} p_{\theta'}(\tau)}{p_\theta(\tau)} r(\tau) \right] = E_{\tau \sim p_\theta(\tau)} \left[\frac{\cancel{p_{\theta'}(\tau)}}{\cancel{p_\theta(\tau)}} \nabla_{\theta'} \log p_{\theta'}(\tau) r(\tau) \right]$$

now estimate locally, at $\theta = \theta'$: $\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) r(\tau)]$

The off-policy policy gradient

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

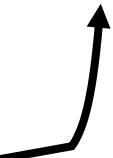
$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau) \right] \quad \text{when } \theta \neq \theta'$$

$$\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}$$

$$= E_{\tau \sim p_\theta(\tau)} \left[\left(\prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \right) \left(\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right] \text{what about causality?}$$

$$= E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left(\underbrace{\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'} | \mathbf{s}_{t'})}}_{\text{future actions don't affect current weight}} \right) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \left(\prod_{t''=t}^{t'} \frac{\pi_{\theta'}(\mathbf{a}_{t''} | \mathbf{s}_{t''})}{\pi_\theta(\mathbf{a}_{t''} | \mathbf{s}_{t''})} \right) \right) \right]$$

future actions don't affect current weight



if we ignore this, we get
a policy iteration algorithm
(more on this in a later lecture)

A first-order approximation for IS (preview)

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_{\theta}(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

let's write the objective a bit differently...

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

off-policy policy gradient: $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

We'll see why this is reasonable later in the course!

$$= \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \cancel{\frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t})}} \frac{\pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

ignore this part

exponential in T ...

$(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \sim \pi_{\theta}(\mathbf{s}_t, \mathbf{a}_t)$

Implementing Policy Gradients

Policy gradient with automatic differentiation

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

pretty inefficient to compute these explicitly!

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

maximum likelihood: $\nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})$ $J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})$

Just implement “pseudo-loss” as a weighted maximum likelihood:

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

cross entropy (discrete) or squared error (Gaussian)

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Maximum likelihood:

```
# Given:  
# actions - (N*T) x Da tensor of actions  
# states - (N*T) x Ds tensor of states  
# Build the graph:  
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits  
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)  
loss = tf.reduce_mean(negative_likelihoods)  
gradients = loss.gradients(loss, variables)
```

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:  
# actions - (N*T) x Da tensor of actions  
# states - (N*T) x Ds tensor of states  
# q_values - (N*T) x 1 tensor of estimated state-action values  
# Build the graph:  
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits  
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)  
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)  
loss = tf.reduce_mean(weighted_negative_likelihoods)  
gradients = loss.gradients(loss, variables)
```

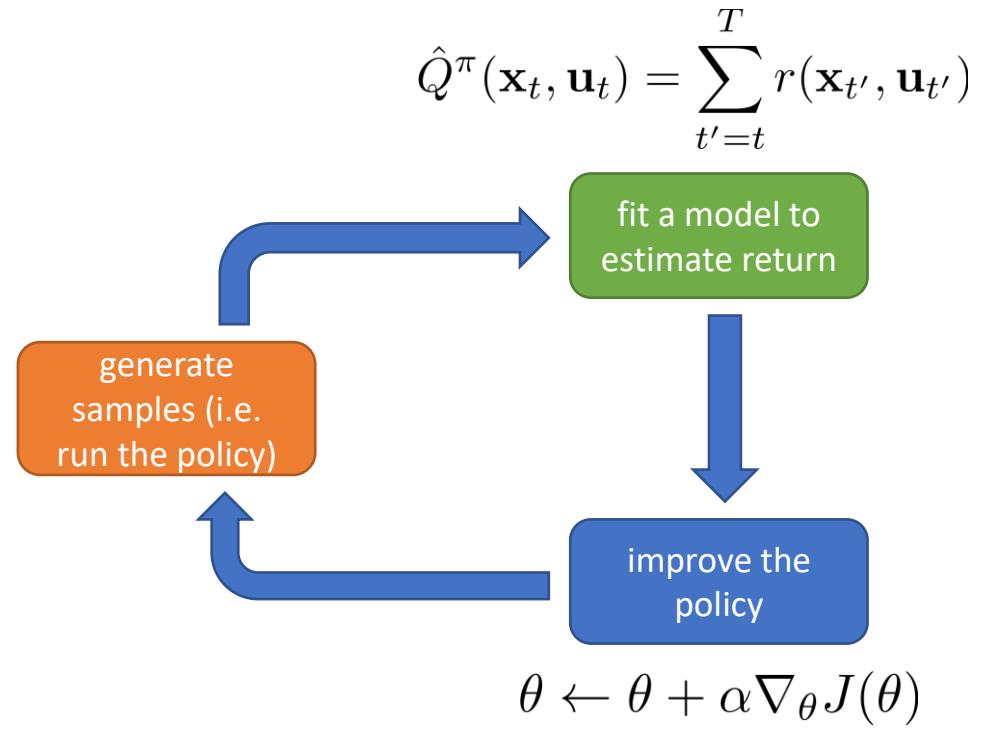
$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Policy gradient in practice

- Remember that the gradient has high variance
 - This isn't the same as supervised learning!
 - Gradients will be really noisy!
- Consider using much larger batches
- Tweaking learning rates is very hard
 - Adaptive step size rules like ADAM can be OK-ish
 - We'll learn about policy gradient-specific learning rate adjustment methods later!

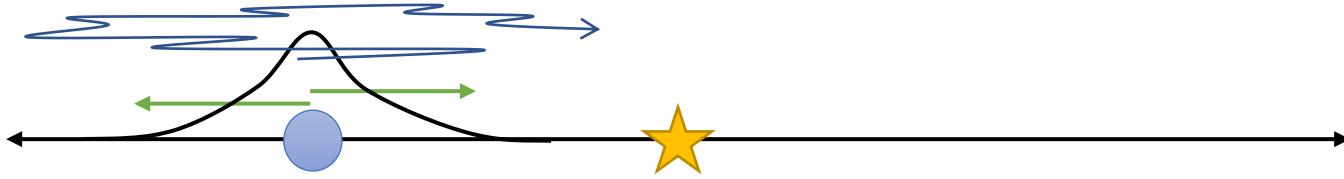
Review

- Policy gradient is on-policy
- Can derive off-policy variant
 - Use importance sampling
 - Exponential scaling in T
 - Can ignore state portion (approximation)
- Can implement with automatic differentiation – need to know what to backpropagate
- Practical considerations: batch size, learning rates, optimizers



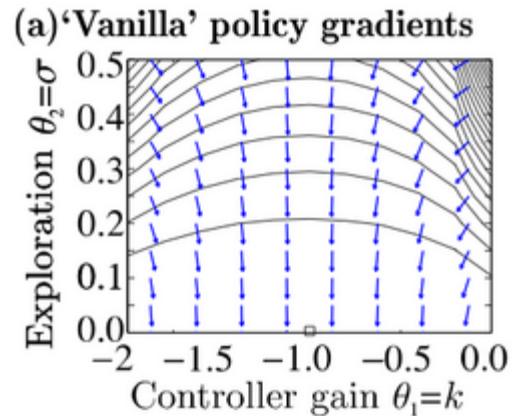
Advanced Policy Gradients

What else is wrong with the policy gradient?



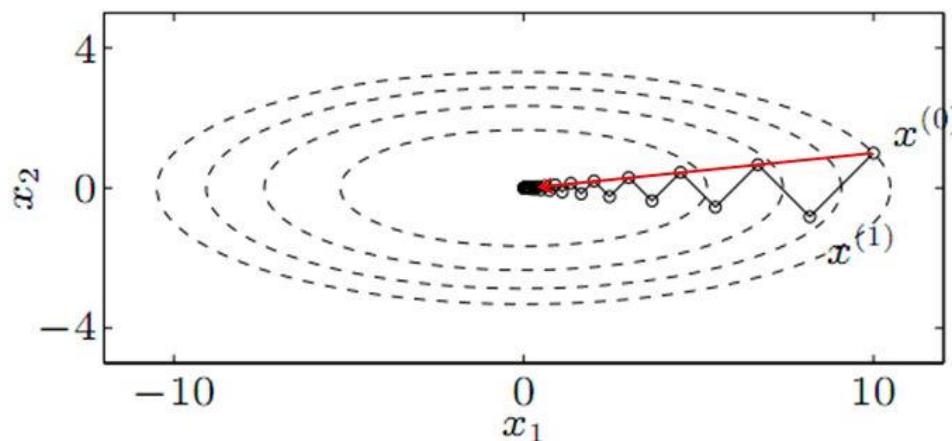
$$r(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^2 - \mathbf{a}_t^2$$

$$\log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2\sigma^2}(k\mathbf{s}_t - \mathbf{a}_t)^2 + \text{const} \quad \theta = (k, \sigma)$$



(image from Peters & Schaal 2008)

Essentially the same
problem as this:



Covariant/natural policy gradient

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

$$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$$

some parameters change probabilities a lot more than others!

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta) \text{ s.t. } \underline{\|\theta' - \theta\|^2 \leq \epsilon}$$

controls how far we go

can we *rescale* the gradient so this doesn't happen?

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta) \text{ s.t. } \underline{D(\pi_{\theta'}, \pi_{\theta}) \leq \epsilon}$$

parameterization-independent divergence measure

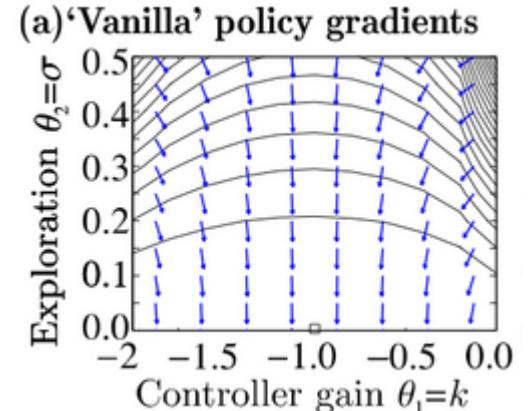
$$\text{usually KL-divergence: } D_{\text{KL}}(\pi_{\theta'} \| \pi_{\theta}) = E_{\pi_{\theta'}} [\log \pi_{\theta} - \log \pi_{\theta'}]$$

$$D_{\text{KL}}(\pi_{\theta'} \| \pi_{\theta}) \approx (\theta' - \theta)^T \underline{\mathbf{F}}(\theta' - \theta)$$

Fisher-information matrix

$$\mathbf{F} = E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\mathbf{a} | \mathbf{s}) \nabla_{\theta} \log \pi_{\theta}(\mathbf{a} | \mathbf{s})^T]$$

can estimate with samples



Covariant/natural policy gradient

$$D_{\text{KL}}(\pi_{\theta'} \| \pi_{\theta}) \approx (\theta' - \theta)^T \mathbf{F}(\theta' - \theta)$$

$$\mathbf{F} = E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s})^T]$$

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta) \text{ s.t. } D(\pi_{\theta'}, \pi_{\theta}) \leq \epsilon$$

$$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_{\theta} J(\theta)$$

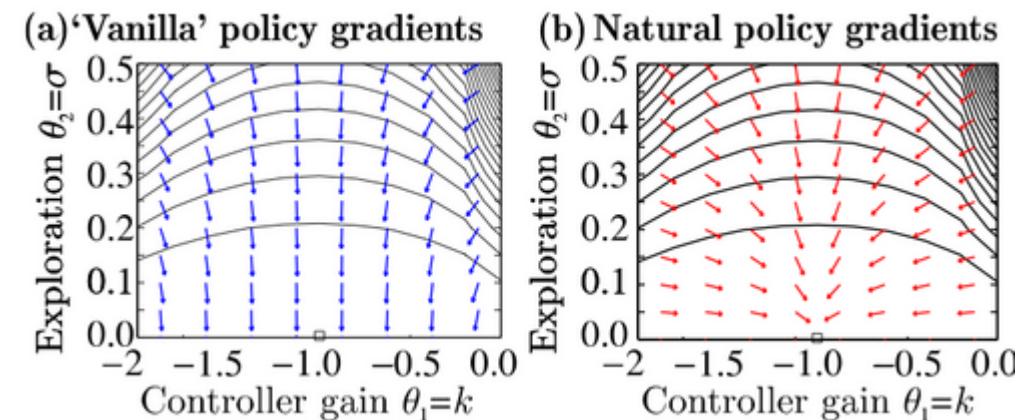
natural gradient: pick α

trust region policy optimization: pick ϵ

can solve for optimal α while solving $\mathbf{F}^{-1} \nabla_{\theta} J(\theta)$

conjugate gradient works well for this

see Schulman, L., Moritz, Jordan, Abbeel (2015) Trust region policy optimization



(figure from Peters & Schaal 2008)

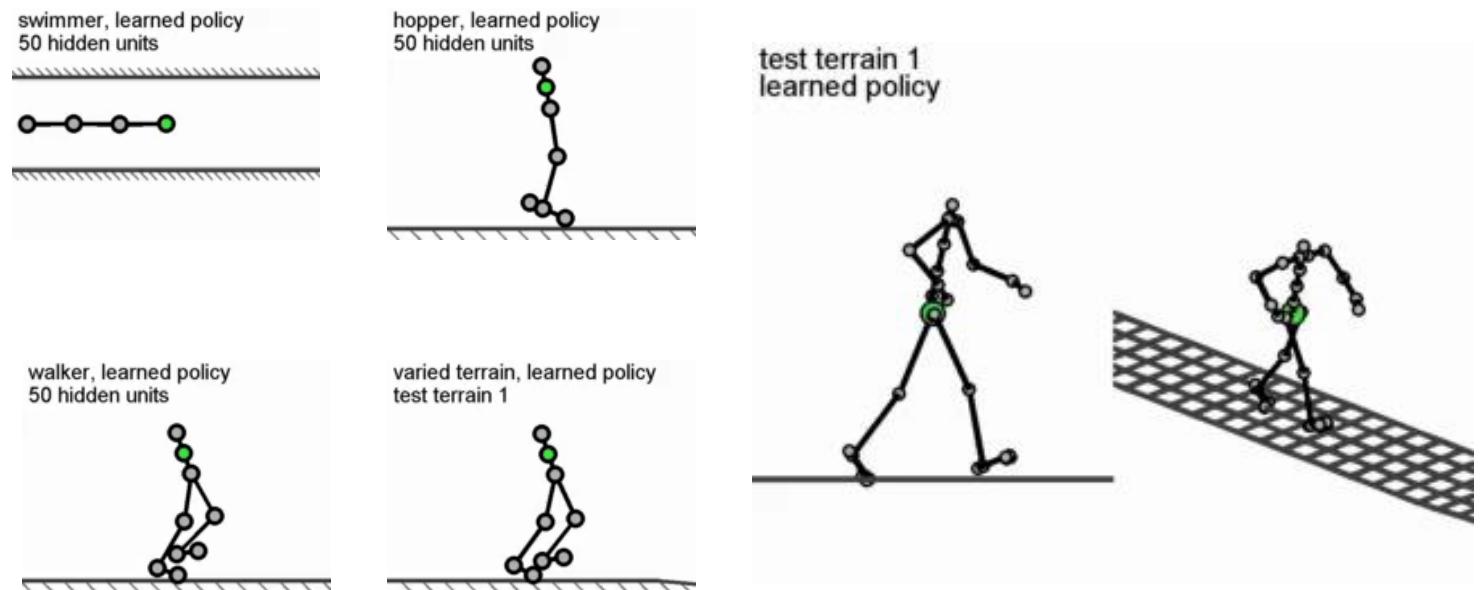
Advanced policy gradient topics

- What more is there?
- Next time: introduce value functions and Q-functions
- Later in the class: more on natural gradient and automatic step size adjustment

Example: policy gradient with importance sampling

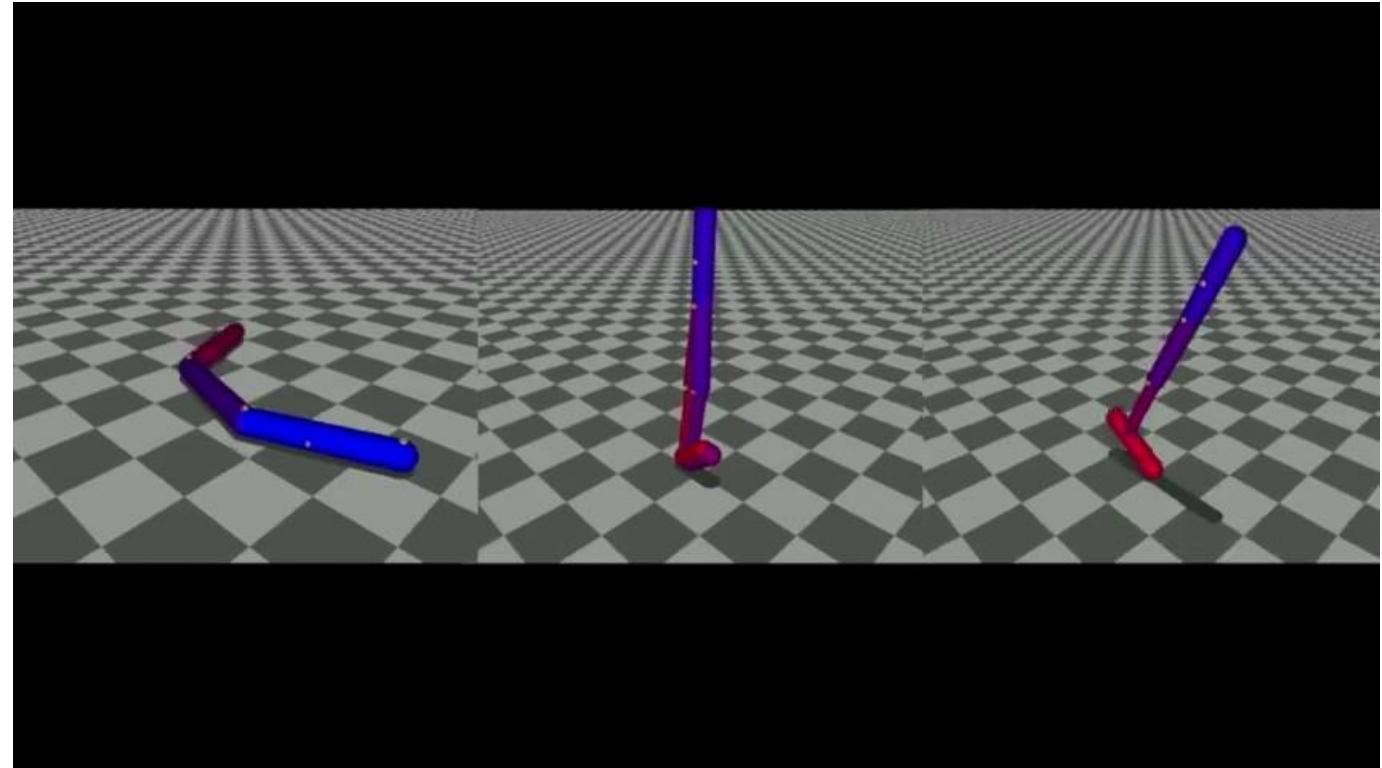
$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

- Incorporate example demonstrations using importance sampling
- Neural network policies



Example: trust region policy optimization

- Natural gradient with automatic step adjustment
- Discrete and continuous actions
- Code available (see Duan et al. '16)



Policy gradients suggested readings

- Classic papers
 - Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm
 - Baxter & Bartlett (2001). Infinite-horizon policy-gradient estimation: temporally decomposed policy gradient (not the first paper on this! see actor-critic section later)
 - Peters & Schaal (2008). Reinforcement learning of motor skills with policy gradients: very accessible overview of optimal baselines and natural gradient
- Deep reinforcement learning policy gradient papers
 - Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient (unrelated to later discussion of guided policy search)
 - Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
 - Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

Actor-Critic Algorithms

CS 285

Instructor: Sergey Levine
UC Berkeley



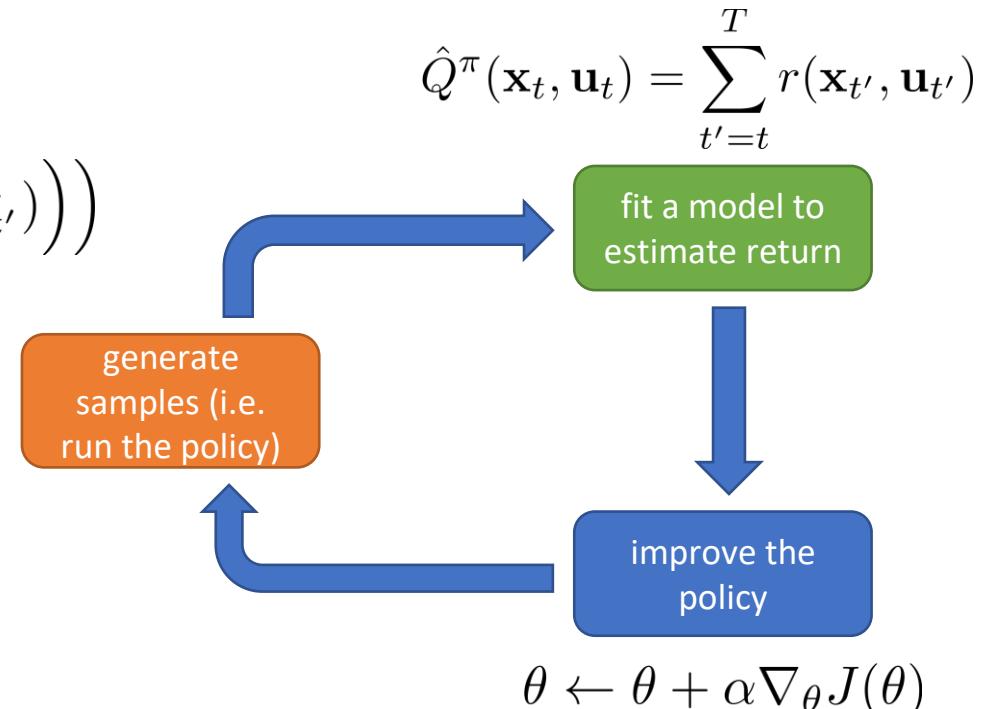
Recap: policy gradients

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i) \right) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

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

“reward to go”



Improving the policy gradient

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

“reward to go”

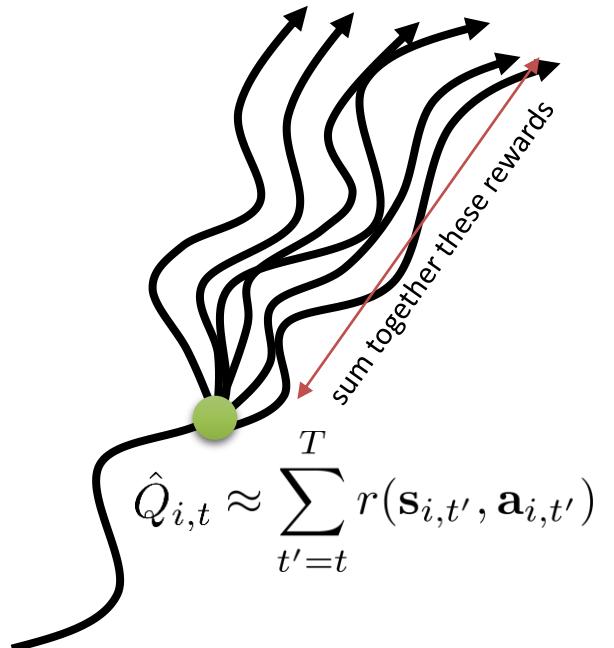
$$\hat{Q}_{i,t}$$

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

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



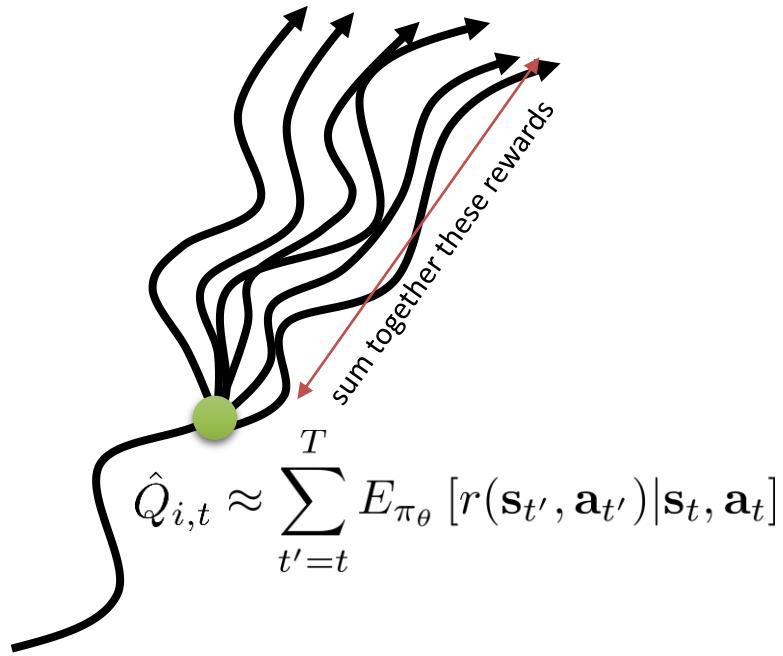
What about the baseline?

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

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

$$b_t = \frac{1}{N} \sum_i Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \quad \text{average what?}$$

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



State & state-action value functions

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

fit Q^π , V^π , or A^π

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

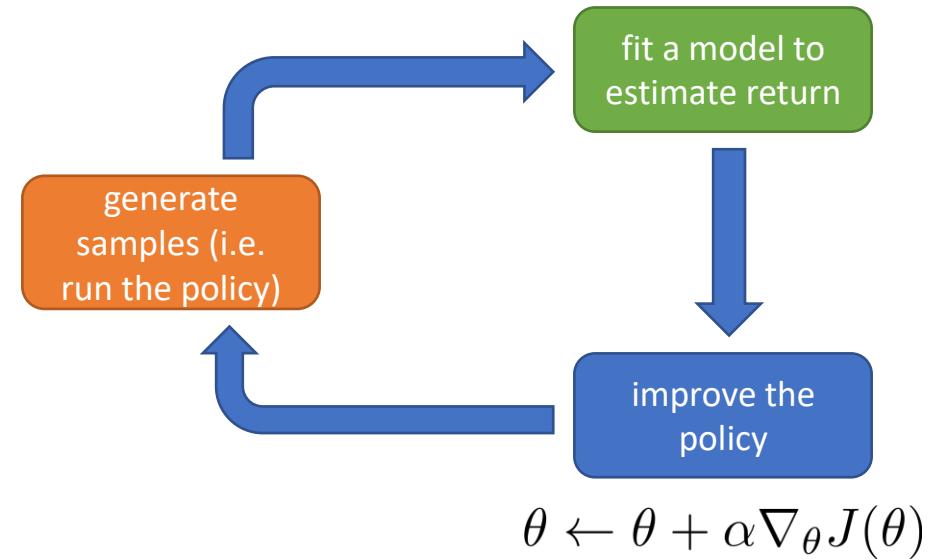
$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t): \text{how much better } \mathbf{a}_t \text{ is}$$

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



the better this estimate, the lower the variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\underbrace{\sum_{t'=1}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - b}_{\text{unbiased, but high variance single-sample estimate}} \right)$$



Value function fitting

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

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

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

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

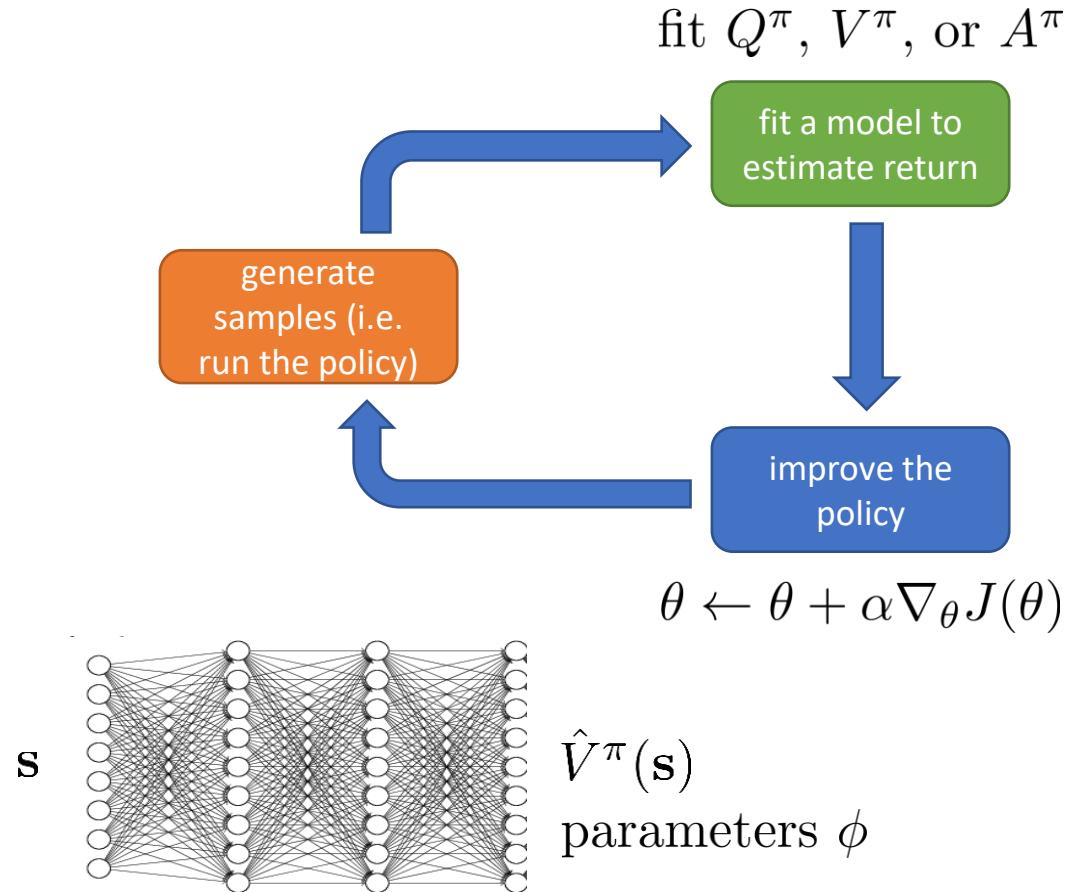
fit *what* to *what*?

Q^π, V^π, A^π ?

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \sum_{t'=t+1}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$$

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

let's just fit $V^\pi(\mathbf{s})$!



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

Policy evaluation

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

$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} [V^\pi(\mathbf{s}_1)]$$

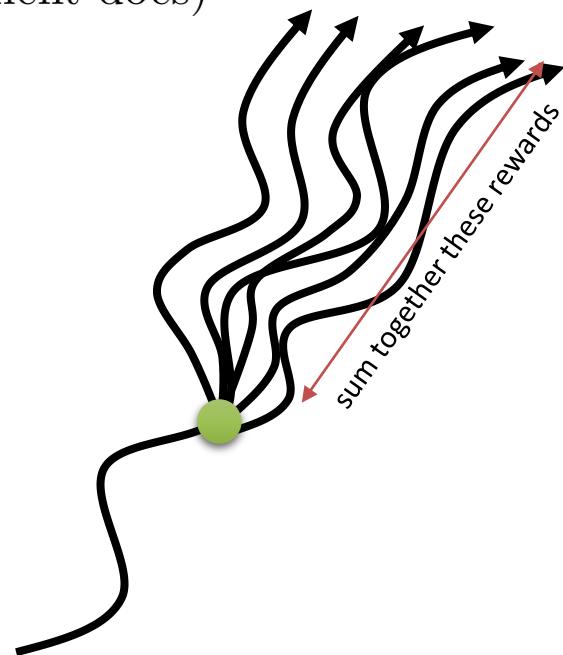
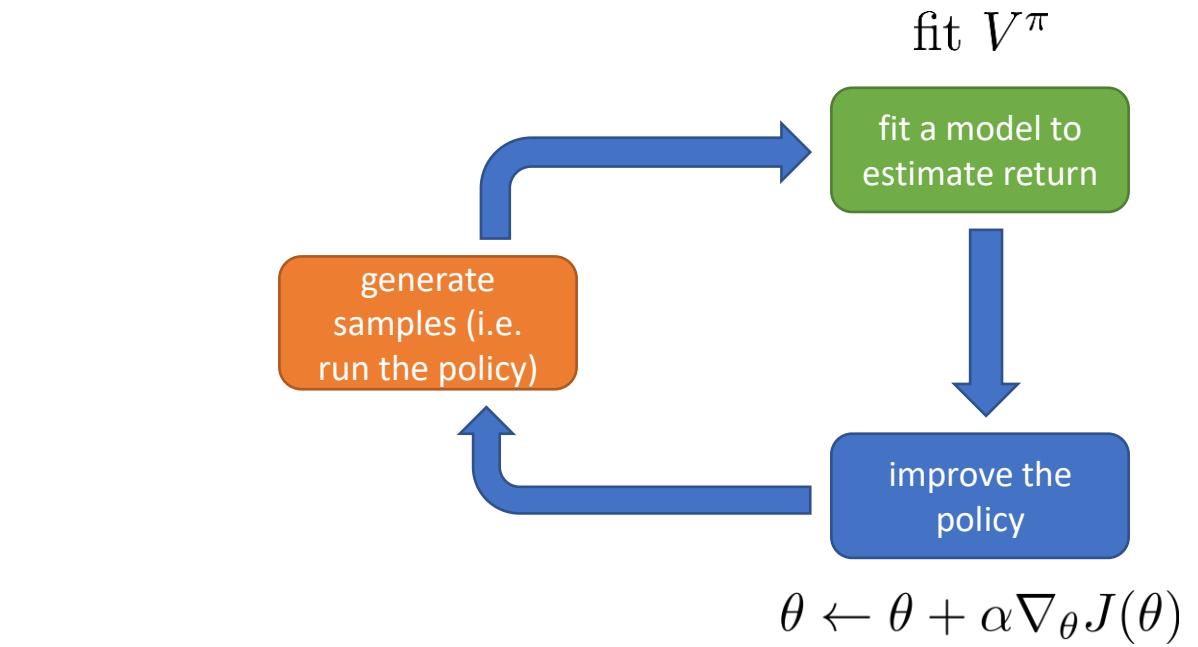
how can we perform policy evaluation?

Monte Carlo policy evaluation (this is what policy gradient does)

$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

(requires us to reset the simulator)



Monte Carlo evaluation with function approximation

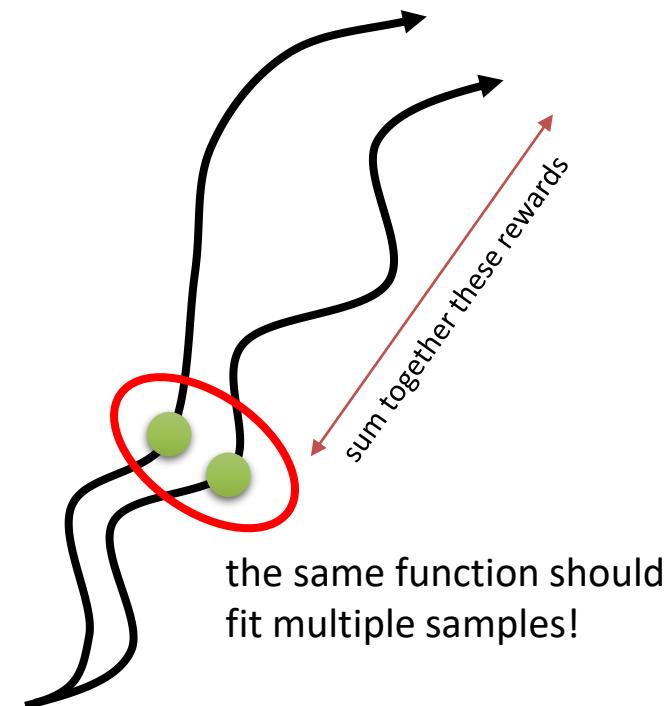
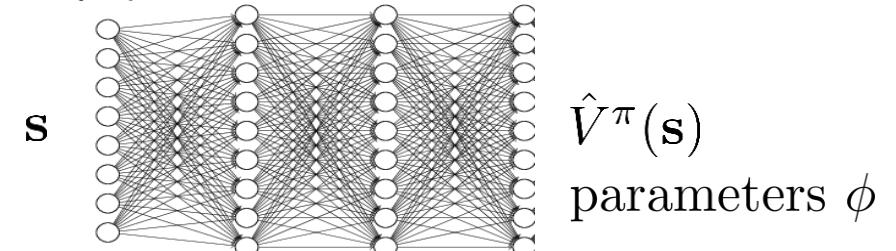
$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

not as good as this: $V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$

but still pretty good!

training data: $\left\{ \left(\mathbf{s}_{i,t}, \underbrace{\sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})}_{y_{i,t}} \right) \right\}$

supervised regression: $\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$



Can we do better?

ideal target: $y_{i,t} = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_{i,t}] \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^\pi(\mathbf{s}_{i,t+1}) \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$

Monte Carlo target: $y_{i,t} = \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$

directly use previous fitted value function!

training data: $\left\{ \left(\mathbf{s}_{i,t}, \underbrace{r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})}_{y_{i,t}} \right) \right\}$

supervised regression: $\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$

sometimes referred to as a “bootstrapped” estimate

Policy evaluation examples

TD-Gammon, Gerald Tesauro 1992

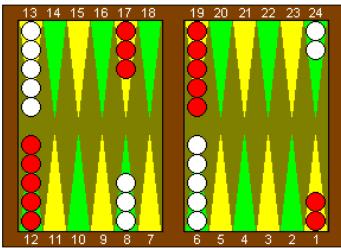


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference, 13-9, 24-23. TD-Gammon's analysis is given in Table 2.

AlphaGo, Silver et al. 2016

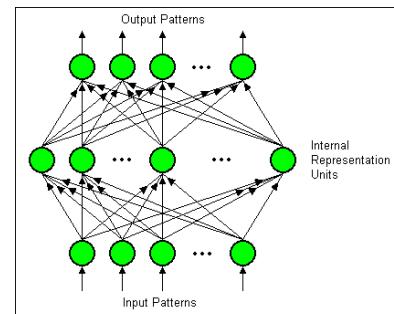


Figure 1. An illustration of the multilayer perceptron architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:

expected outcome given board state

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:

expected outcome given board state

From Evaluation to Actor Critic

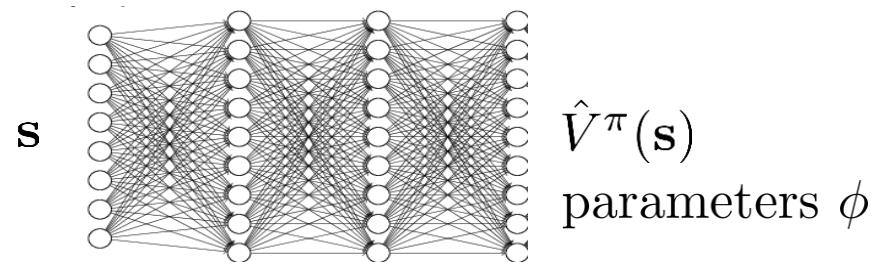
An actor-critic algorithm

batch actor-critic algorithm:

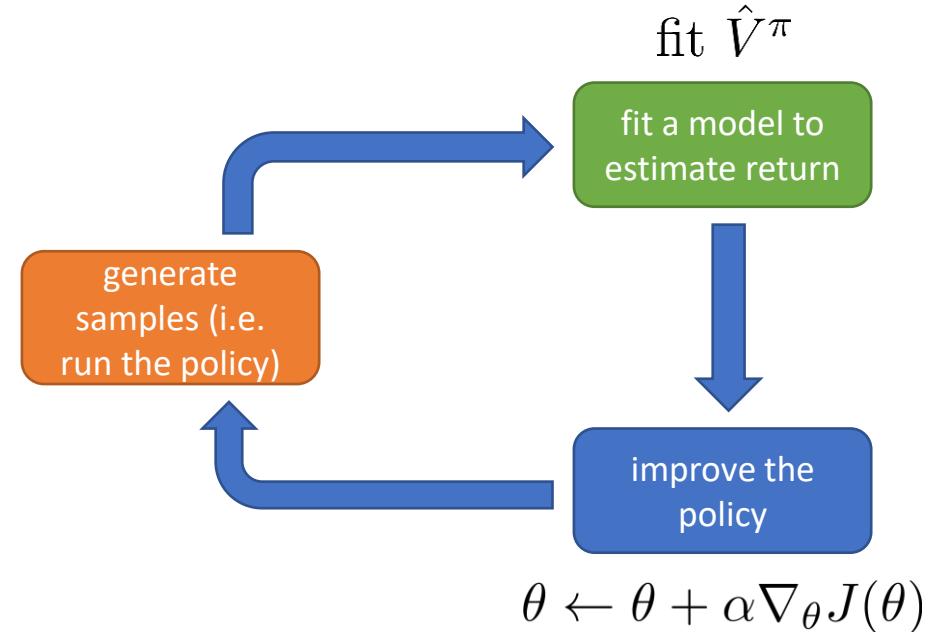
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$y_{i,t} \approx \sum_{t'=t}^T r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$



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



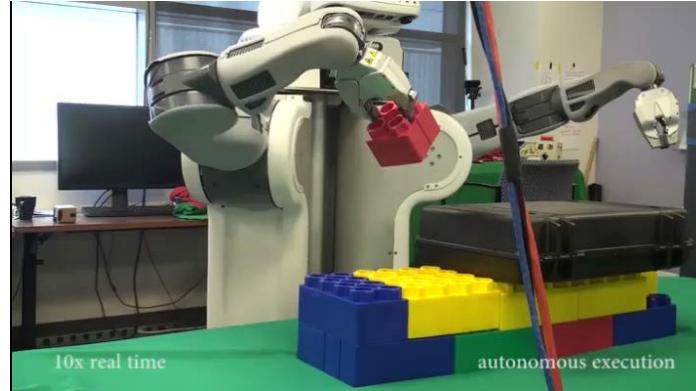
Aside: discount factors

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

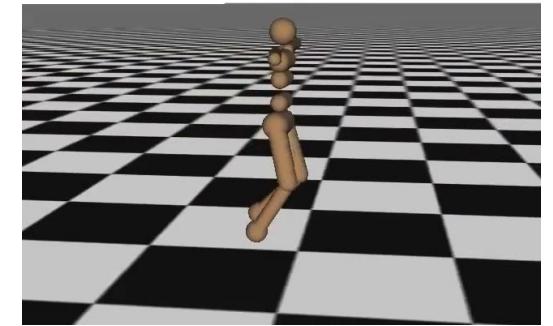
what if T (episode length) is ∞ ?

\hat{V}_ϕ^π can get infinitely large in many cases



episodic tasks

Iteration 2000



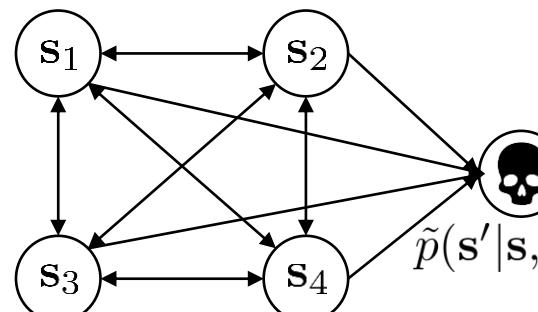
continuous/cyclical tasks

simple trick: better to get rewards sooner than later

γ changes the MDP:

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

↑
discount factor $\gamma \in [0, 1]$ (0.99 works well)



Aside: discount factors for policy gradients

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

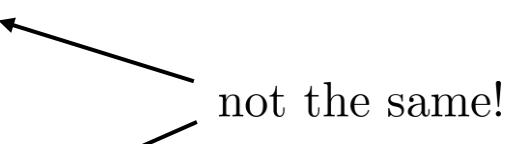
$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

with critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \underbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{i,t}) \right)}_{\hat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

what about (Monte Carlo) policy gradients?

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



not the same!

option 2: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^T \gamma^{t-1} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$

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

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

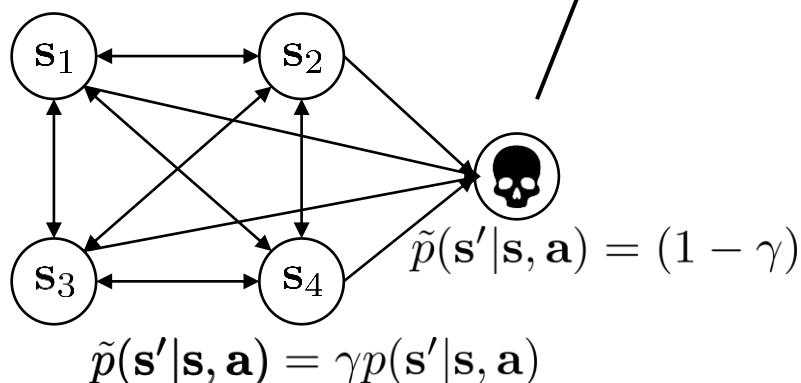
Which version is the right one?

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

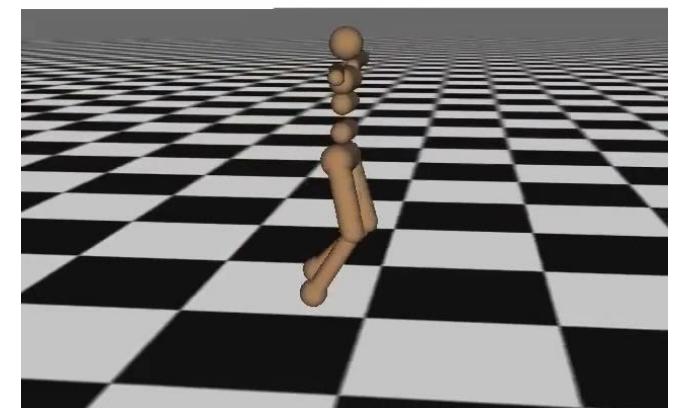
this is what we actually use...
why?

option 2: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^T \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$

later steps don't matter if you're dead!

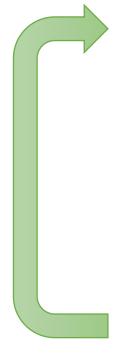


Iteration 2000

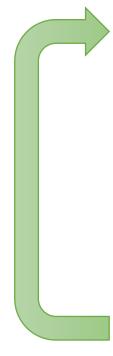


Actor-critic algorithms (with discount)

batch actor-critic algorithm:

- 
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
 2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
 3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
 4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

online actor-critic algorithm:

- 
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
 2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
 3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
 4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

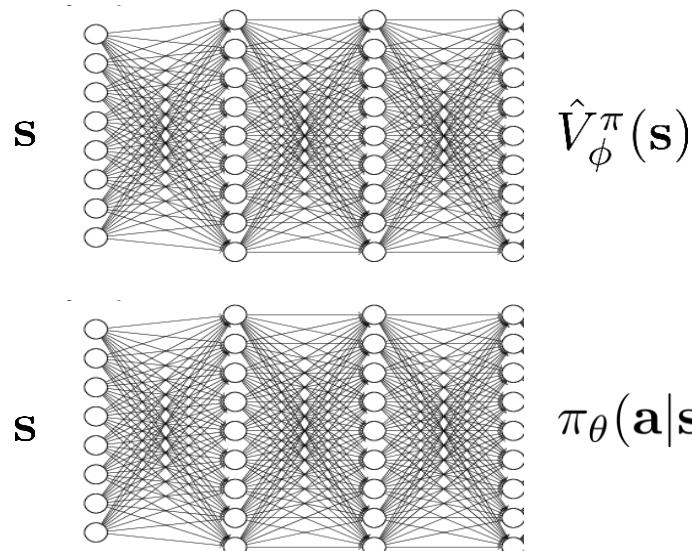
Actor-Critic Design Decisions

Architecture design

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

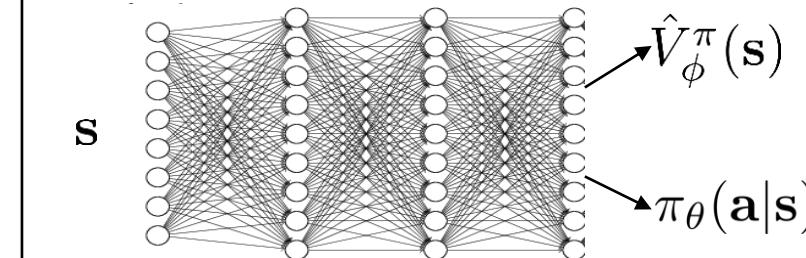
two network design



+ simple & stable

- no shared features between actor & critic

shared network design

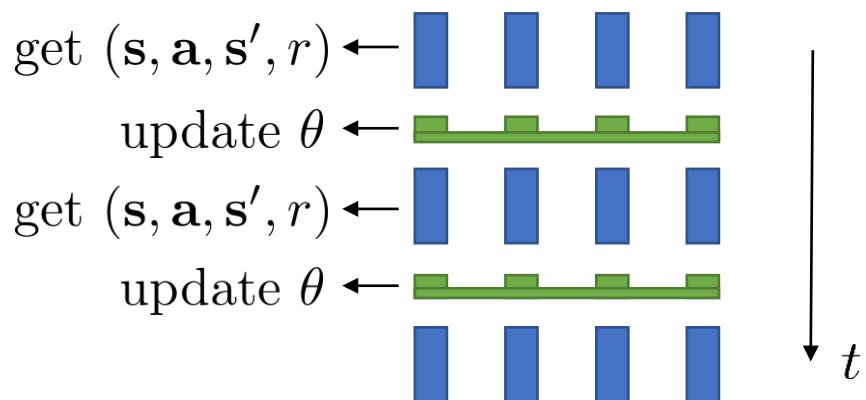


Online actor-critic in practice

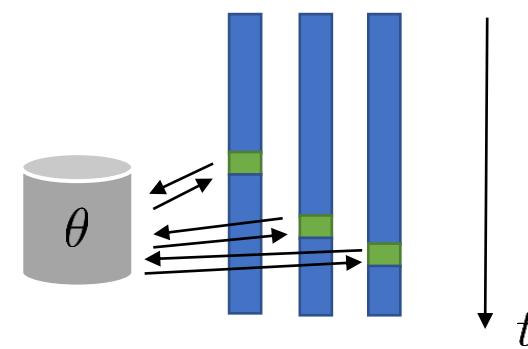
online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}') \leftarrow$ works best with a batch (e.g., parallel workers)
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a}) \leftarrow$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

synchronized parallel actor-critic



asynchronous parallel actor-critic

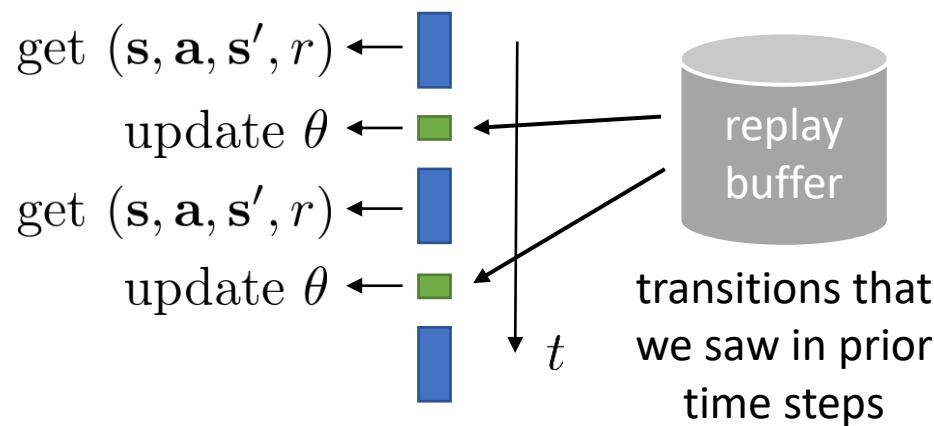


Can we remove the on-policy assumption entirely?

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
 2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
 3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
 4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- form a **batch** by
using old previously
seen transitions**

off-policy actor-critic

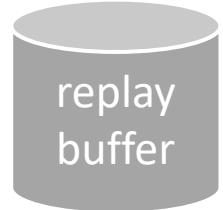


Let's see what that looks like

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in \mathcal{R}
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer \mathcal{R}
3. update \hat{V}_ϕ^π using targets $y_i = r_i + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i)$ for each \mathbf{s}_i
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

not the action π_θ would have taken!



$$\mathcal{L}(\phi) = \frac{1}{N} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

↑
batch size

This algorithm is broken!

Can you spot the problems?

Fixing the value function

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in \mathcal{R}
 2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer \mathcal{R}
 3. update \hat{V}_ϕ^π using targets $y_i = r_i + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i)$ for each \mathbf{s}_i
 4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i) \leftarrow \hat{V}_\phi^\pi(\mathbf{s}_i)$
 5. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
 6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- not the right target value
- not the action π_θ would have taken!
- where does this come from?

$$\begin{aligned} 3. \text{ update } \hat{Q}_\phi^\pi \text{ using targets } y_i &= r_i + \gamma \hat{V}_\phi^\pi(\mathbf{s}'_i) \text{ for each } \mathbf{s}_i, \mathbf{a}_i \\ &= r_i + \gamma \hat{Q}_\phi^\pi(\mathbf{s}'_i, \mathbf{a}'_i) \end{aligned}$$

↑
not from replay buffer \mathcal{R} !

$$\mathbf{a}'_i \sim \pi_\theta(\mathbf{a}'_i|\mathbf{s}'_i)$$

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

$$\cancel{V^\pi(\mathbf{s}_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t]}$$

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

“total reward we get if we take \mathbf{a}_t in \mathbf{s}_t ...
... and then follow the policy π ”

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_i \left\| \hat{Q}_\phi^\pi(\mathbf{s}_i, \mathbf{a}_i) - y_i \right\|^2$$

Fixing the policy update

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in \mathcal{R}
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer \mathcal{R}
3. update \hat{Q}_ϕ^π using targets $y_i = r_i + \gamma \hat{Q}_\phi^\pi(\mathbf{s}'_i, \mathbf{a}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = Q(\mathbf{s}_i, \mathbf{a}_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

not the action π_θ would have taken!

use the same trick, but this time for \mathbf{a}_i rather than \mathbf{a}'_i !

sample $\mathbf{a}_i^\pi \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i^\pi|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i^\pi)$$

↑
not from replay buffer $\mathcal{R}!$

higher variance, but convenient
why is higher variance OK here?

$$\text{in practice: } \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i^\pi|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}_i^\pi)$$

What else is left?

online actor-critic algorithm:

- 
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in \mathcal{R}
 2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer \mathcal{R}
 3. update \hat{Q}_ϕ^π using targets $y_i = r_i + \gamma \hat{Q}_\phi^\pi(\mathbf{s}'_i, \mathbf{a}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
 4. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i^\pi | \mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}_i^\pi)$ where $\mathbf{a}_i^\pi \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Is there any remaining problem?

\mathbf{s}_i didn't come from $p_\theta(\mathbf{s})$

nothing we can do here, just accept it

intuition: we want optimal policy on $p_\theta(\mathbf{s})$

but we get optimal policy on a *broader* distribution

Some implementation details

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in \mathcal{R}
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer \mathcal{R}
3. update \hat{Q}_ϕ^π using targets $y_i = r_i + \gamma \hat{Q}_\phi^\pi(\mathbf{s}'_i, \mathbf{a}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
4. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i^\pi | \mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}_i^\pi)$ where $\mathbf{a}_i^\pi \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

could also use **reparameterization trick**
to better estimate the integral

Example practical algorithm:

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. 2018.

We'll also learn about algorithms that do this with deterministic policies later!

lots of fancier ways to fit Q-functions
(more on this in next two lectures)

Critics as Baselines

Critics as state-dependent baselines

Actor-critic: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1}) - \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t}) \right)$

+ lower variance (due to critic)
- not unbiased (if the critic is not perfect)

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

+ no bias
- higher variance (because single-sample estimate)

can we use \hat{V}_{ϕ}^{π} and still keep the estimator unbiased?

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

+ no bias
+ lower variance (baseline is closer to rewards)

Control variates: action-dependent baselines

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

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

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

$$\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - V_\phi^\pi(\mathbf{s}_t)$$

- + no bias
- higher variance (because single-sample estimate)

$$\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - Q_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

- + goes to zero in expectation if critic is correct!
- not correct

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\hat{Q}_{i,t} - Q_\phi^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right) + \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta E_{\mathbf{a} \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_{i,t})} [Q_\phi^\pi(\mathbf{s}_{i,t}, \mathbf{a}_t)]$$

use a critic *without* the bias (still unbiased),
provided second term can be evaluated

Gu et al. 2016 (Q-Prop)

Eligibility traces & n-step returns

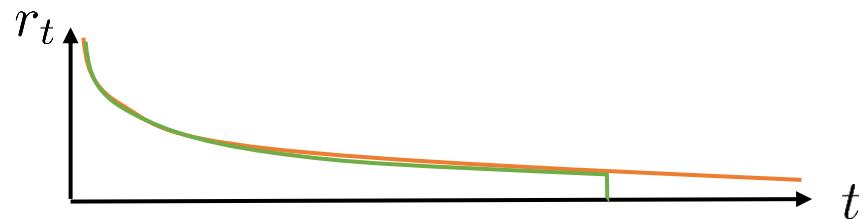
$$\hat{A}_C^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1}) - \hat{V}_\phi^\pi(s_t)$$

- + lower variance
- higher bias if value is wrong (it always is)

$$\hat{A}_{MC}^\pi(s_t, a_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t)$$

- + no bias
- higher variance (because single-sample estimate)

Can we combine these two, to control bias/variance tradeoff?

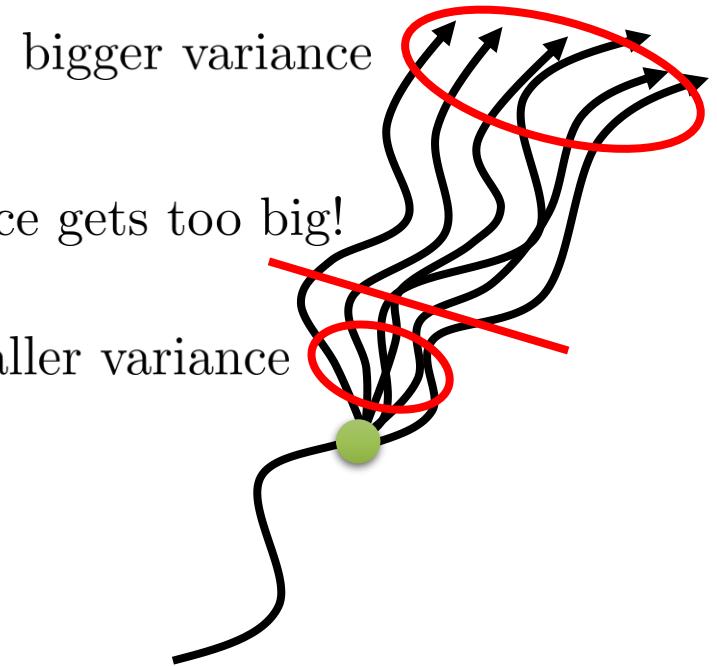


$$\hat{A}_n^\pi(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t) + \gamma^n \hat{V}_\phi^\pi(s_{t+n})$$

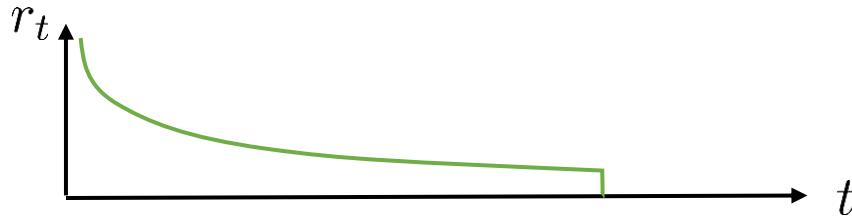
choosing $n > 1$ often works better!

cut here before variance gets too big!

smaller variance



Generalized advantage estimation



Do we have to choose just one n ?

Cut everywhere all at once!

$$\hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n})$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

weighted combination of n -step returns

How to weight?

Mostly prefer cutting earlier (less variance)

$$w_n \propto \lambda^{n-1}$$

exponential falloff

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+1}) + \lambda(r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+2}) + \lambda r(\mathbf{s}_{t+2}, \mathbf{a}_{t+2}) + \dots))$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma \lambda)^{t'-t} \delta_{t'}$$

$$\delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t'+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{t'})$$

similar effect as discount!

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

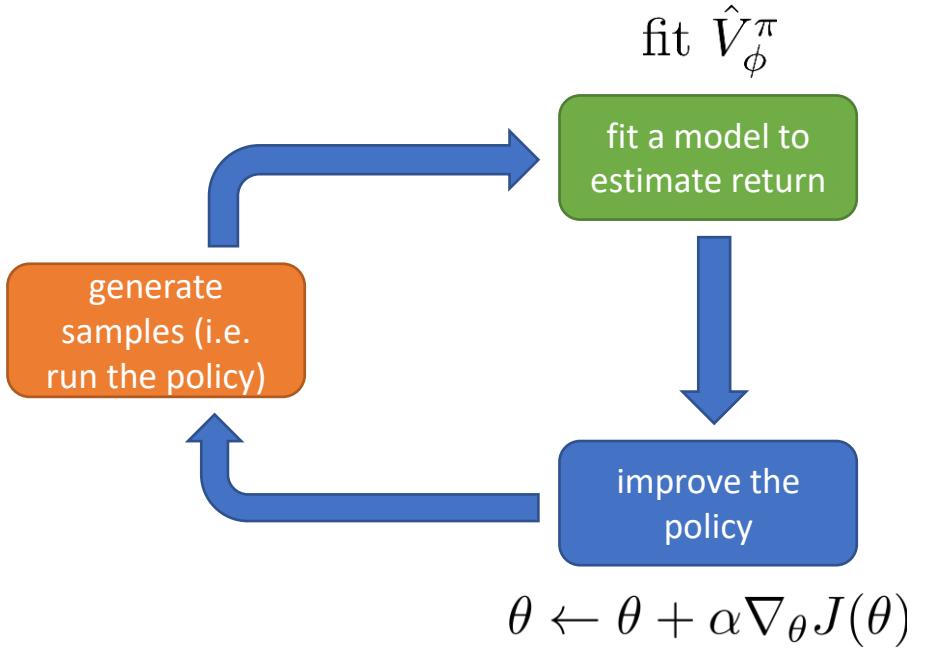
remember this?

discount = variance reduction!

Review, Examples, and Additional Readings

Review

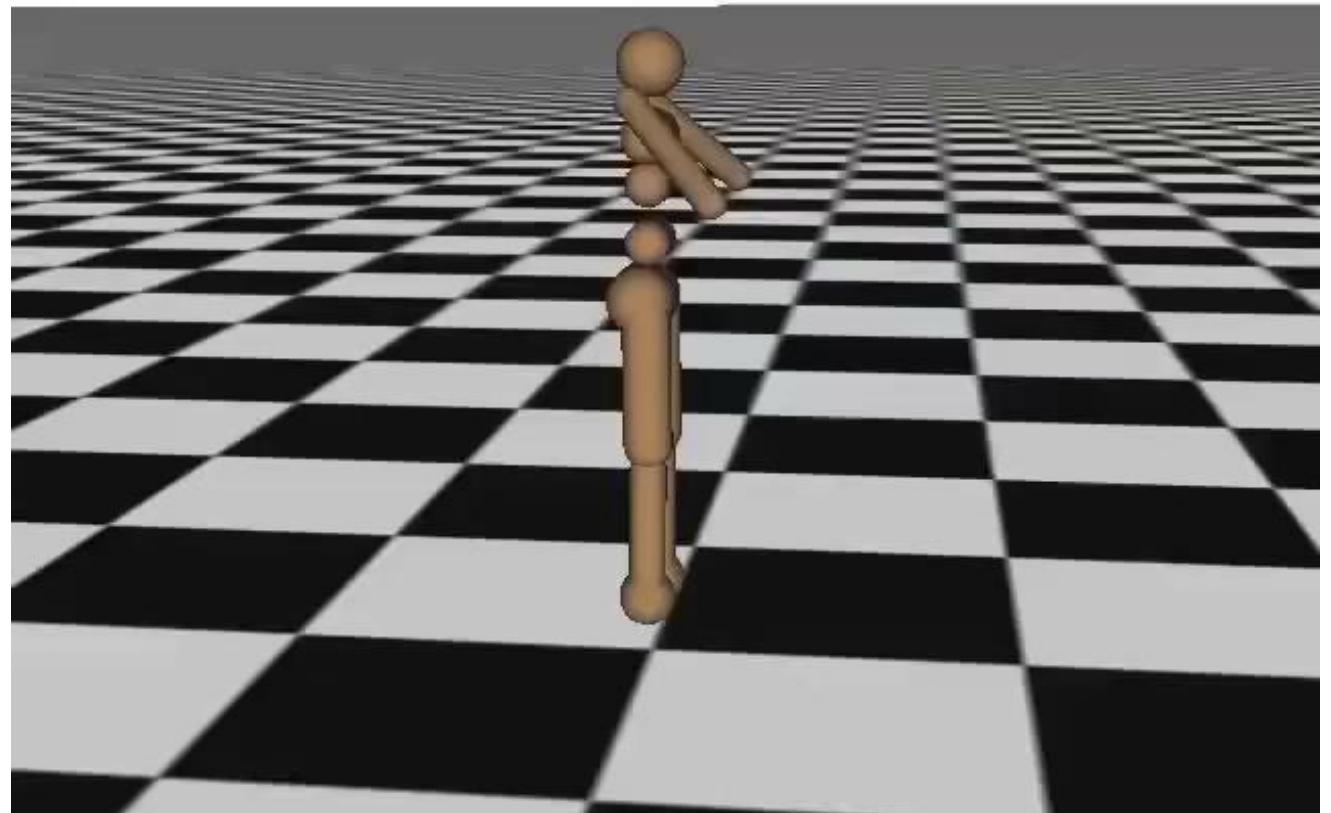
- Actor-critic algorithms:
 - Actor: the policy
 - Critic: value function
 - Reduce variance of policy gradient
- Policy evaluation
 - Fitting value function to policy
- Discount factors
 - Carpe diem Mr. Robot 
 - ...but also a variance reduction trick
- Actor-critic algorithm design
 - One network (with two heads) or two networks
 - Batch-mode, or online (+ parallel)
- State-dependent baselines
 - Another way to use the critic
 - Can combine: n-step returns or GAE



Actor-critic examples

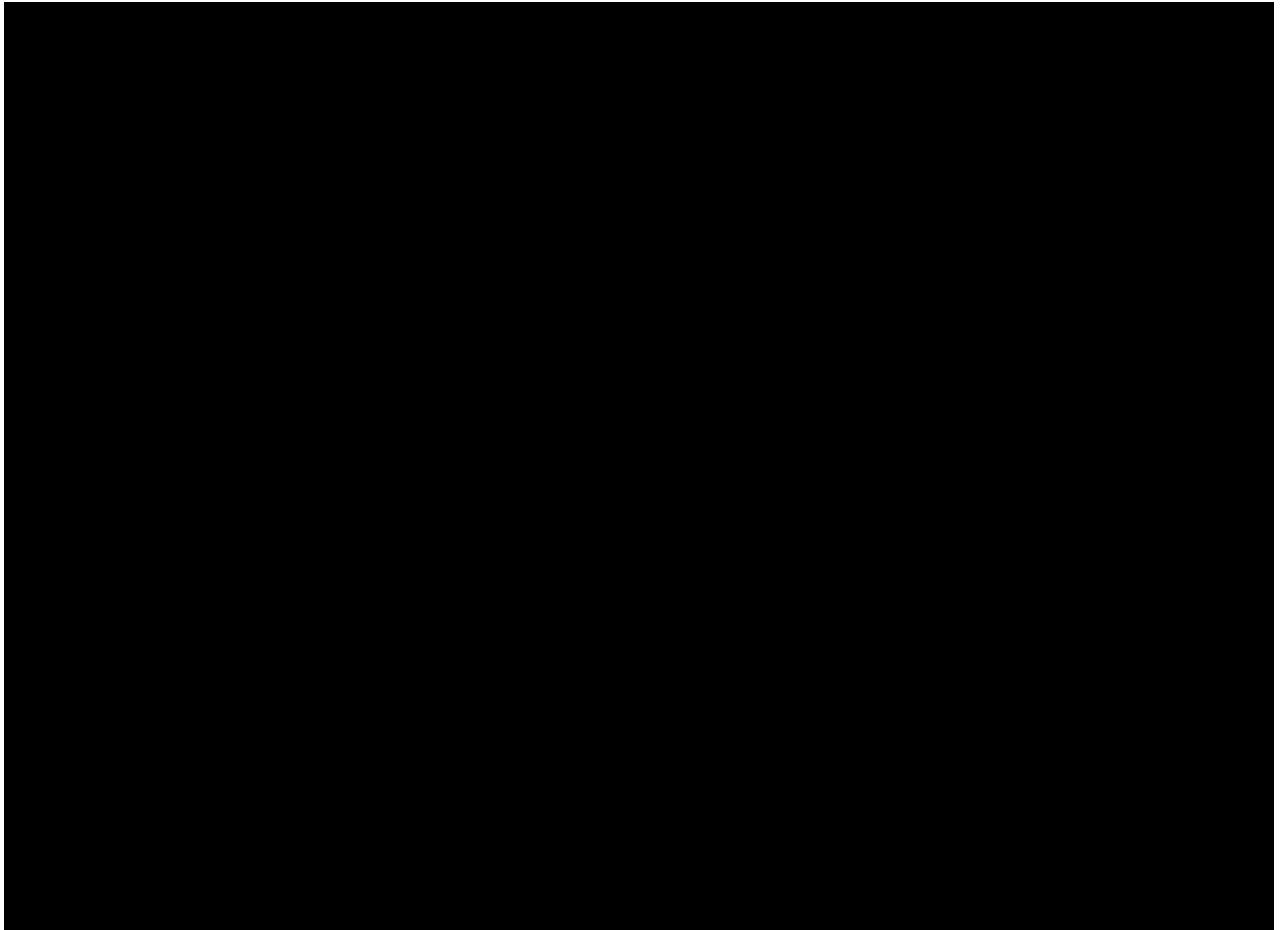
- High dimensional continuous control with generalized advantage estimation
(Schulman, Moritz, L., Jordan, Abbeel '16)
- Batch-mode actor-critic
- Blends Monte Carlo and function approximator estimators (GAE)

Iteration 0



Actor-critic examples

- Asynchronous methods for deep reinforcement learning (Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, Kavukcuoglu '16)
- Online actor-critic, parallelized batch
- N-step returns with $N = 4$
- Single network for actor and critic



Actor-critic suggested readings

- Classic papers
 - Sutton, McAllester, Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation: actor-critic algorithms with value function approximation
- Deep reinforcement learning actor-critic papers
 - Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning: A3C -- parallel online actor-critic
 - Schulman, Moritz, L., Jordan, Abbeel (2016). High-dimensional continuous control using generalized advantage estimation: batch-mode actor-critic with blended Monte Carlo and function approximator returns
 - Gu, Lillicrap, Ghahramani, Turner, L. (2017). Q-Prop: sample-efficient policy-gradient with an off-policy critic: policy gradient with Q-function control variate

Value Function Methods

CS 285

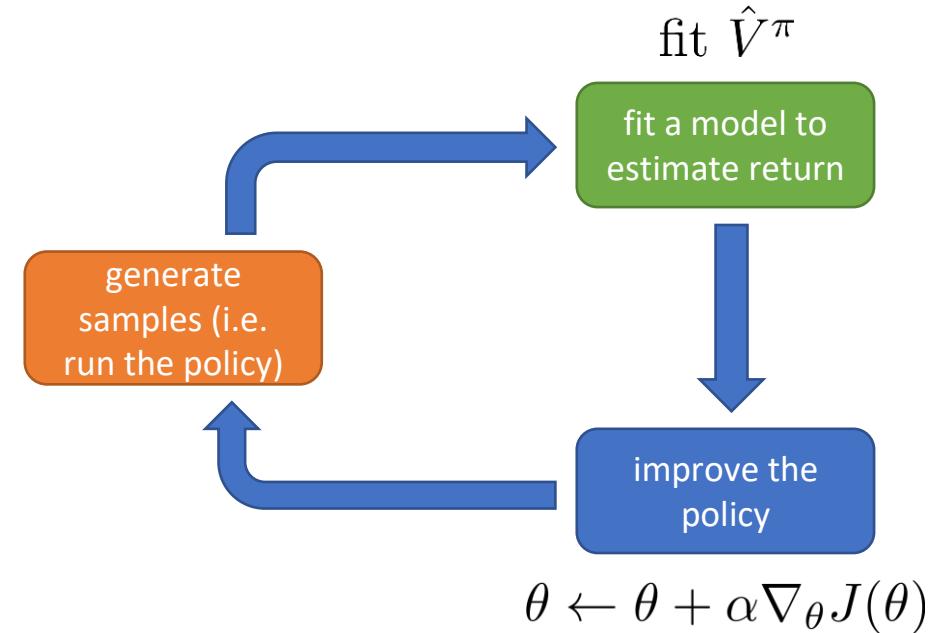
Instructor: Sergey Levine
UC Berkeley



Recap: actor-critic

batch actor-critic algorithm:

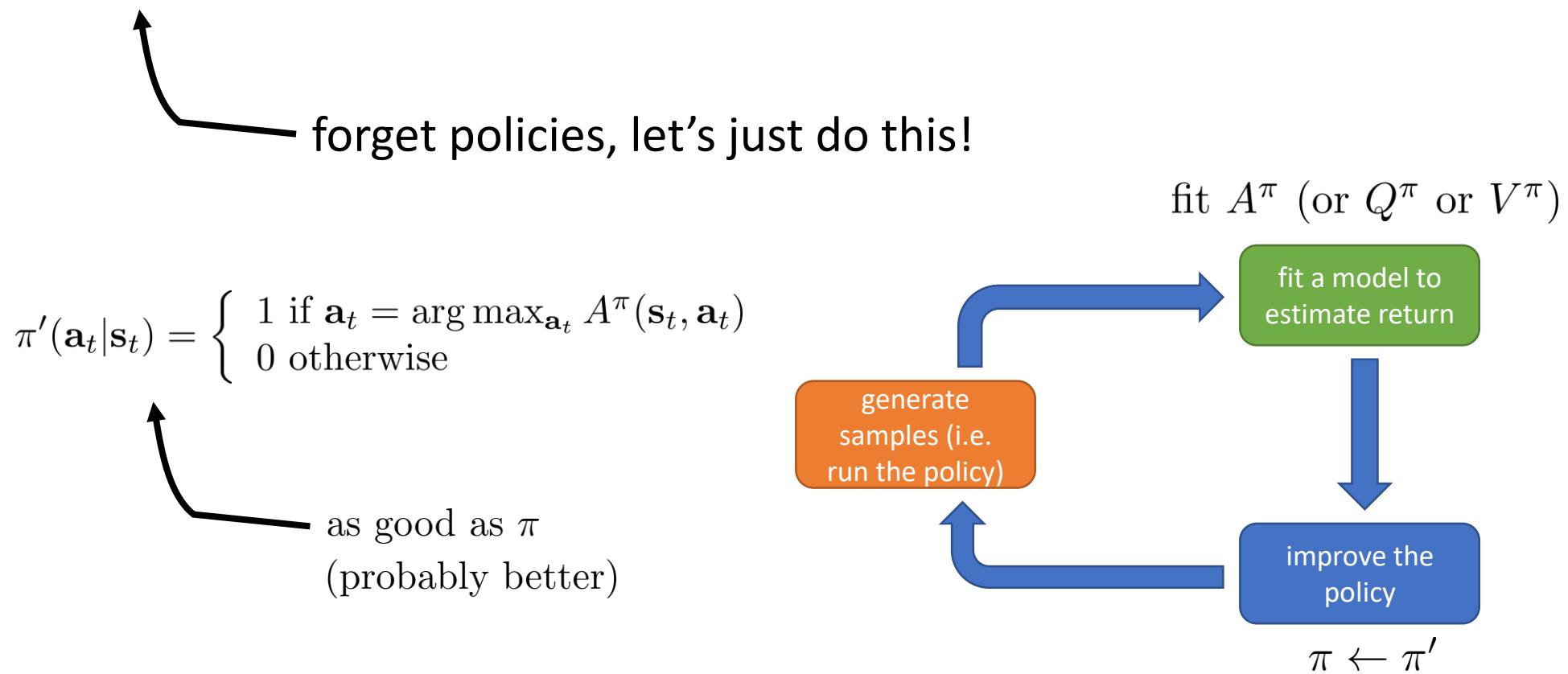
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Can we omit policy gradient completely?

$A^\pi(s_t, a_t)$: how much better is a_t than the average action according to π at least as good as any $a_t \sim \pi(a_t|s_t)$

$\arg \max_{a_t} A^\pi(s_t, a_t)$: best action from s_t , if we then follow π regardless of what $\pi(a_t|s_t)$ is!



Policy iteration

High level idea:

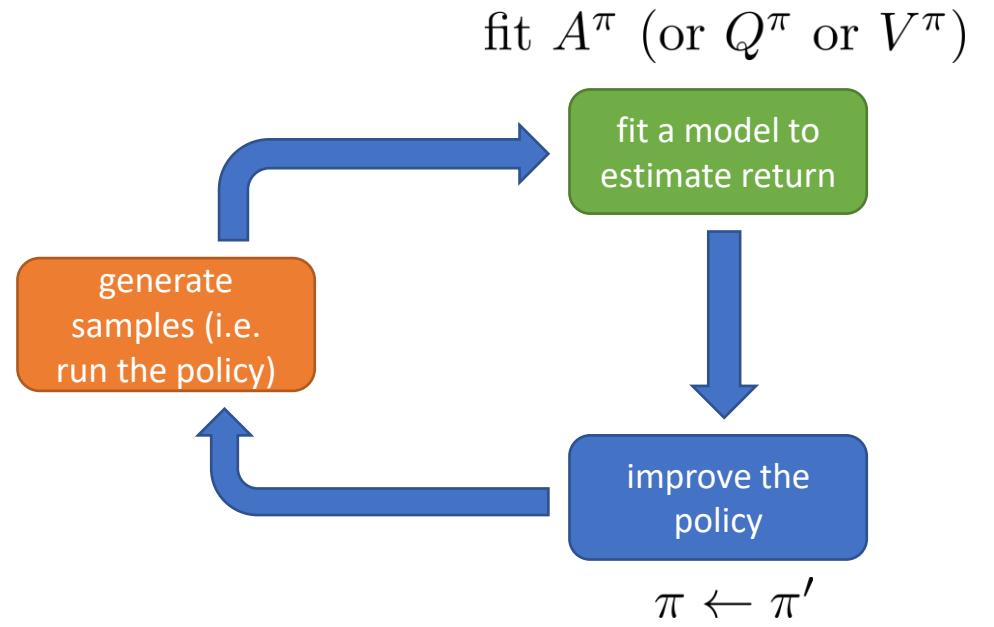
policy iteration algorithm:

- 
1. evaluate $A^\pi(\mathbf{s}, \mathbf{a})$ ← how to do this?
 2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

as before: $A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')]$ – $V^\pi(\mathbf{s})$

let's evaluate $V^\pi(\mathbf{s})$!



Dynamic programming

Let's assume we know $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, and \mathbf{s} and \mathbf{a} are both discrete (and small)

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

16 states, 4 actions per state
can store full $V^\pi(\mathbf{s})$ in a table!
 \mathcal{T} is $16 \times 16 \times 4$ tensor

bootstrapped update: $V^\pi(\mathbf{s}) \leftarrow E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})}[V^\pi(\mathbf{s}')]]$



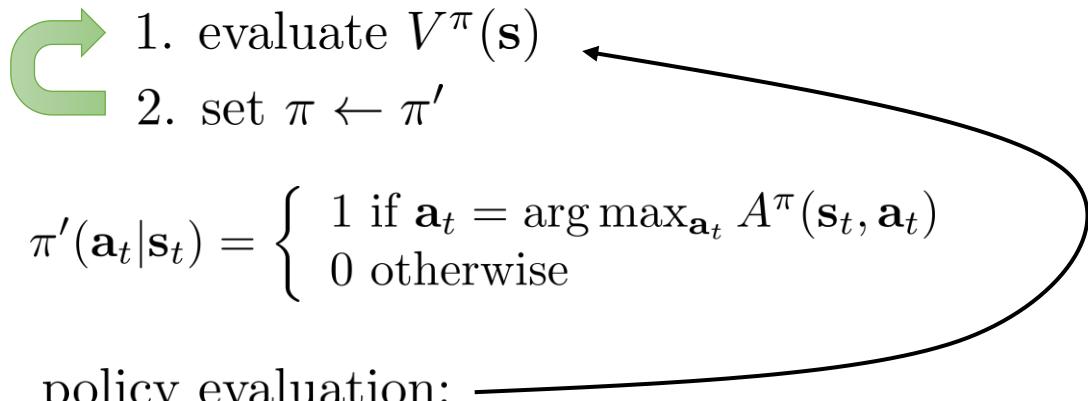
just use the current estimate here

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \longrightarrow \text{deterministic policy } \pi(\mathbf{s}) = \mathbf{a}$$

simplified: $V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))}[V^\pi(\mathbf{s}')]$

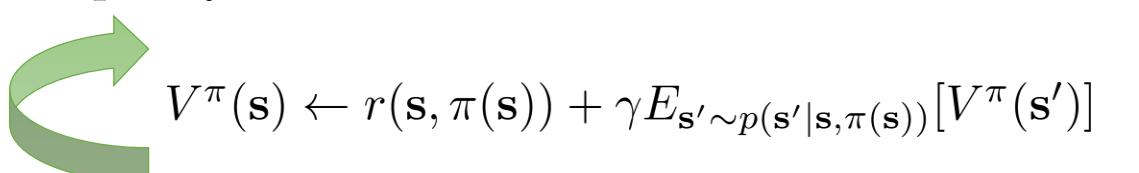
Policy iteration with dynamic programming

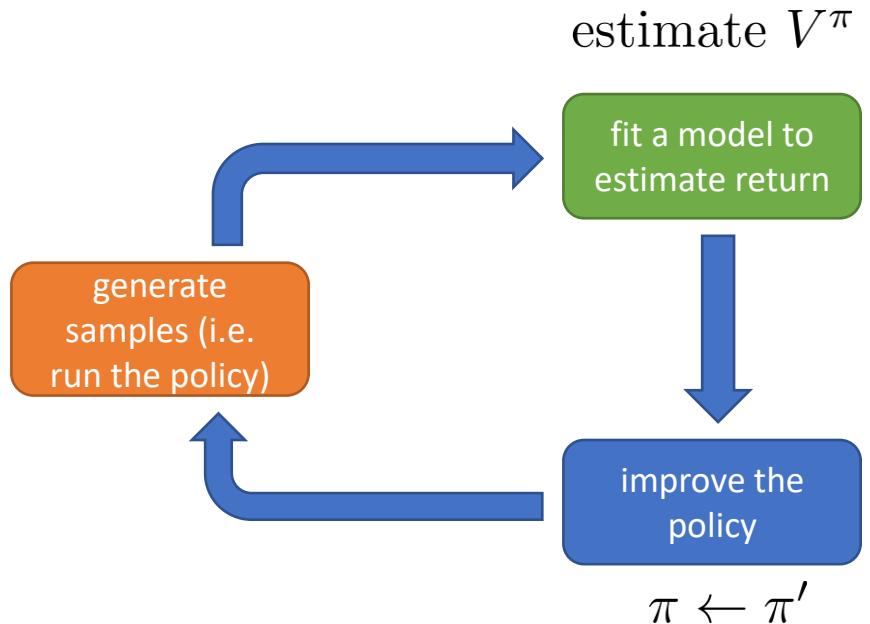
policy iteration:

- 
1. evaluate $V^\pi(\mathbf{s})$
 2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

policy evaluation:


$$V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}' | \mathbf{s}, \pi(\mathbf{s}))}[V^\pi(\mathbf{s}')]$$



0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

16 states, 4 actions per state

can store full $V^\pi(\mathbf{s})$ in a table!

\mathcal{T} is $16 \times 16 \times 4$ tensor

Even simpler dynamic programming

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

$$A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')]$$

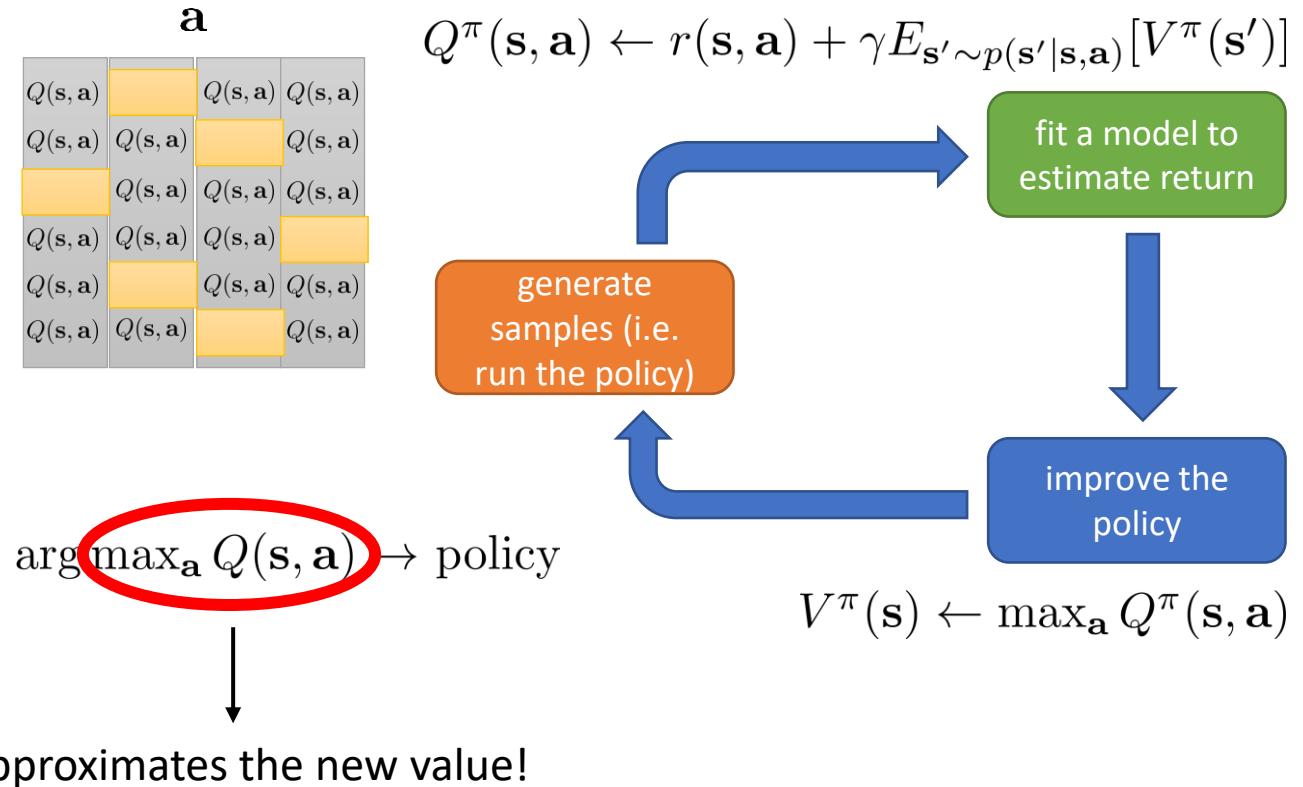
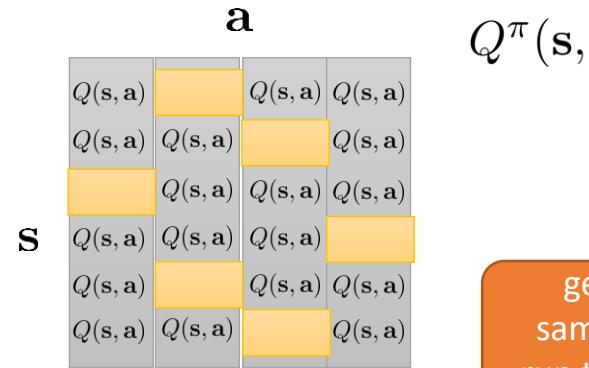
$$\arg \max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) = \arg \max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')]$$
 (a bit simpler)

skip the policy and compute values directly!

value iteration algorithm:

- 1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
- 2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

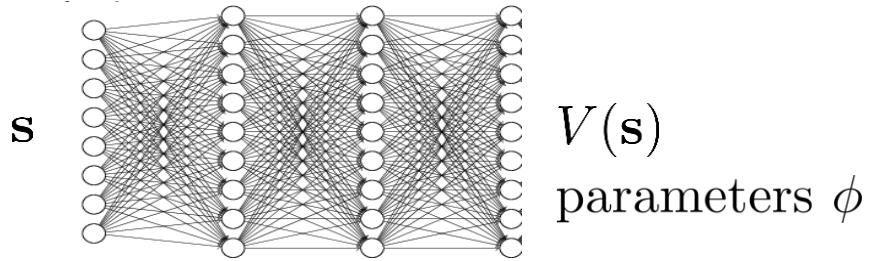


Fitted Value Iteration & Q-Iteration

Fitted value iteration

how do we represent $V(\mathbf{s})$?

big table, one entry for each discrete \mathbf{s}
neural net function $V : \mathcal{S} \rightarrow \mathbb{R}$



$$\mathcal{L}(\phi) = \frac{1}{2} \left\| V_\phi(\mathbf{s}) - \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \right\|^2$$

fitted value iteration algorithm:

- 1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
- 2. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$

$$\mathbf{s} = 0 : V(\mathbf{s}) = 0.2$$

$$\mathbf{s} = 1 : V(\mathbf{s}) = 0.3$$

$$\mathbf{s} = 2 : V(\mathbf{s}) = 0.5$$

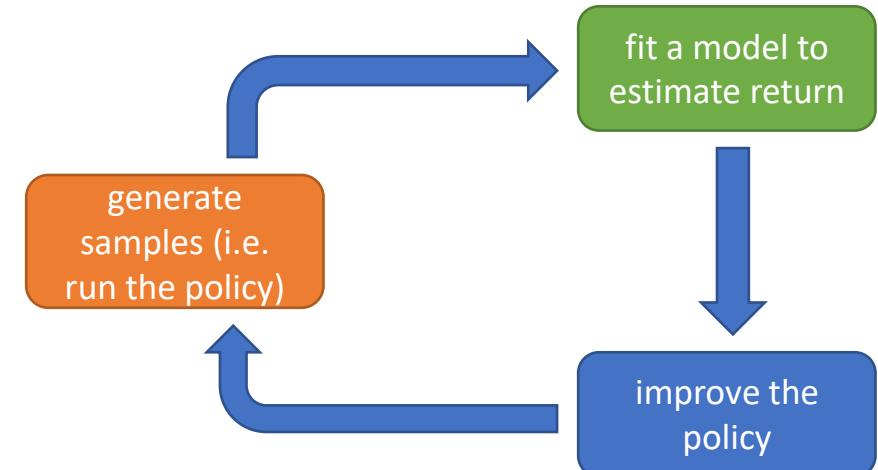


$$|\mathcal{S}| = (255^3)^{200 \times 200}$$

(more than atoms in the universe)

curse of
dimensionality

$$Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}' | \mathbf{s}, \mathbf{a})}[V^\pi(\mathbf{s}')]$$



$$V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$$

What if we don't know the transition dynamics?

fitted value iteration algorithm:

1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
2. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$

need to know outcomes
for different actions!

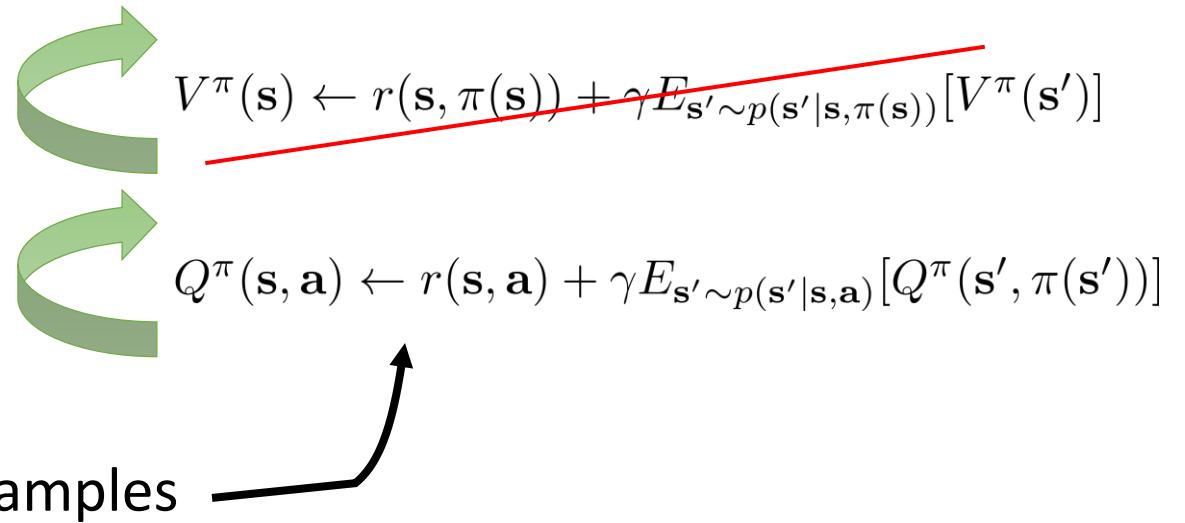
Back to policy iteration...

policy iteration:

1. evaluate $Q^\pi(\mathbf{s}, \mathbf{a})$
2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

policy evaluation:



Can we do the “max” trick again?

policy iteration:

- 
1. evaluate $V^\pi(\mathbf{s})$
 2. set $\pi \leftarrow \pi'$



fitted value iteration algorithm:

- 
1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
 2. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$

forget policy, compute value directly

can we do this with Q-values **also**, without knowing the transitions?

fitted Q iteration algorithm:

- 
1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)]$
 2. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$
- approximate $E[V(\mathbf{s}'_i)] \approx \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$

doesn't require simulation of actions!

+ works even for off-policy samples (unlike actor-critic)

+ only one network, no high-variance policy gradient

- no convergence guarantees for non-linear function approximation (more on this later)

Fitted Q-iteration

full fitted Q-iteration algorithm:

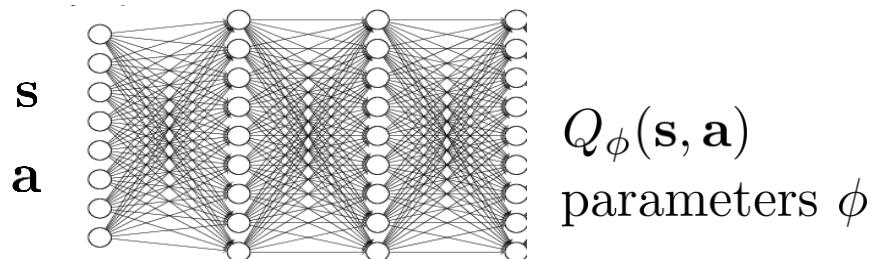
- 1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
- 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
- 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

parameters

dataset size N , collection policy

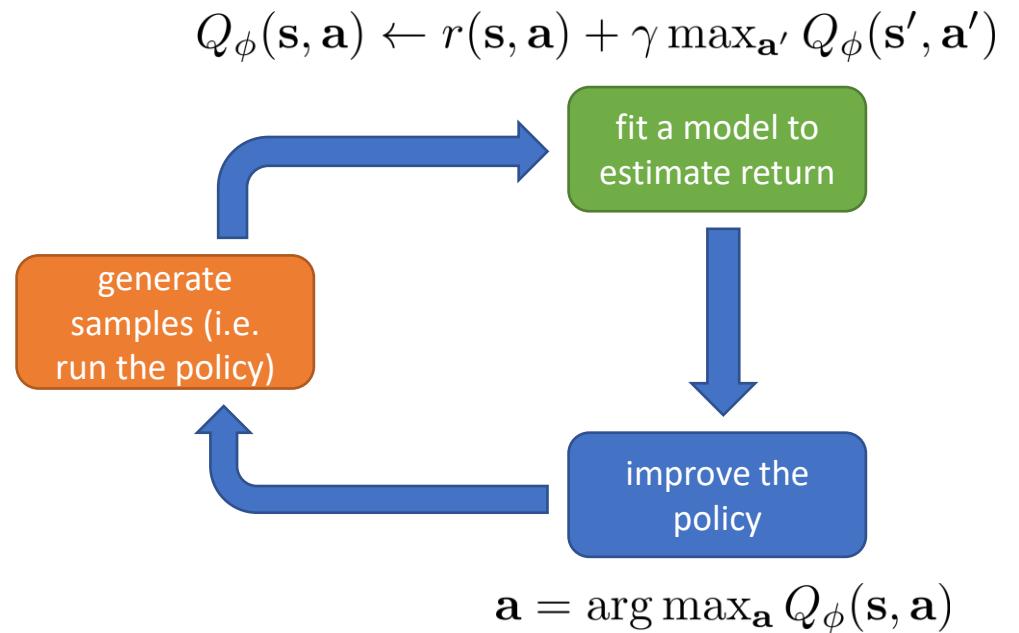
iterations K

gradient steps S



Review

- Value-based methods
 - Don't learn a policy explicitly
 - Just learn value or Q-function
- If we have value function, we have a policy
- Fitted Q-iteration



From Q-Iteration to Q-Learning

Why is this algorithm off-policy?

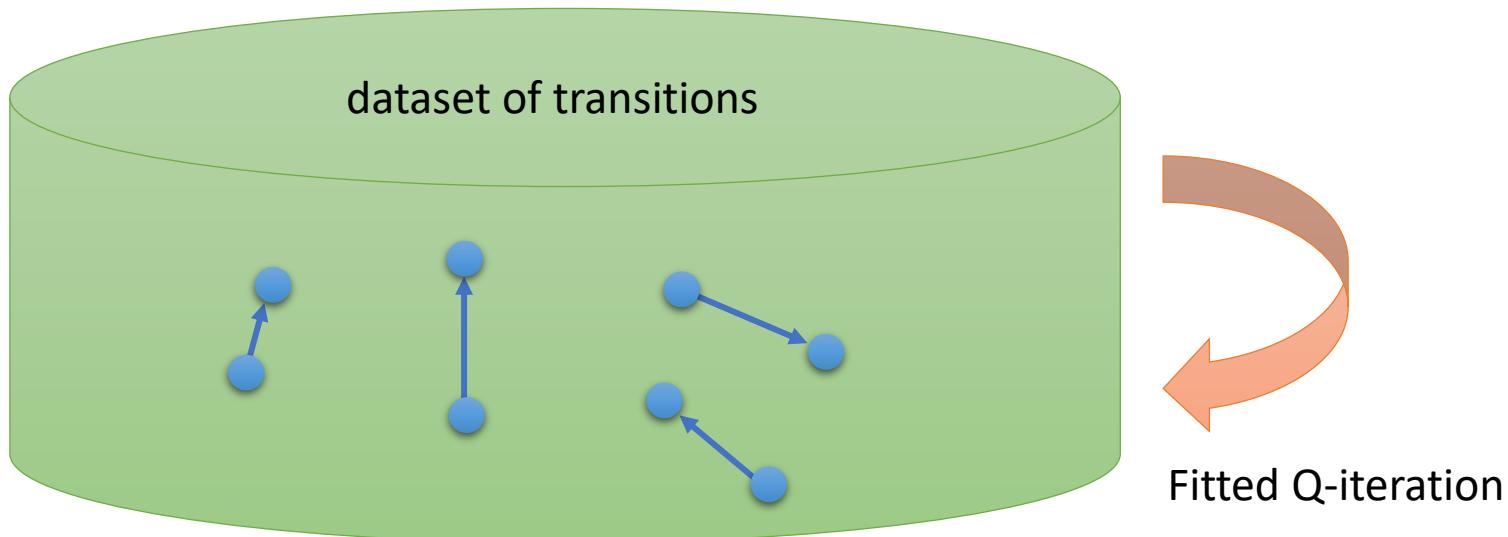
full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

given \mathbf{s} and \mathbf{a} , transition is independent of π

this approximates the value of π' at \mathbf{s}'_i

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$



What is fitted Q-iteration optimizing?

full fitted Q-iteration algorithm:

-
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ ←———— this max improves the policy (tabular case)
 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

$$\begin{array}{c} \uparrow \\ \text{error } \mathcal{E} \end{array}$$

$$\mathcal{E} = \frac{1}{2} E_{(\mathbf{s}, \mathbf{a}) \sim \beta} \left[\left(Q_\phi(\mathbf{s}, \mathbf{a}) - [r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')] \right)^2 \right]$$

if $\mathcal{E} = 0$, then $Q_\phi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$

this is an *optimal* Q-function, corresponding to optimal policy π' :

$$\pi'(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{maximizes reward} \\ \text{sometimes written } Q^* \text{ and } \pi^* \end{array}$$

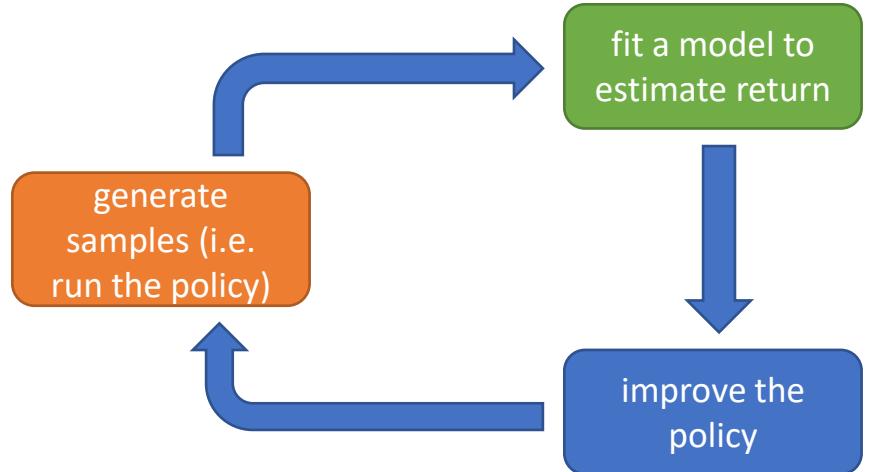
most guarantees are lost when we leave the tabular case (e.g., use neural networks)

Online Q-learning algorithms

full fitted Q-iteration algorithm:

- 1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
- $K \times$ 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
- 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

$$Q_\phi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$$



$$\mathbf{a} = \arg \max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$$

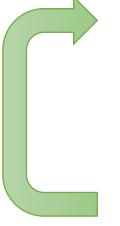
off policy, so many choices here!

online Q iteration algorithm:

- 1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
- 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
- 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

Exploration with Q-learning

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

final policy:

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

why is this a bad idea for step 1?

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 - \epsilon & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ \epsilon / (|\mathcal{A}| - 1) & \text{otherwise} \end{cases}$$

“epsilon-greedy”

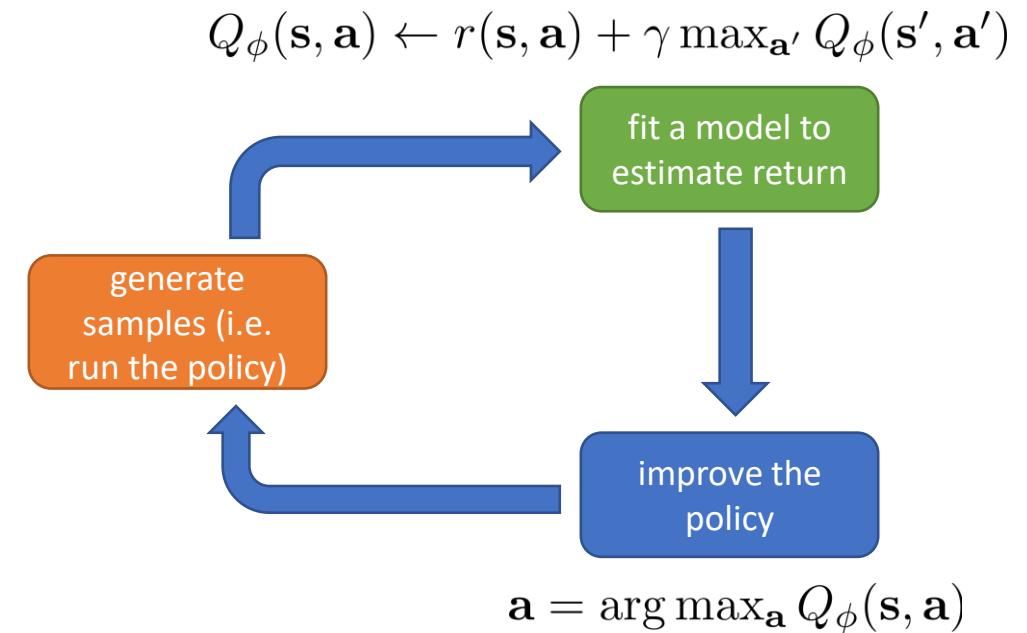
$$\pi(\mathbf{a}_t | \mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$$

“Boltzmann exploration”

We'll discuss exploration in detail in a later lecture!

Review

- Value-based methods
 - Don't learn a policy explicitly
 - Just learn value or Q-function
- If we have value function, we have a policy
- Fitted Q-iteration
 - Batch mode, off-policy method
- Q-learning
 - Online analogue of fitted Q-iteration



Value Functions in Theory

Value function learning theory

value iteration algorithm:

- 
1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
 2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

does it converge?

and if so, to what?

define an operator \mathcal{B} : $\mathcal{B}V = \max_{\mathbf{a}} r_{\mathbf{a}} + \gamma \mathcal{T}_{\mathbf{a}}V$

stacked vector of rewards at all states for action \mathbf{a}

matrix of transitions for action \mathbf{a} such that $\mathcal{T}_{\mathbf{a},i,j} = p(\mathbf{s}' = i | \mathbf{s} = j, \mathbf{a})$

V^* is a *fixed point* of \mathcal{B}

$V^*(\mathbf{s}) = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma E[V^*(\mathbf{s}')]$, so $V^* = \mathcal{B}V^*$

always exists, is always unique, always corresponds to the optimal policy

...but will we reach it?

Value function learning theory

value iteration algorithm:

- 
1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
 2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

V^* is a *fixed point* of \mathcal{B}

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma E[V^*(\mathbf{s}')], \text{ so } V^* = \mathcal{B}V^*$$

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

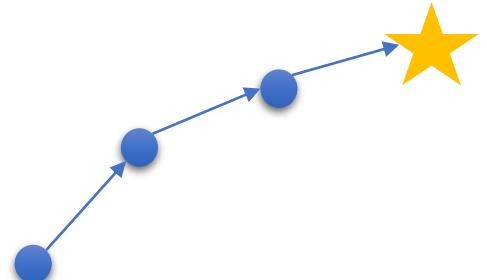
we can prove that value iteration reaches V^* because \mathcal{B} is a *contraction*

contraction: for any V and \bar{V} , we have $\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \underline{\gamma} \|V - \bar{V}\|_\infty$

gap always gets smaller by γ !
(with respect to ∞ -norm)

what if we choose V^* as \bar{V} ? $\mathcal{B}V^* = V^*$!

$$\|\mathcal{B}V - V^*\|_\infty \leq \gamma \|V - V^*\|_\infty$$



Non-tabular value function learning

value iteration algorithm (using \mathcal{B}):

$$\text{→ } 1. V \leftarrow \mathcal{B}V$$

fitted value iteration algorithm (using \mathcal{B} and Π):

$$\text{→ } 1. V \leftarrow \Pi \mathcal{B}V$$

define new operator Π : $\Pi V = \arg \min_{V' \in \Omega} \frac{1}{2} \sum \|V'(\mathbf{s}) - V(\mathbf{s})\|^2$

Π is a *projection* onto Ω (in terms of ℓ_2 norm)

fitted value iteration algorithm:

$$\text{→ } 1. \text{ set } \mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$$
$$2. \text{ set } \phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$$

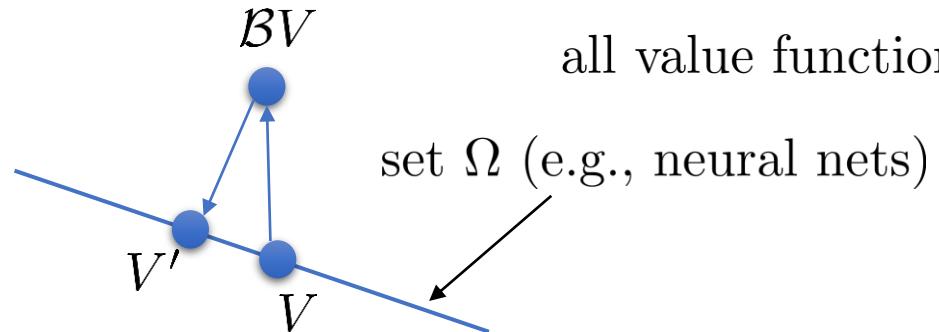
what does this do?

updated value function

$$V' \leftarrow \arg \min_{V' \in \Omega} \frac{1}{2} \sum \|V'(\mathbf{s}) - (\mathcal{B}V)(\mathbf{s})\|^2$$



all value functions represented by, e.g., neural nets

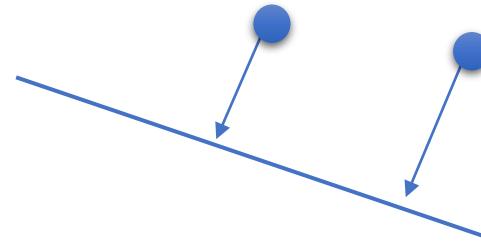


set Ω (e.g., neural nets)

Non-tabular value function learning

fitted value iteration algorithm (using \mathcal{B} and Π):

➡ 1. $V \leftarrow \Pi \mathcal{B} V$



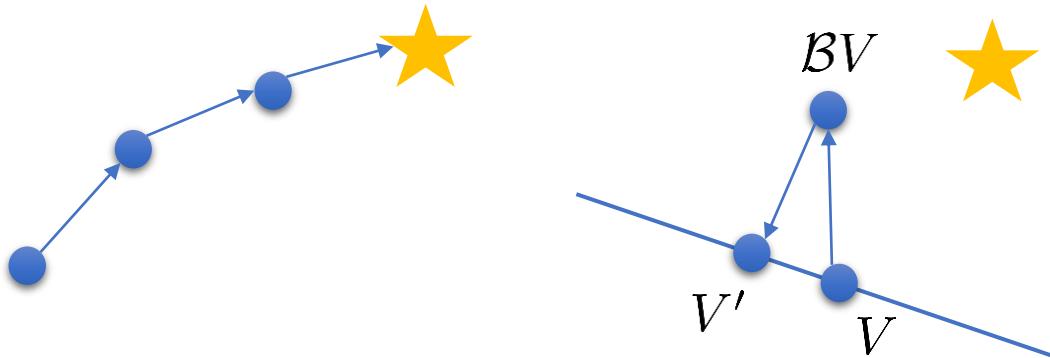
\mathcal{B} is a contraction w.r.t. ∞ -norm (“max” norm)

$$\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \gamma \|V - \bar{V}\|_\infty$$

Π is a contraction w.r.t. ℓ_2 -norm (Euclidean distance)

$$\|\Pi V - \Pi \bar{V}\|^2 \leq \|V - \bar{V}\|^2$$

but... $\Pi \mathcal{B}$ is not a contraction of any kind



Conclusions:
value iteration converges
(tabular case)
fitted value iteration does **not**
converge
not in general
often not in practice

What about fitted Q-iteration?

fitted Q iteration algorithm:

- 
1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)]$
 2. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

define an operator \mathcal{B} : $\mathcal{B}Q = r + \gamma \mathcal{T} \max_{\mathbf{a}} Q$



max now after the transition operator

define an operator Π : $\Pi Q = \arg \min_{Q' \in \Omega} \frac{1}{2} \sum \|Q'(\mathbf{s}, \mathbf{a}) - Q(\mathbf{s}, \mathbf{a})\|^2$

fitted Q-iteration algorithm (using \mathcal{B} and Π):

- 
1. $Q \leftarrow \Pi \mathcal{B}Q$

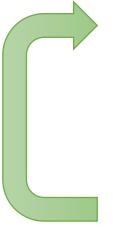
\mathcal{B} is a contraction w.r.t. ∞ -norm (“max” norm)

Π is a contraction w.r.t. ℓ_2 -norm (Euclidean distance)

$\Pi \mathcal{B}$ is not a contraction of any kind **Applies also to online Q-learning**

But... it's just regression!

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

← isn't this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

no gradient through target value

A sad corollary

batch actor-critic algorithm:

- 
1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
 2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
 3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_\phi^\pi(\mathbf{s}'_i) - \hat{V}_\phi^\pi(\mathbf{s}_i)$
 4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

ℓ_∞ contraction \mathcal{B} (but without max)

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

ℓ_2 contraction Π

An aside regarding terminology

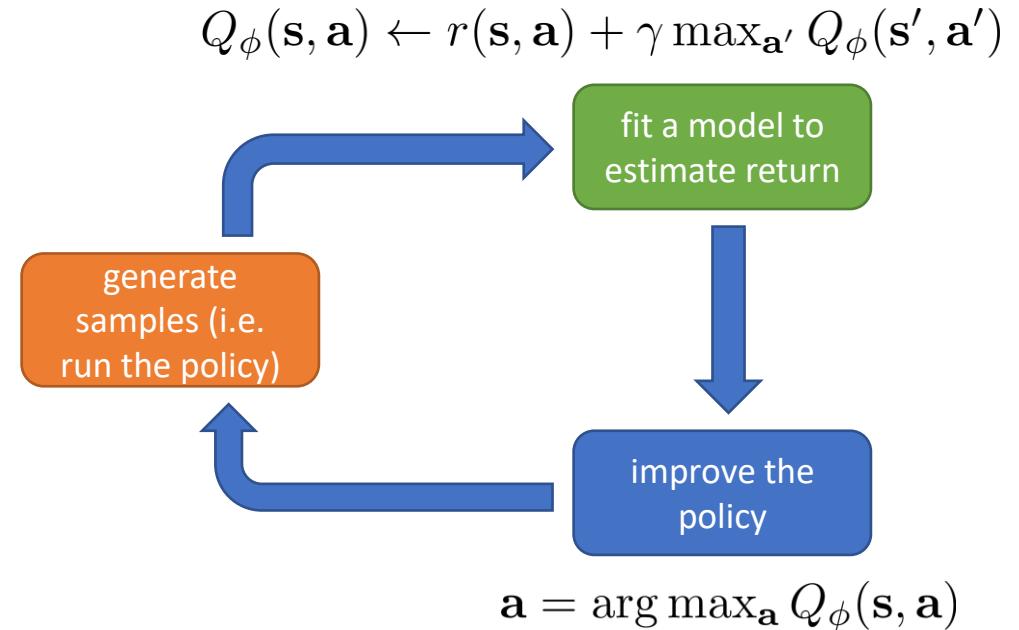
V^π : value function for policy π
this is what the critic does

V^* : value function for optimal policy π^*
this is what value iteration does

fitted bootstrapped policy evaluation doesn't converge!

Review

- Value iteration theory
 - Operator for backup
 - Operator for projection
 - Backup is contraction
 - Value iteration converges
- Convergence with function approximation
 - Projection is also a contraction
 - Projection + backup is **not** a contraction
 - Fitted value iteration does not in general converge
- Implications for Q-learning
 - Q-learning, fitted Q-iteration, etc. does not converge with function approximation
- But we can make it work in practice!
 - Sometimes – tune in next time



Deep RL with Q-Functions

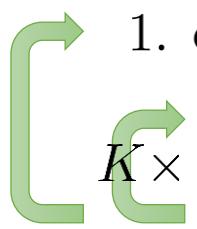
CS 285

Instructor: Sergey Levine
UC Berkeley

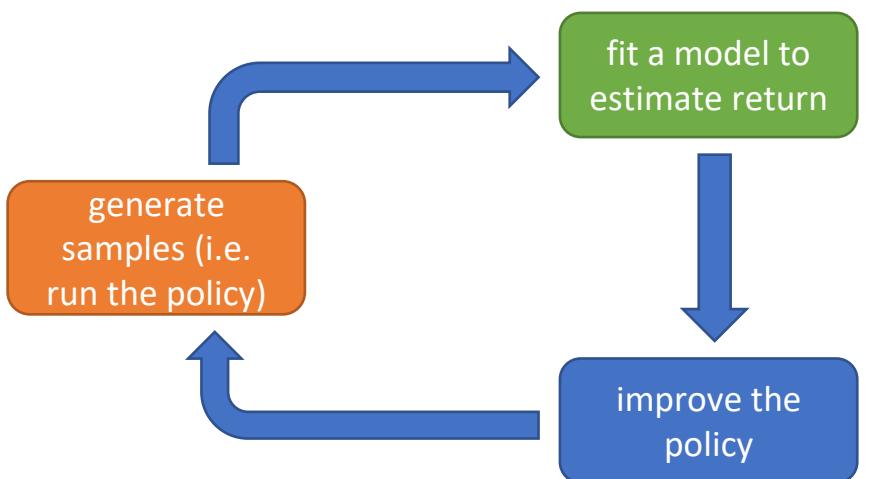


Recap: Q-learning

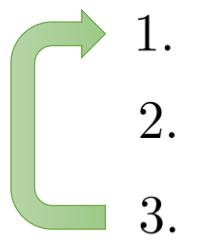
full fitted Q-iteration algorithm:

- 
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

$$Q_\phi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$$



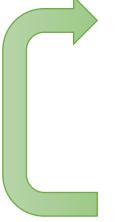
online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

$$\mathbf{a} = \arg \max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$$

What's wrong?

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- these are correlated!
- isn't this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

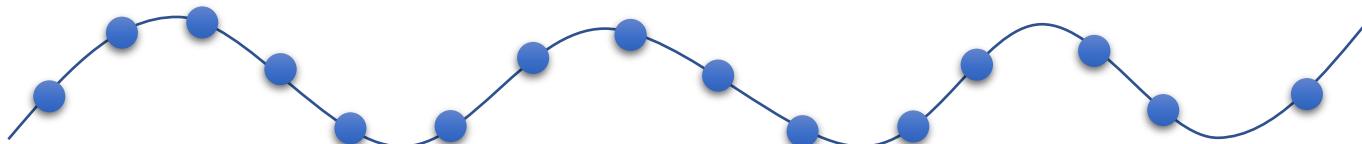
no gradient through target value

Correlated samples in online Q-learning

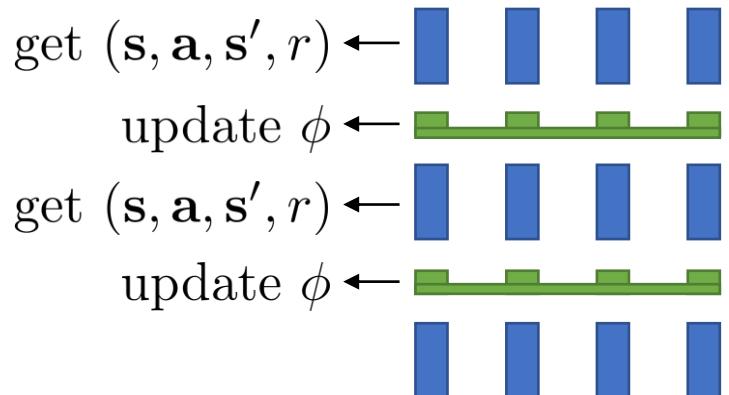
online Q iteration algorithm:

- 1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
- 2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

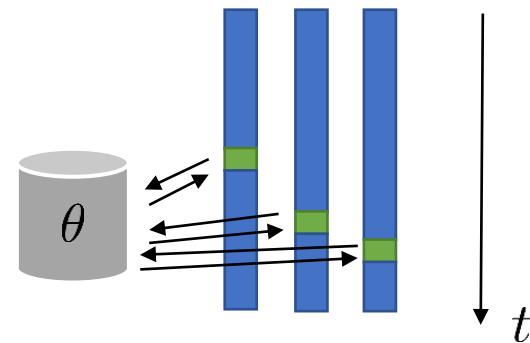
- sequential states are strongly correlated
- target value is always changing



synchronized parallel Q-learning



asynchronous parallel Q-learning



Another solution: replay buffers

online Q iteration algorithm:

- 1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
- 2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

special case with $K = 1$, and one gradient step

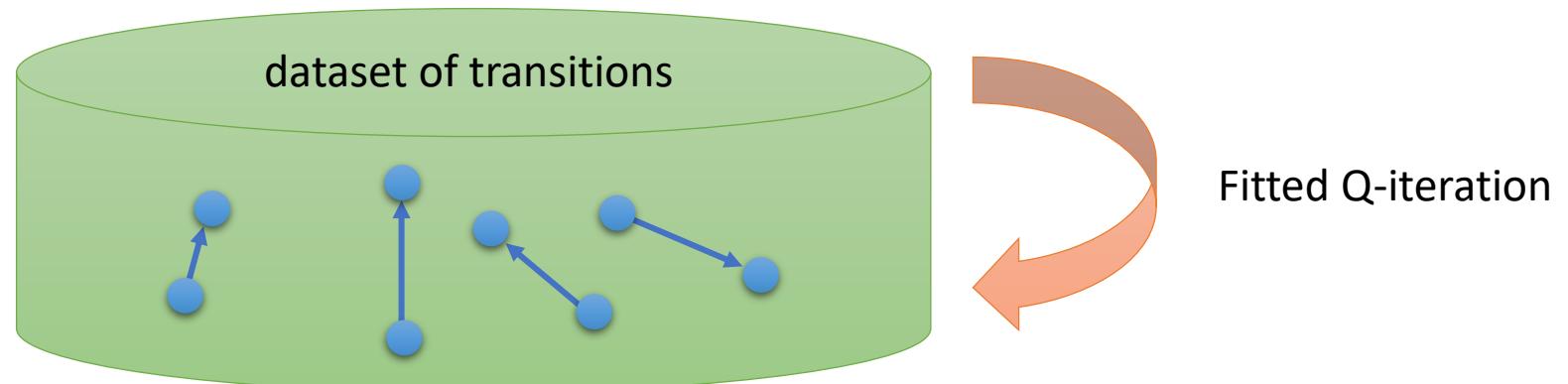
full fitted Q-iteration algorithm:

- 1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
- 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
- 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

any policy will work! (with broad support)

just load data from a buffer here

still use one gradient step



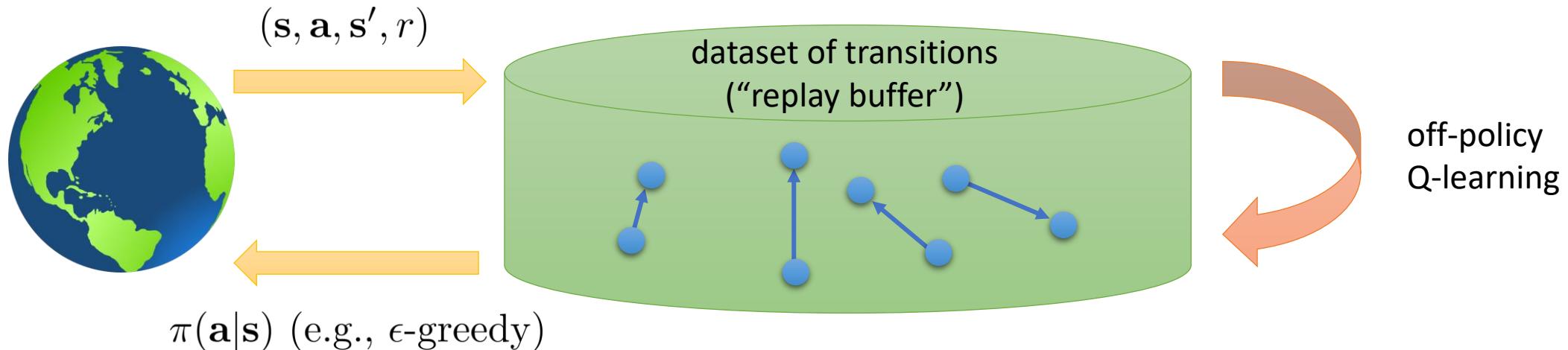
Another solution: replay buffers

Q-learning with a replay buffer:

- 1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B} + samples are no longer correlated
- 2. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$
+ multiple samples in the batch (low-variance gradient)

but where does the data come from?

need to periodically feed the replay buffer...

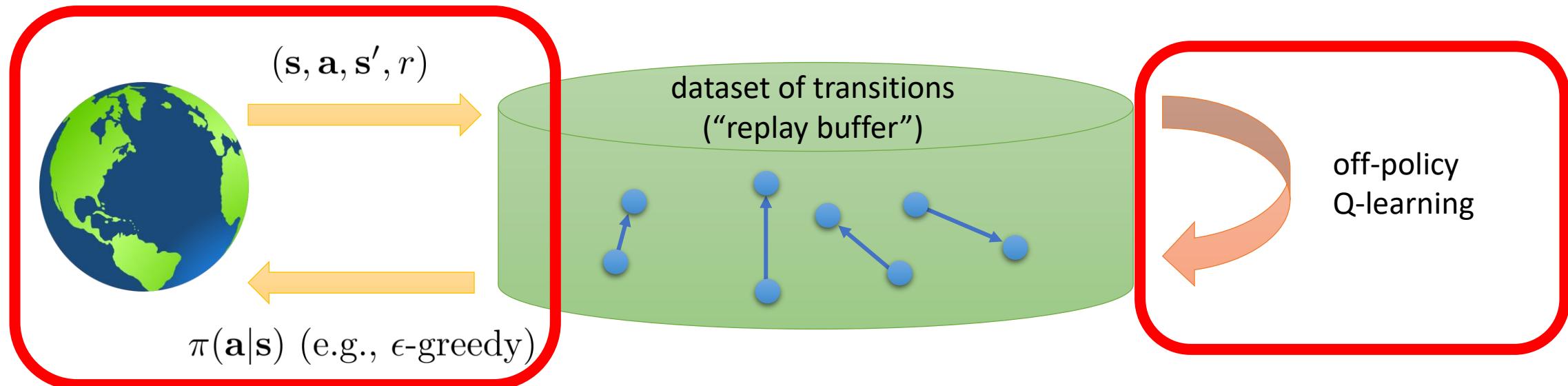


Putting it together

full Q-learning with replay buffer:

- 1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
- 2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
- 3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

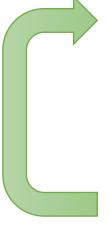
K = 1 is common, though larger K more efficient



Target Networks

What's wrong?

online Q iteration algorithm:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
 2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- use replay buffer**
- these are correlated!**

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

no gradient through target value

This is still a problem!

Q-Learning and Regression

full Q-learning with replay buffer:

-
- 1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 - 2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 - 3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

one gradient step, moving target

full fitted Q-iteration algorithm:

-
- 1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
 - 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 - 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

perfectly well-defined, stable regression

Q-Learning with target networks

supervised regression

Q-learning with replay buffer and target network:

-
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$
- targets don't change in inner loop!**

“Classic” deep Q-learning algorithm (DQN)

Q-learning with replay buffer and target network:

-
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

“classic” deep Q-learning algorithm:

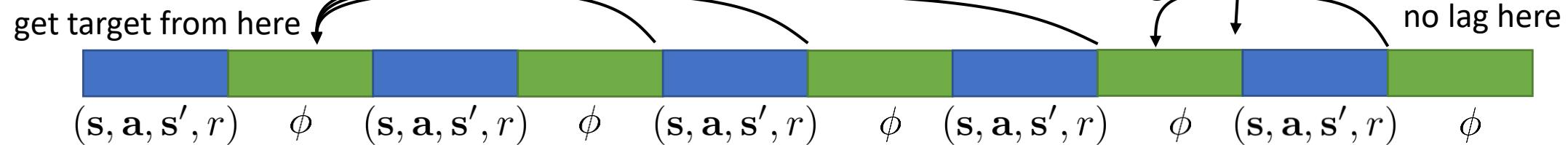
-
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
 2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
 3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
 5. update ϕ' : copy ϕ every N steps

Alternative target network

“classic” deep Q-learning algorithm:

1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target network* $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_{\phi}}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_{\phi}(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ'

Intuition:



Feels weirdly uneven, can we always have the same lag?

Popular alternative (similar to Polyak averaging):

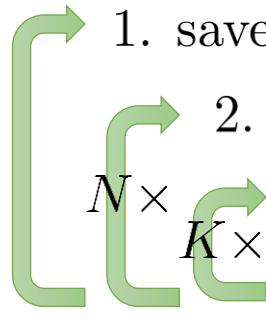
$$5. \text{ update } \phi': \phi' \leftarrow \tau\phi' + (1 - \tau)\phi \qquad \tau = 0.999 \text{ works well}$$

A General View of Q-Learning

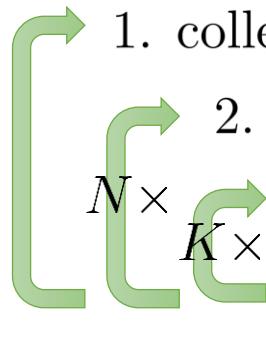
Fitted Q-iteration and Q-learning

Q-learning with replay buffer and target network:

DQN: $N = 1, K = 1$

- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to \mathcal{B}
 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

Fitted Q-learning (written similarly as above):

- 
1. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to \mathcal{B}
 2. save target network parameters: $\phi' \leftarrow \phi$
 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$
- just SGD**

A more general view

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$
2. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to \mathcal{B}
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

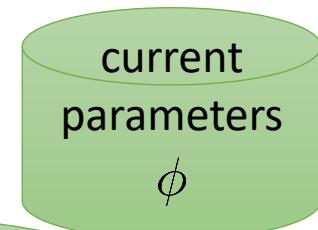
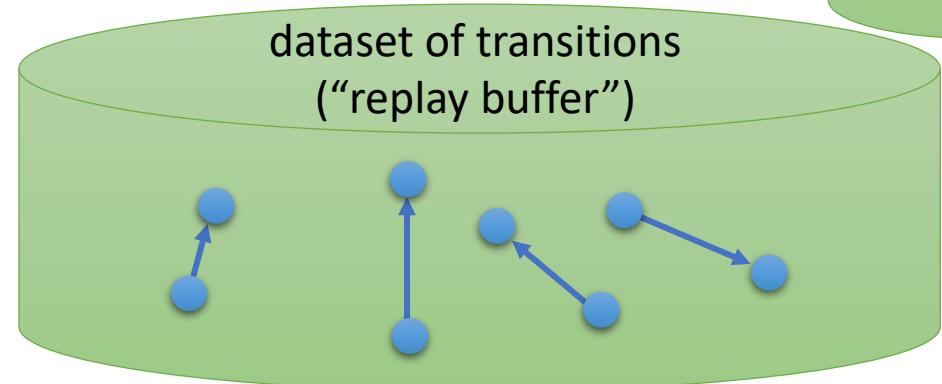
process 1: data collection



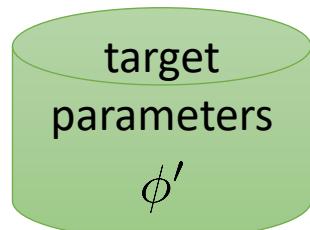
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$



$\pi(\mathbf{a}|\mathbf{s})$ (e.g., ϵ -greedy)



process 2
target update

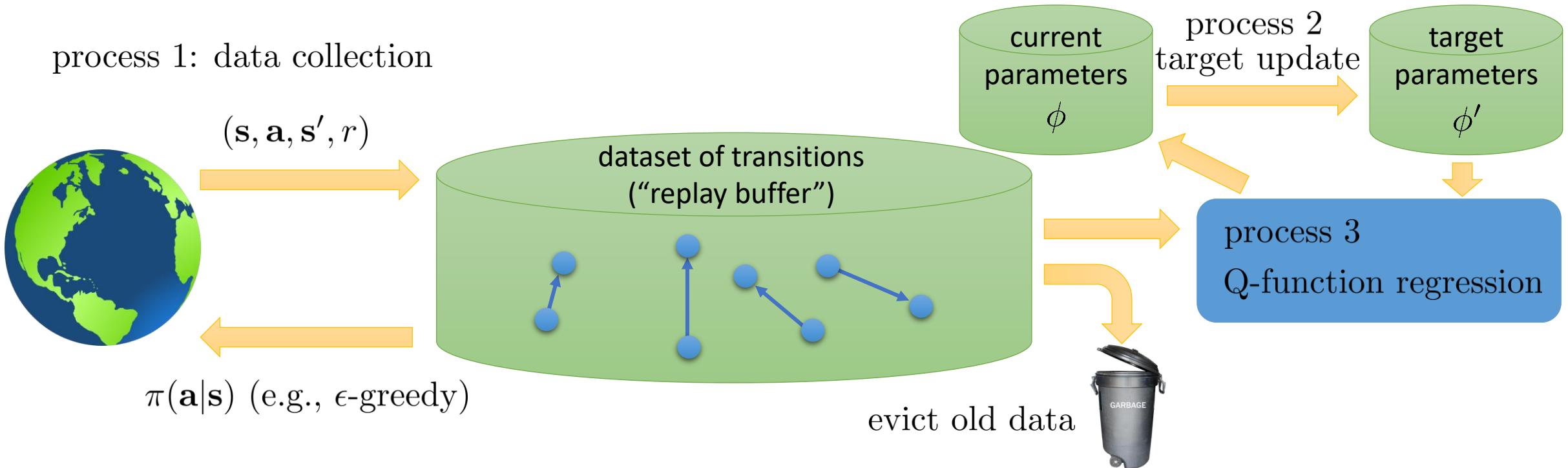


process 3
Q-function regression

evict old data



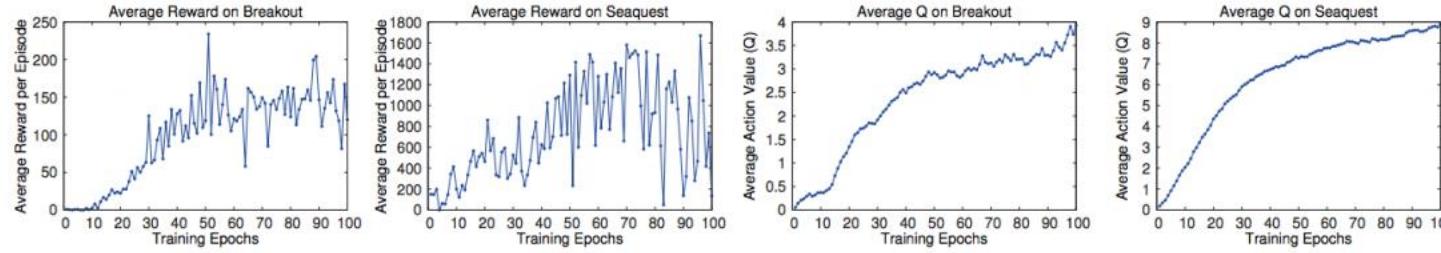
A more general view



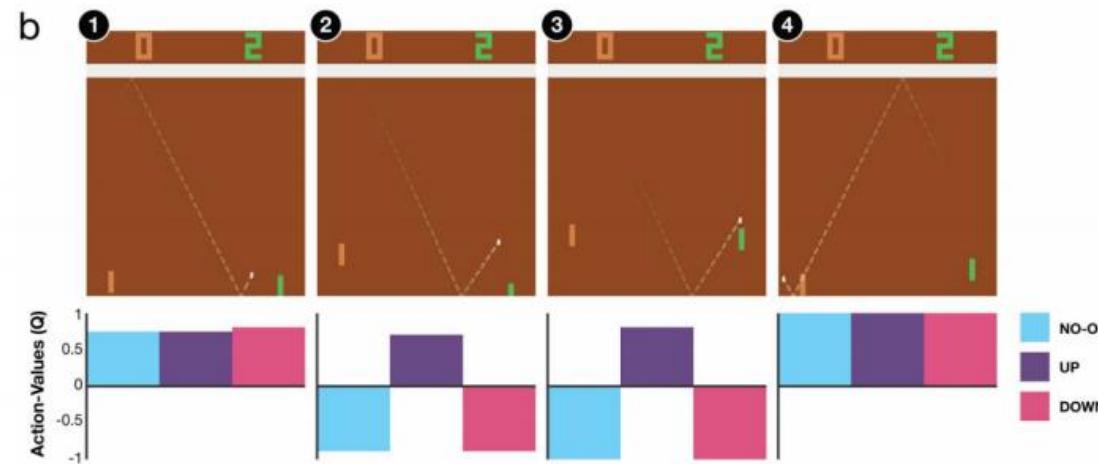
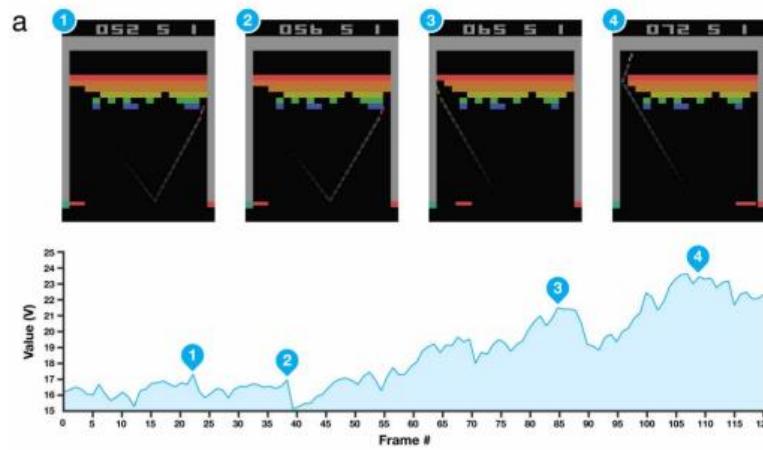
- Online Q-learning (last lecture): evict immediately, process 1, process 2, and process 3 all run at the same speed
- DQN: process 1 and process 3 run at the same speed, process 2 is slow
- Fitted Q-iteration: process 3 in the inner loop of process 2, which is in the inner loop of process 1

Improving Q-Learning

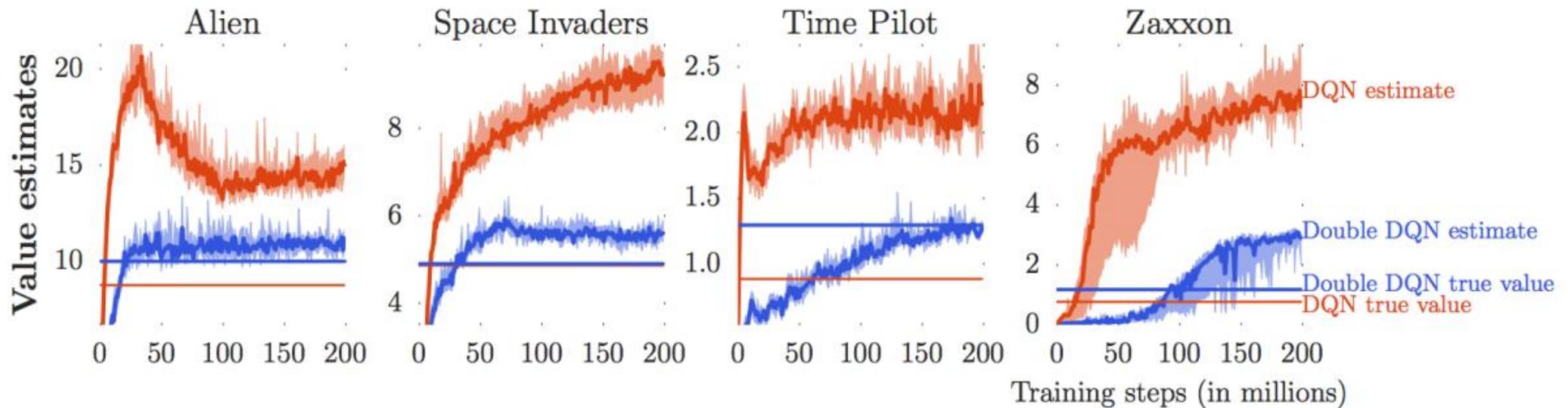
Are the Q-values accurate?



As predicted Q increases, so does the return



Are the Q-values accurate?



Overestimation in Q-learning

$$\text{target value } y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$$

↑
this last term is the problem

imagine we have two random variables: X_1 and X_2

$$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$$

$Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ is not perfect – it looks “noisy”

hence $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ *overestimates* the next value!

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))}$

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

Double Q-learning

$$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$$

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))}$

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

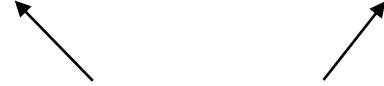

if the noise in these is decorrelated, the problem goes away!

idea: don't use the same network to choose the action and evaluate value!

“double” Q-learning: use two networks:

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}', \mathbf{a}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}', \mathbf{a}'))$$


if the two Q's are noisy in *different* ways, there is no problem

Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

Multi-step returns

$$\text{Q-learning target: } y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$$

these are the only values that matter if $Q_{\phi'}$ is bad! these values are important if $Q_{\phi'}$ is good

where does the signal come from?

remember this?

Actor-critic:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1}) - \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t}) \right)$$

+ lower variance (due to critic)
- not unbiased (if the critic is not perfect)

Policy gradient:

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

+ no bias
- higher variance (because single-sample estimate)

can we construct multi-step targets, like in actor-critic?

$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

N -step return estimator

Q-learning with N-step returns

$$y_{j,t} = \frac{\sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})}{}$$

this is supposed to estimate $Q^\pi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})$ for π

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

we need transitions $\mathbf{s}_{j,t'}, \mathbf{a}_{j,t'}, \mathbf{s}_{j,t'+1}$ to come from π for $t' - t < N - 1$

(not an issue when $N = 1$)

how to fix?

- ignore the problem
 - often works very well
- cut the trace – dynamically choose N to get only on-policy data
 - works well when data mostly on-policy, and action space is small
- importance sampling

+ less biased target values when Q-values are inaccurate
+ typically faster learning, especially early on
- only actually correct when learning on-policy

why?

Q-Learning with Continuous Actions

Q-learning with continuous actions

What's the problem with continuous actions?

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg \max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases}$$

target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

this max
particularly problematic (inner loop of training)

How do we perform the max?

Option 1: optimization

- gradient based optimization (e.g., SGD) a bit slow in the inner loop
- action space typically low-dimensional – what about stochastic optimization?

Q-learning with stochastic optimization

Simple solution:

$$\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \approx \max \{Q(\mathbf{s}, \mathbf{a}_1), \dots, Q(\mathbf{s}, \mathbf{a}_N)\}$$

$(\mathbf{a}_1, \dots, \mathbf{a}_N)$ sampled from some distribution (e.g., uniform)

- + dead simple
- + efficiently parallelizable
- not very accurate

but... do we care? How good does the target need to be anyway?

More accurate solution:

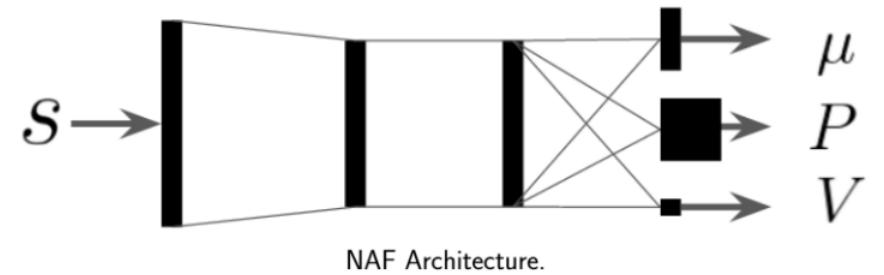
- cross-entropy method (CEM)
 - simple iterative stochastic optimization
- CMA-ES
 - substantially less simple iterative stochastic optimization

works OK, for up to about 40 dimensions

Easily maximizable Q-functions

Option 2: use function class that is easy to optimize

$$Q_\phi(\mathbf{s}, \mathbf{a}) = -\frac{1}{2}(\mathbf{a} - \mu_\phi(\mathbf{s}))^T P_\phi(\mathbf{s})(\mathbf{a} - \mu_\phi(\mathbf{s})) + V_\phi(\mathbf{s})$$



NAF: Normalized Advantage Functions

$$\arg \max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = \mu_\phi(\mathbf{s}) \quad \max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = V_\phi(\mathbf{s})$$

- + no change to algorithm
- + just as efficient as Q-learning
- loses representational power

Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG (Lillicrap et al., ICLR 2016)

“deterministic” actor-critic
(really approximate Q-learning)

$$\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = Q_\phi(\mathbf{s}, \arg \max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}))$$

idea: train another network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \arg \max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

how? just solve $\theta \leftarrow \arg \max_\theta Q_\phi(\mathbf{s}, \mu_\theta(\mathbf{s}))$

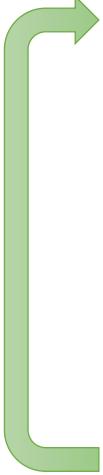
$$\frac{dQ_\phi}{d\theta} = \frac{d\mathbf{a}}{d\theta} \frac{dQ_\phi}{d\mathbf{a}}$$

$$\text{new target } y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_\theta(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j))$$

Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG:

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
 2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
 3. compute $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
 5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mu(\mathbf{s}_j))$
 6. update ϕ' and θ' (e.g., Polyak averaging)

Implementation Tips and Examples

Simple practical tips for Q-learning

- Q-learning takes some care to stabilize
 - Test on easy, reliable tasks first, make sure your implementation is correct

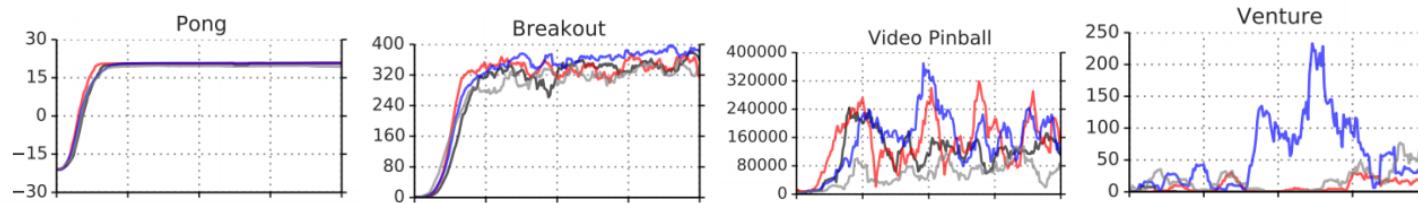


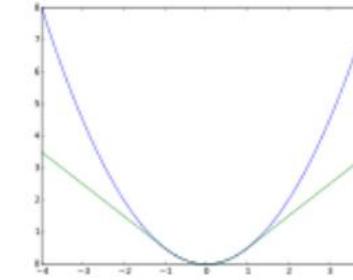
Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. “Prioritized experience replay”. *arXiv preprint arXiv:1511.05952* (2015), Figure 7

- Large replay buffers help improve stability
 - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (ϵ -epsilon) and gradually reduce

Advanced tips for Q-learning

- Bellman error gradients can be big; clip gradients or use Huber loss

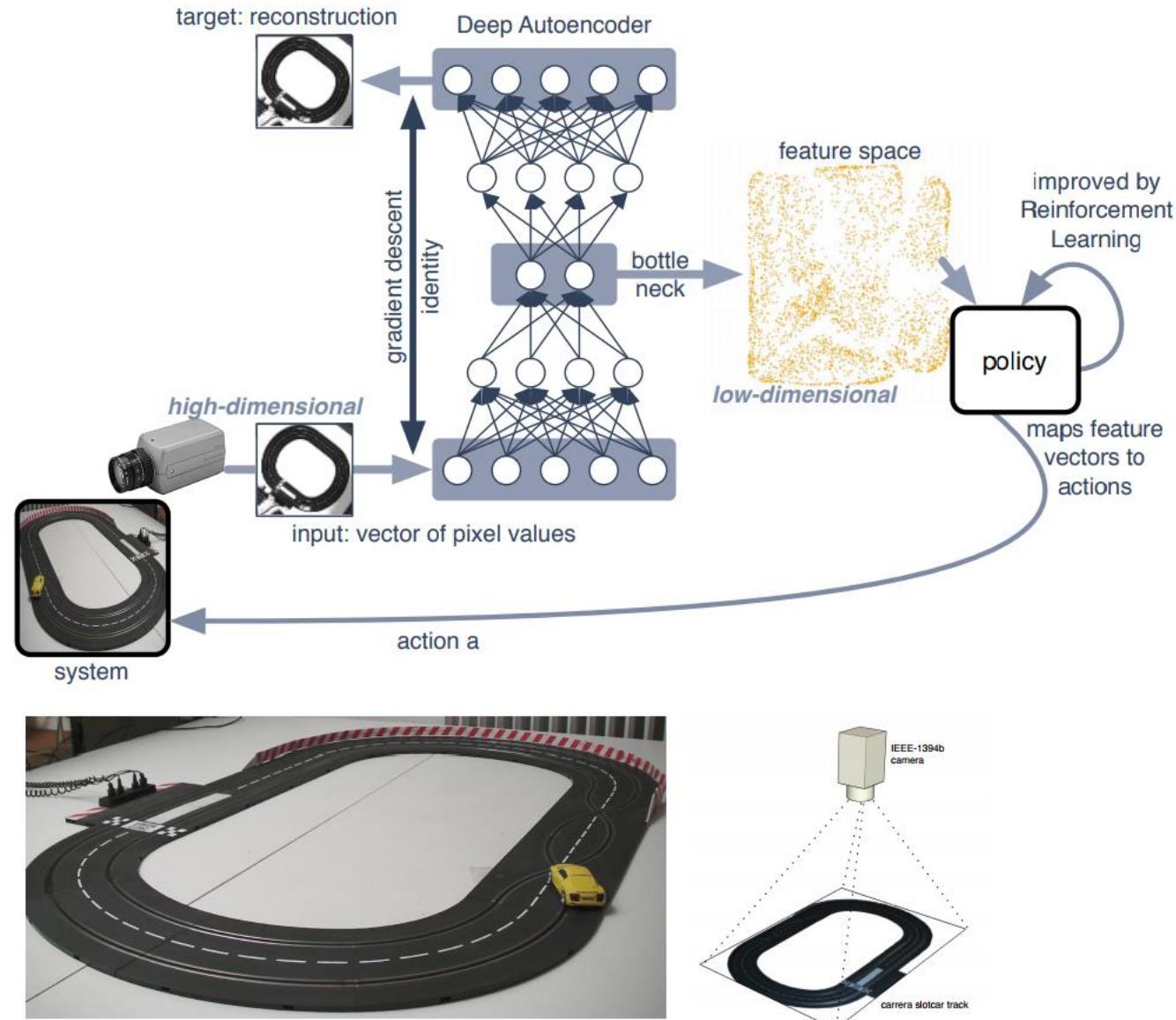
$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- Double Q-learning helps *a lot* in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it's very inconsistent between runs

Fitted Q-iteration in a latent space

- “Autonomous reinforcement learning from raw visual data,” Lange & Riedmiller ’12
- Q-learning on top of latent space learned with autoencoder
- Uses fitted Q-iteration
- Extra random trees for function approximation (but neural net for embedding)



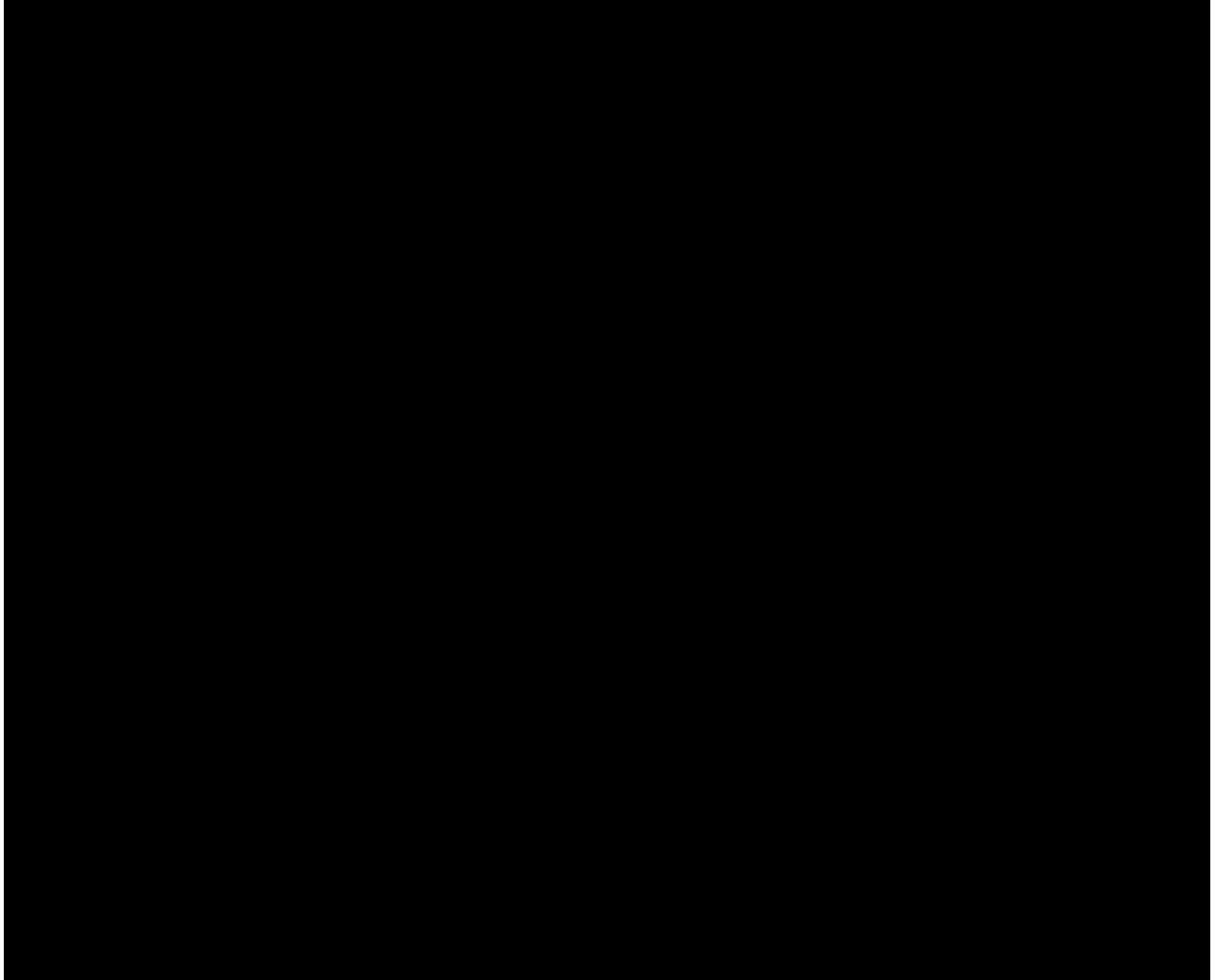
Q-learning with convolutional networks

- “Human-level control through deep reinforcement learning,” Mnih et al. ‘13
- Q-learning with convolutional networks
- Uses replay buffer and target network
- One-step backup
- One gradient step
- Can be improved a lot with double Q-learning (and other tricks)



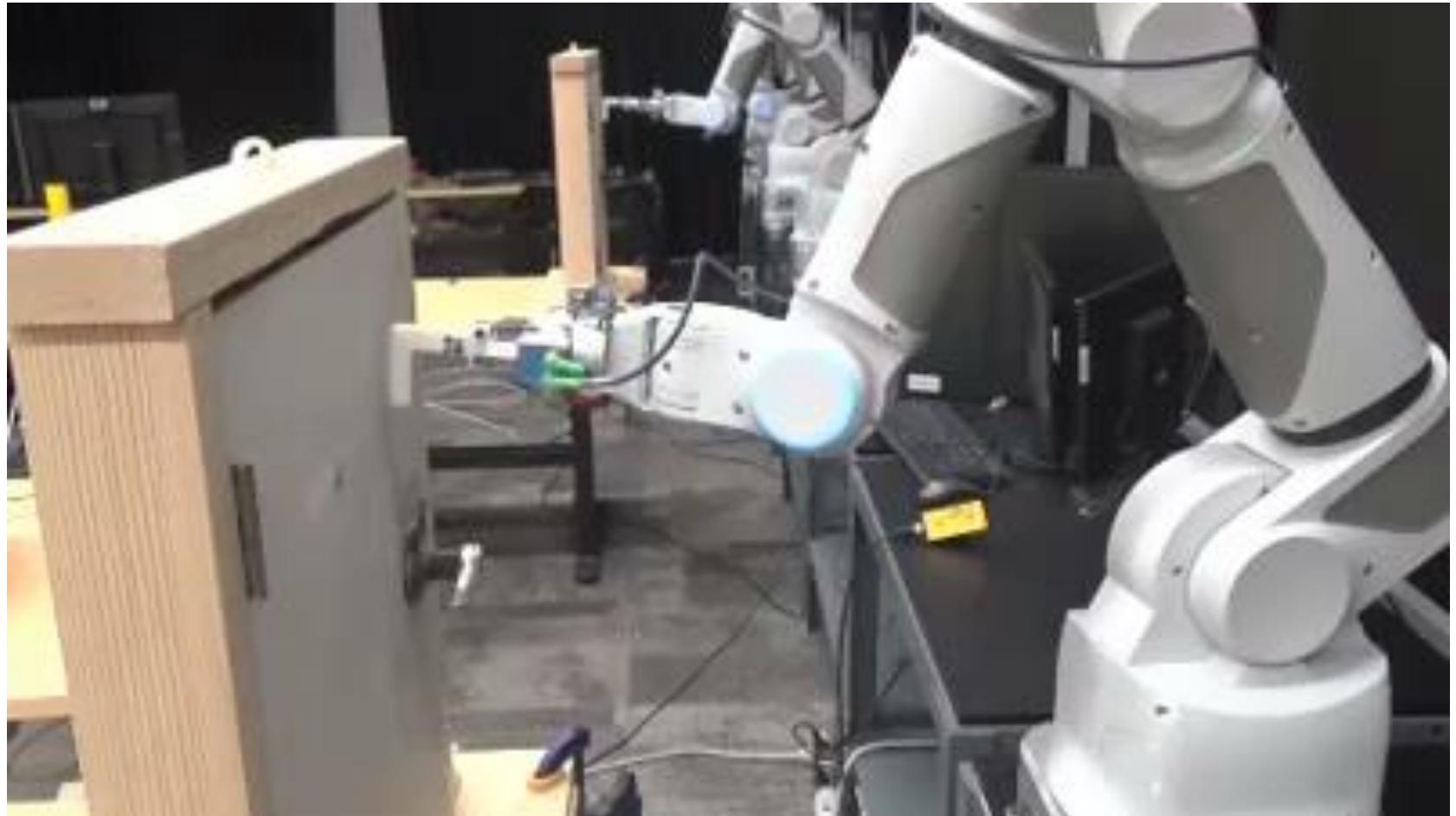
Q-learning with continuous actions

- “Continuous control with deep reinforcement learning,” Lillicrap et al. ‘15
- Continuous actions with maximizer network
- Uses replay buffer and target network (with Polyak averaging)
- One-step backup
- One gradient step per simulator step

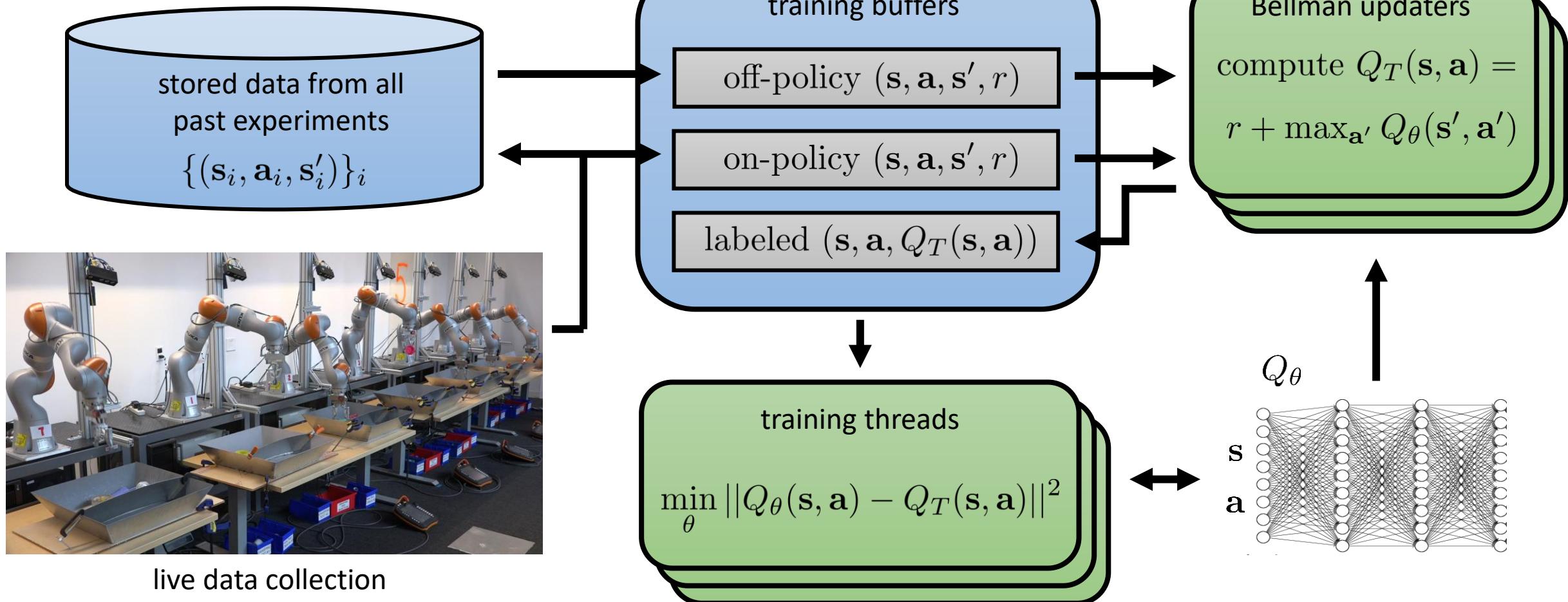


Q-learning on a real robot

- “Robotic manipulation with deep reinforcement learning and ...,” Gu*, Holly*, et al. ‘17
- Continuous actions with NAF (quadratic in actions)
- Uses replay buffer and target network
- One-step backup
- Four gradient steps per simulator step for efficiency
- Parallelized across multiple robots



Large-scale Q-learning with continuous actions (QT-Opt)

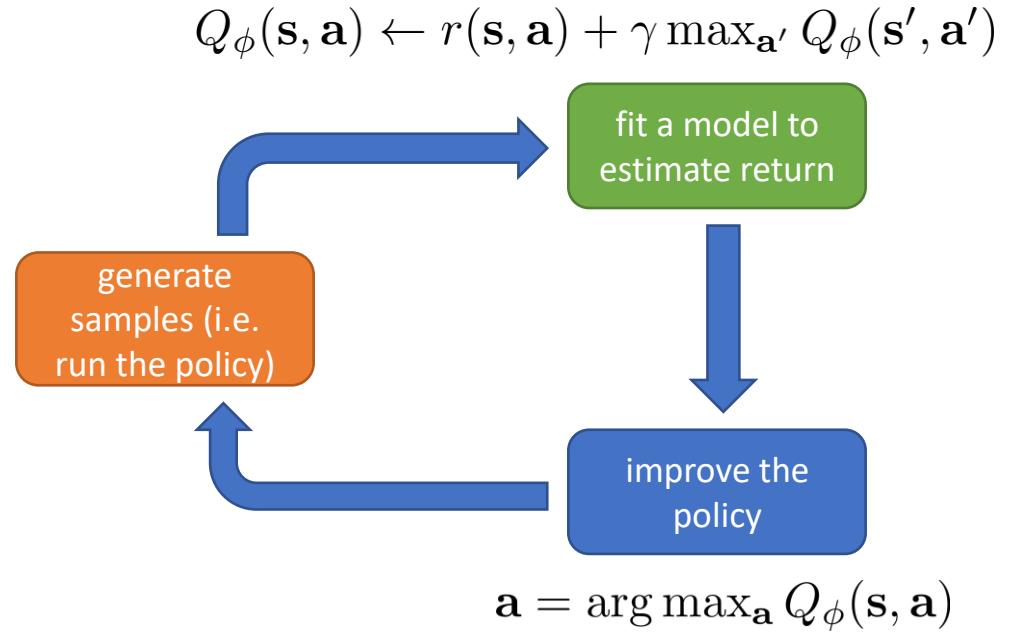


Q-learning suggested readings

- Classic papers
 - Watkins. (1989). Learning from delayed rewards: introduces Q-learning
 - Riedmiller. (2005). Neural fitted Q-iteration: batch-mode Q-learning with neural networks
- Deep reinforcement learning Q-learning papers
 - Lange, Riedmiller. (2010). Deep auto-encoder neural networks in reinforcement learning: early image-based Q-learning method using autoencoders to construct embeddings
 - Mnih et al. (2013). Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
 - Van Hasselt, Guez, Silver. (2015). Deep reinforcement learning with double Q-learning: a very effective trick to improve performance of deep Q-learning.
 - Lillicrap et al. (2016). Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.
 - Gu, Lillicrap, Stuskever, L. (2016). Continuous deep Q-learning with model-based acceleration: continuous Q-learning with action-quadratic value functions.
 - Wang, Schaul, Hessel, van Hasselt, Lanctot, de Freitas (2016). Dueling network architectures for deep reinforcement learning: separates value and advantage estimation in Q-function.

Review

- Q-learning in practice
 - Replay buffers
 - Target networks
- Generalized fitted Q-iteration
- Double Q-learning
- Multi-step Q-learning
- Q-learning with continuous actions
 - Random sampling
 - Analytic optimization
 - Second “actor” network



Advanced Policy Gradients

CS 285

Instructor: Sergey Levine
UC Berkeley



Recap: policy gradients

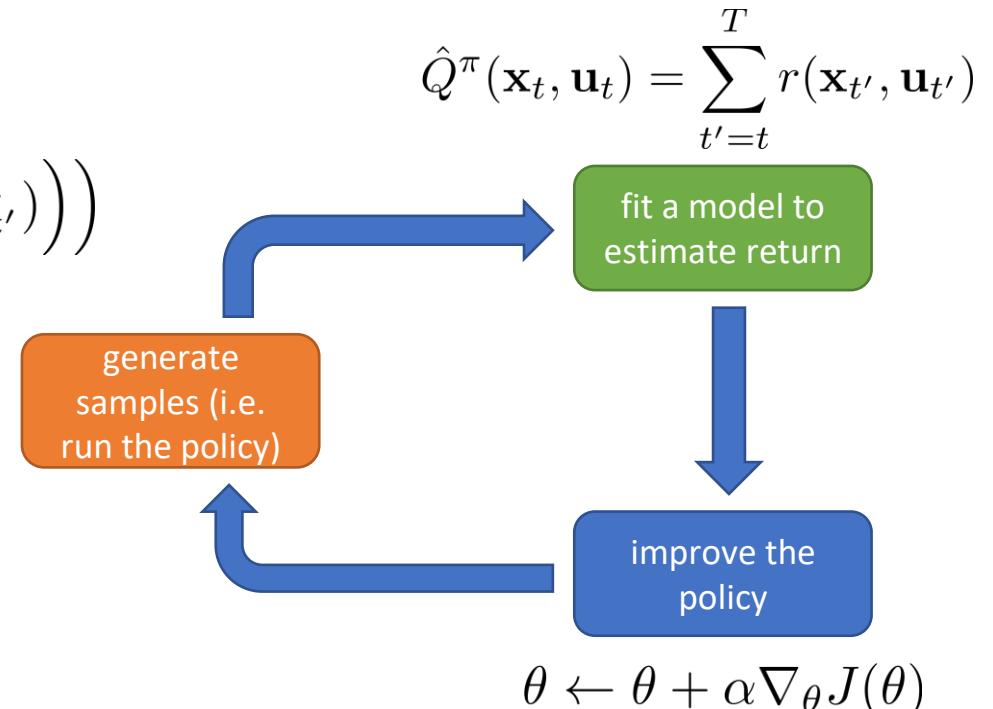
REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i) \right) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

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

“reward to go”

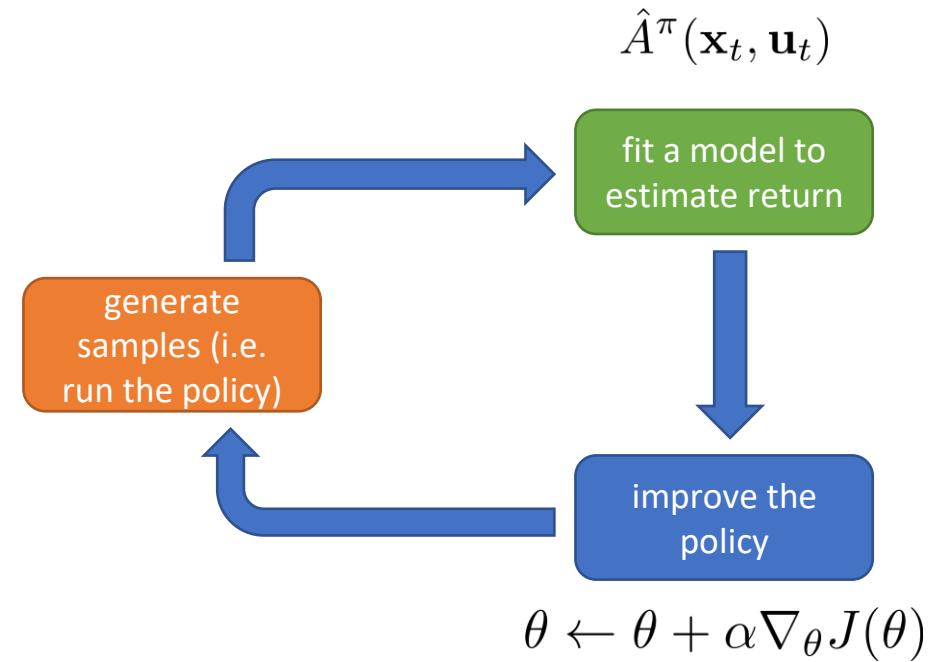
can also use function approximation here



Why does policy gradient work?

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

- C 1. Estimate $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ for current policy π
2. Use $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ to get *improved* policy π'



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

look familiar?

policy iteration algorithm:

- C 1. evaluate $A^{\pi}(\mathbf{s}, \mathbf{a})$
2. set $\pi \leftarrow \pi'$

Policy gradient as policy iteration

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$J(\theta') - J(\theta) = J(\theta') - E_{\mathbf{s}_0 \sim p(\mathbf{s}_0)} [V^{\pi_\theta}(\mathbf{s}_0)]$$

$$= J(\theta') - E_{\tau \sim p_{\theta'}(\tau)} [V^{\pi_\theta}(\mathbf{s}_0)]$$

claim: $J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$

$$= J(\theta') - E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t V^{\pi_\theta}(\mathbf{s}_t) - \sum_{t=1}^{\infty} \gamma^t V^{\pi_\theta}(\mathbf{s}_t) \right]$$

$$= J(\theta') + E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t)) \right]$$

$$= E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] + E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t)) \right]$$

$$= E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t)) \right]$$

$$= E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Policy gradient as policy iteration

$$J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

↑ ↑
 expectation under $\pi_{\theta'}$ advantage under π_θ

$$\begin{aligned} E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] &= \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)} \left[\gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \\ &= \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \\ &\quad \uparrow \\ &\text{is it OK to use } p_\theta(\mathbf{s}_t) \text{ instead?} \end{aligned}$$

importance sampling

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x \sim q(x)} \left[\frac{p(x)}{q(x)}f(x) \right] \end{aligned}$$

Ignoring distribution mismatch?

$$\sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \stackrel{?}{\approx} \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$



why do we want this to be true? $\bar{A}(\theta')$

$$J(\theta') - J(\theta) \approx \bar{A}(\theta') \Rightarrow \theta' \leftarrow \arg \max_{\theta'} \bar{A}(\theta')$$

2. Use $\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t)$ to get *improved* policy π'

is it true? and when?

Claim: $p_\theta(\mathbf{s}_t)$ is *close* to $p_{\theta'}(\mathbf{s}_t)$ when π_θ is *close* to $\pi_{\theta'}$

Bounding the Distribution Change

Ignoring distribution mismatch?

$$\sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \stackrel{?}{\approx} \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$



why do we want this to be true? $\bar{A}(\theta')$

$$J(\theta') - J(\theta) \approx \bar{A}(\theta') \Rightarrow \theta' \leftarrow \arg \max_{\theta'} \bar{A}(\theta)$$

2. Use $\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t)$ to get *improved* policy π'

is it true? and when?

Claim: $p_\theta(\mathbf{s}_t)$ is *close* to $p_{\theta'}(\mathbf{s}_t)$ when π_θ is *close* to $\pi_{\theta'}$

Bounding the distribution change

Claim: $p_\theta(\mathbf{s}_t)$ is *close* to $p_{\theta'}(\mathbf{s}_t)$ when π_θ is *close* to $\pi_{\theta'}$

Simple case: assume π_θ is a *deterministic* policy $\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$

$\pi_{\theta'}$ is *close* to π_θ if $\pi_{\theta'}(\mathbf{a}_t \neq \pi_\theta(\mathbf{s}_t) | \mathbf{s}_t) \leq \epsilon$

$$p_{\theta'}(\mathbf{s}_t) = \underbrace{(1 - \epsilon)^t p_\theta(\mathbf{s}_t)}_{\text{probability we made no mistakes}} + \underbrace{(1 - (1 - \epsilon)^t)) p_{\text{mistake}}(\mathbf{s}_t)}_{\text{some other distribution}}$$

seem familiar?

$$|p_{\theta'}(\mathbf{s}_t) - p_\theta(\mathbf{s}_t)| = (1 - (1 - \epsilon)^t) |p_{\text{mistake}}(\mathbf{s}_t) - p_\theta(\mathbf{s}_t)| \leq 2(1 - (1 - \epsilon)^t)$$

$$\text{useful identity: } (1 - \epsilon)^t \geq 1 - \epsilon t \text{ for } \epsilon \in [0, 1] \quad \leq 2\epsilon t$$

not a great bound, but a bound!

Bounding the distribution change

Claim: $p_\theta(\mathbf{s}_t)$ is *close* to $p_{\theta'}(\mathbf{s}_t)$ when π_θ is *close* to $\pi_{\theta'}$

General case: assume π_θ is an arbitrary distribution

$\pi_{\theta'}$ is *close* to π_θ if $|\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) - \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)| \leq \epsilon$ for all \mathbf{s}_t

Useful lemma: if $|p_X(x) - p_Y(x)| = \epsilon$, exists $p(x, y)$ such that $p(x) = p_X(x)$ and $p(y) = p_Y(y)$ and $p(x = y) = 1 - \epsilon$

$\Rightarrow p_X(x)$ “agrees” with $p_Y(y)$ with probability ϵ

$\Rightarrow \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)$ takes a different action than $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ with probability at most ϵ

$$\begin{aligned} |p_{\theta'}(\mathbf{s}_t) - p_\theta(\mathbf{s}_t)| &= (1 - (1 - \epsilon)^t) |p_{\text{mistake}}(\mathbf{s}_t) - p_\theta(\mathbf{s}_t)| \leq 2(1 - (1 - \epsilon)^t) \\ &\leq 2\epsilon t \end{aligned}$$

Bounding the objective value

$\pi_{\theta'}$ is *close* to π_θ if $|\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) - \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)| \leq \epsilon$ for all \mathbf{s}_t

$$|p_{\theta'}(\mathbf{s}_t) - p_\theta(\mathbf{s}_t)| \leq 2\epsilon t$$

$$\begin{aligned} E_{p_{\theta'}(\mathbf{s}_t)}[f(\mathbf{s}_t)] &= \sum_{\mathbf{s}_t} p_{\theta'}(\mathbf{s}_t) f(\mathbf{s}_t) \geq \sum_{\mathbf{s}_t} p_\theta(\mathbf{s}_t) f(\mathbf{s}_t) - |p_\theta(\mathbf{s}_t) - p_{\theta'}(\mathbf{s}_t)| \max_{\mathbf{s}_t} f(\mathbf{s}_t) \\ &\geq E_{p_\theta(\mathbf{s}_t)}[f(\mathbf{s}_t)] - 2\epsilon t \max_{\mathbf{s}_t} f(\mathbf{s}_t) \end{aligned}$$

$$\begin{aligned} \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] &\geq O(T r_{\max}) \text{ or } O\left(\frac{r_{\max}}{1-\gamma}\right) \\ \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] - \sum_t 2\epsilon t C \end{aligned}$$

maximizing this maximizes a bound on the thing we want!

Where are we at so far?

$$\theta' \leftarrow \arg \max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

such that $|\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) - \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)| \leq \epsilon$

for small enough ϵ , this is guaranteed to improve $J(\theta') - J(\theta)$

Policy Gradients with Constraints

A more convenient bound

Claim: $p_\theta(\mathbf{s}_t)$ is *close* to $p_{\theta'}(\mathbf{s}_t)$ when π_θ is *close* to $\pi_{\theta'}$

$\pi_{\theta'}$ is *close* to π_θ if $|\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) - \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)| \leq \epsilon$ for all \mathbf{s}_t

$$|p_{\theta'}(\mathbf{s}_t) - p_\theta(\mathbf{s}_t)| \leq 2\epsilon t$$

a more convenient bound: $|\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) - \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)| \leq \sqrt{\frac{1}{2}D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\|\pi_\theta(\mathbf{a}_t|\mathbf{s}_t))}$

$\Rightarrow D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\|\pi_\theta(\mathbf{a}_t|\mathbf{s}_t))$ bounds state marginal difference

$$D_{\text{KL}}(p_1(x)\|p_2(x)) = E_{x \sim p_1(x)} \left[\log \frac{p_1(x)}{p_2(x)} \right]$$

KL divergence has some very convenient properties that make it much easier to approximate!

How do we optimize the objective?

$$\theta' \leftarrow \arg \max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

for small enough ϵ , this is guaranteed to improve $J(\theta') - J(\theta)$

How do we enforce the constraint?

$$\theta' \leftarrow \arg \max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

$$\mathcal{L}(\theta', \lambda) = \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] - \lambda(D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) - \epsilon)$$

1. Maximize $\mathcal{L}(\theta', \lambda)$ with respect to θ'  **can do this incompletely (for a few grad steps)**
2. $\lambda \leftarrow \lambda + \alpha(D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) - \epsilon)$

Intuition: raise λ if constraint violated too much, else lower it

an instance of *dual gradient descent* (more on this later!)

Natural Gradient

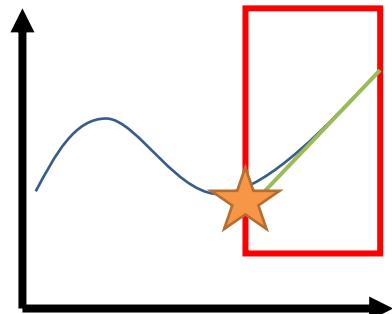
How (else) do we optimize the objective?

$$\theta' \leftarrow \arg \max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$\bar{A}(\theta')$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

for small enough ϵ , this is guaranteed to improve $J(\theta') - J(\theta)$



$$\theta' \leftarrow \arg \max_{\theta'} \nabla_\theta \bar{A}(\theta)^T (\theta' - \theta)$$

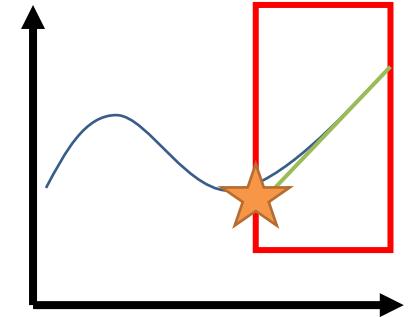
such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

Use first order Taylor approximation for objective (a.k.a., linearization)

How do we optimize the objective?

$$\theta' \leftarrow \arg \max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$



$$\theta' \leftarrow \arg \max_{\theta'} \nabla_\theta \bar{A}(\theta)^T (\theta' - \theta)$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

$$\nabla_{\theta'} \bar{A}(\theta') = \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

(see policy gradient lecture for derivation)

$$\nabla_\theta \bar{A}(\theta) = \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\frac{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \gamma^t \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$\nabla_\theta \bar{A}(\theta) = \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \left[\gamma^t \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] = \nabla_\theta J(\theta)$$

exactly the normal policy gradient!

Can we just use the gradient then?

$$\theta' \leftarrow \arg \max_{\theta'} \nabla_{\theta} J(\theta)^T (\theta' - \theta)$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

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

$$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$$

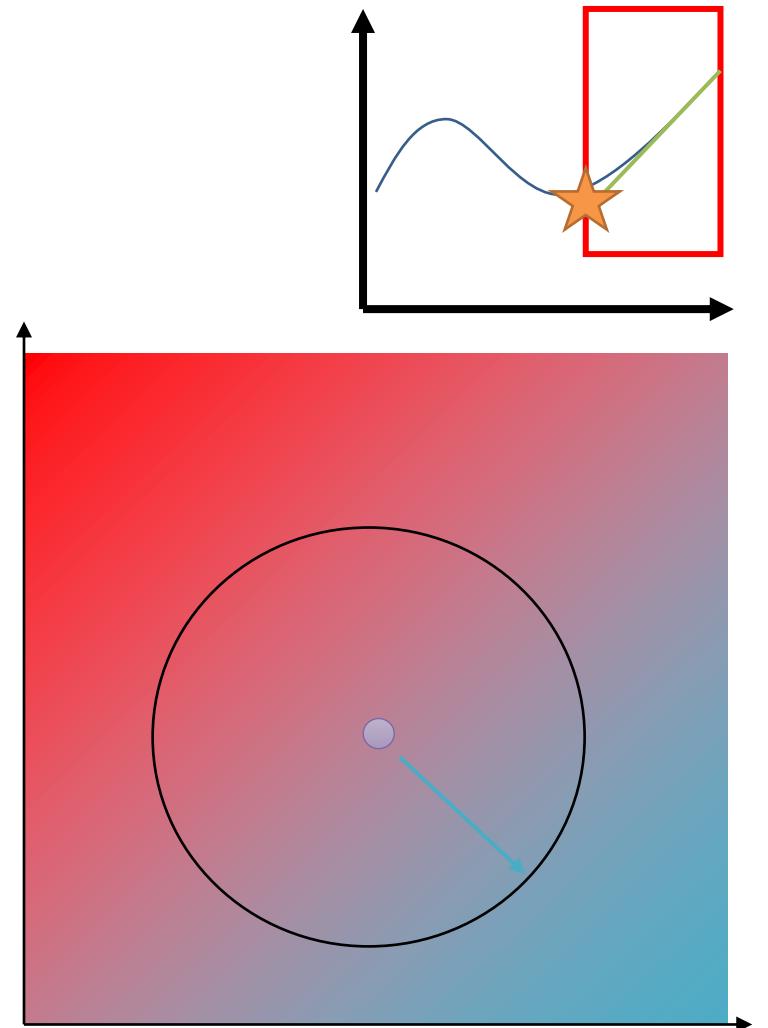
some parameters change probabilities a lot more than others!

Claim: gradient ascent does this:

$$\theta' \leftarrow \arg \max_{\theta'} \nabla_{\theta} J(\theta)^T (\theta' - \theta)$$

such that $\|\theta - \theta'\|^2 \leq \epsilon$

$$\theta' = \theta + \sqrt{\frac{\epsilon}{\|\nabla_{\theta} J(\theta)\|^2}} \nabla_{\theta} J(\theta)$$



Can we just use the gradient then?

$$\theta' \leftarrow \arg \max_{\theta'} \nabla_{\theta} J(\theta)^T (\theta' - \theta)$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

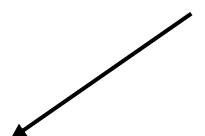


not the same!

$$\theta' \leftarrow \arg \max_{\theta'} \nabla_{\theta} J(\theta)^T (\theta' - \theta)$$

such that $\|\theta - \theta'\|^2 \leq \epsilon$

second order Taylor expansion

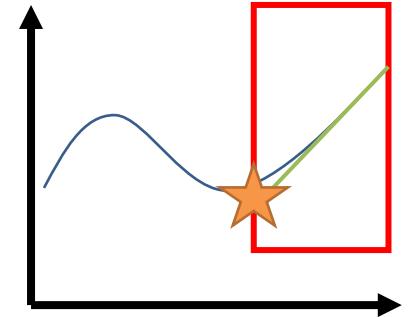


$$D_{\text{KL}}(\pi_{\theta'} \| \pi_{\theta}) \approx \frac{1}{2} (\theta' - \theta)^T \mathbf{F} (\theta' - \theta)$$

Fisher-information matrix

$$\mathbf{F} = E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\mathbf{a} | \mathbf{s}) \nabla_{\theta} \log \pi_{\theta}(\mathbf{a} | \mathbf{s})^T]$$

can estimate with samples



Can we just use the gradient then?

$$\theta' \leftarrow \arg \max_{\theta'} \nabla_{\theta} J(\theta)^T (\theta' - \theta)$$

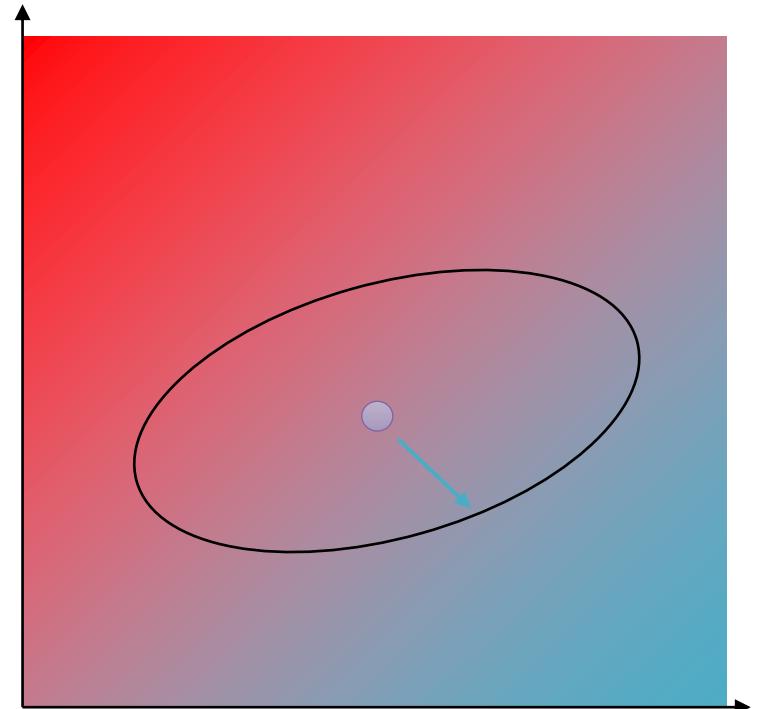
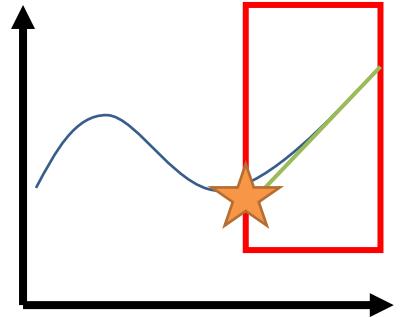
such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \| \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)) \leq \epsilon$

$$D_{\text{KL}}(\pi_{\theta'} \| \pi_{\theta}) \approx \frac{1}{2} (\theta' - \theta)^T \mathbf{F} (\theta' - \theta)$$

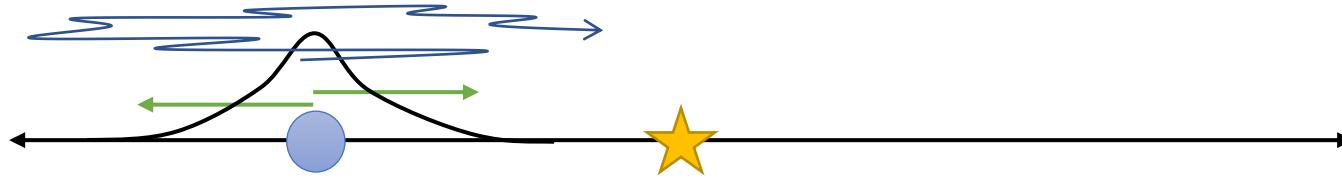
$$\theta' = \theta + \alpha \mathbf{F}^{-1} \nabla_{\theta} J(\theta)$$

natural gradient

$$\alpha = \sqrt{\frac{2\epsilon}{\nabla_{\theta} J(\theta)^T \mathbf{F} \nabla_{\theta} J(\theta)}}$$



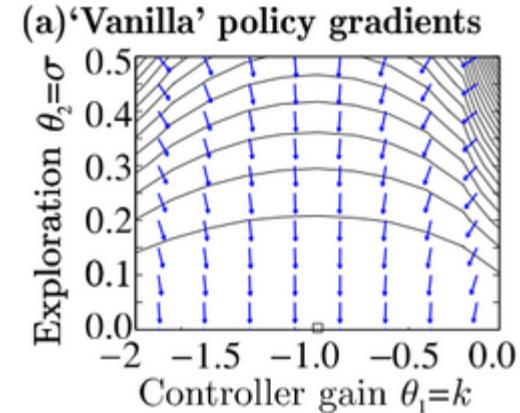
Is this even a problem in practice?



$$r(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^2 - \mathbf{a}_t^2$$

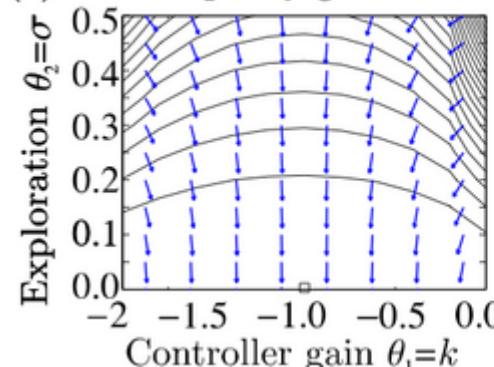
$$\log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2\sigma^2} (k\mathbf{s}_t - \mathbf{a}_t)^2 + \text{const}$$

$$\theta = (k, \sigma)$$

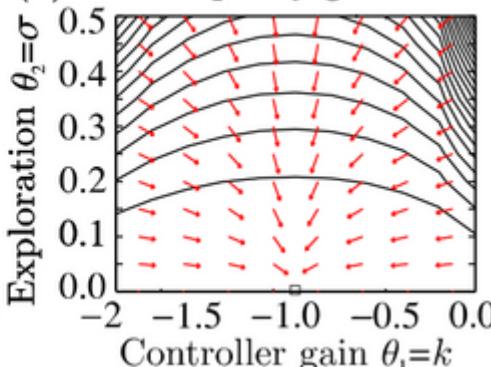


(image from Peters & Schaal 2008)

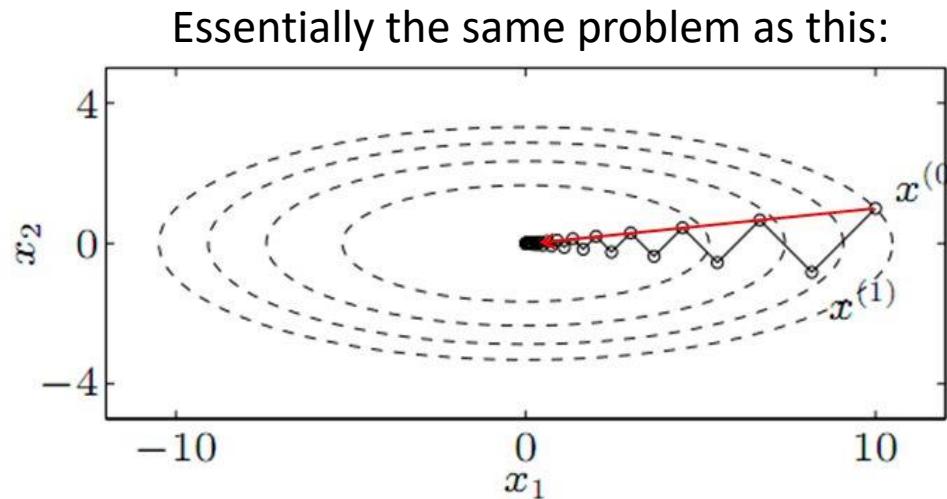
(a) 'Vanilla' policy gradients



(b) Natural policy gradients



(figure from Peters & Schaal 2008)



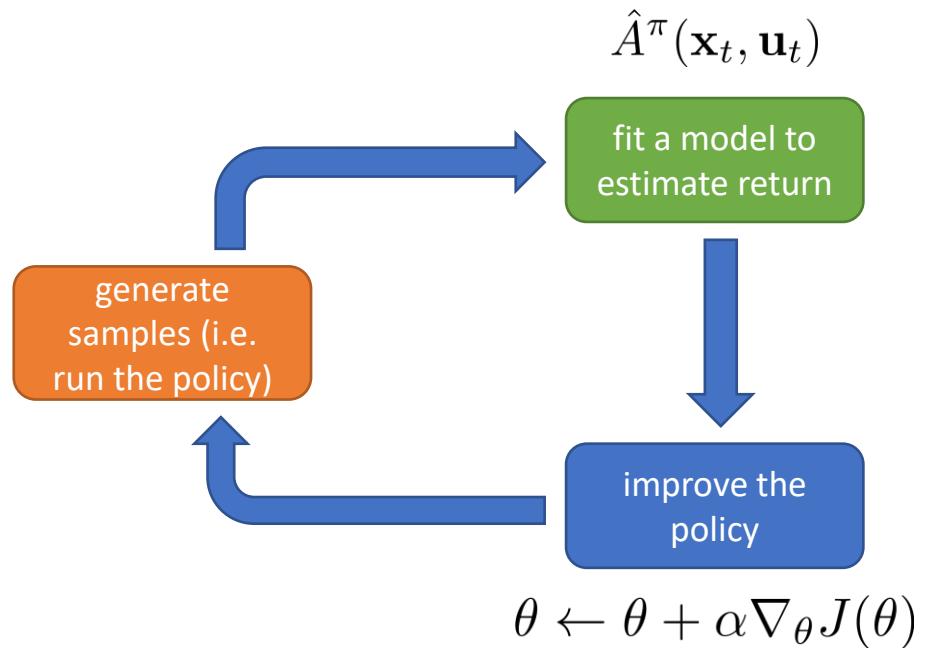
Practical methods and notes

- Natural policy gradient $\theta' = \theta + \alpha \mathbf{F}^{-1} \nabla_{\theta} J(\theta)$
 - Generally a good choice to stabilize policy gradient training
 - See this paper for details:
 - Peters, Schaal. Reinforcement learning of motor skills with policy gradients.
 - Practical implementation: requires efficient Fisher-vector products, a bit non-trivial to do without computing the full matrix
 - See: Schulman et al. Trust region policy optimization
- Trust region policy optimization
- Just use the IS objective directly
 - Use regularization to stay close to old policy
 - See: Proximal policy optimization

$$\alpha = \sqrt{\frac{2\epsilon}{\nabla_{\theta} J(\theta)^T \mathbf{F} \nabla_{\theta} J(\theta)}}$$

Review

- Policy gradient = policy iteration
- Optimize advantage under new policy state distribution
- Using old policy state distribution optimizes a bound, *if* the policies are close enough
- Results in *constrained* optimization problem
- First order approximation to objective = gradient ascent
- Regular gradient ascent has the wrong constraint, use natural gradient
- Practical algorithms
 - Natural policy gradient
 - Trust region policy optimization



Optimal Control and Planning

CS 285

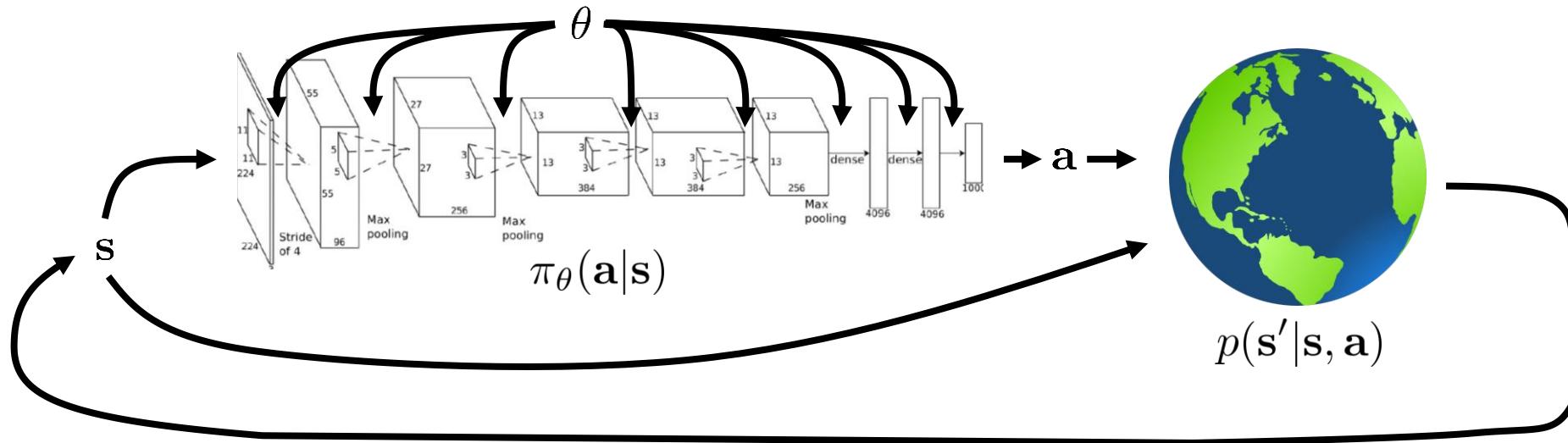
Instructor: Sergey Levine
UC Berkeley



Today's Lecture

1. Introduction to model-based reinforcement learning
 2. What if we know the dynamics? How can we make decisions?
 3. Stochastic optimization methods
 4. Monte Carlo tree search (MCTS)
 5. Trajectory optimization
- Goals:
 - Understand how we can perform planning with known dynamics models in discrete and continuous spaces

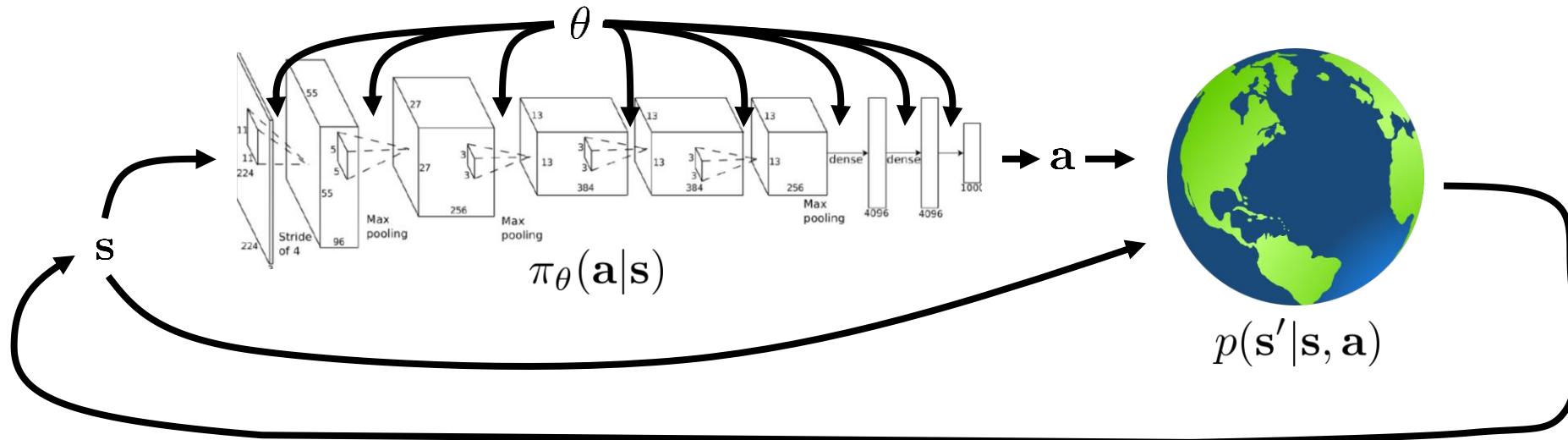
Recap: the reinforcement learning objective



$$p_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \underbrace{\prod_{t=1}^T \pi_\theta(a_t | s_t)}_{\pi_\theta(\tau)} p(s_{t+1} | s_t, a_t)$$

$$\theta^\star = \arg \max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

Recap: model-free reinforcement learning



$$p_\theta(s_1, a_1, \dots, s_T, a_T) = p(s_1) \underbrace{\prod_{t=1}^T \pi_\theta(a_t | s_t)}_{\pi_\theta(\tau)} p(s_{t+1} | s_t, a_t)$$

assume this is unknown
don't even attempt to learn it

$$\theta^\star = \arg \max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

What if we knew the transition dynamics?

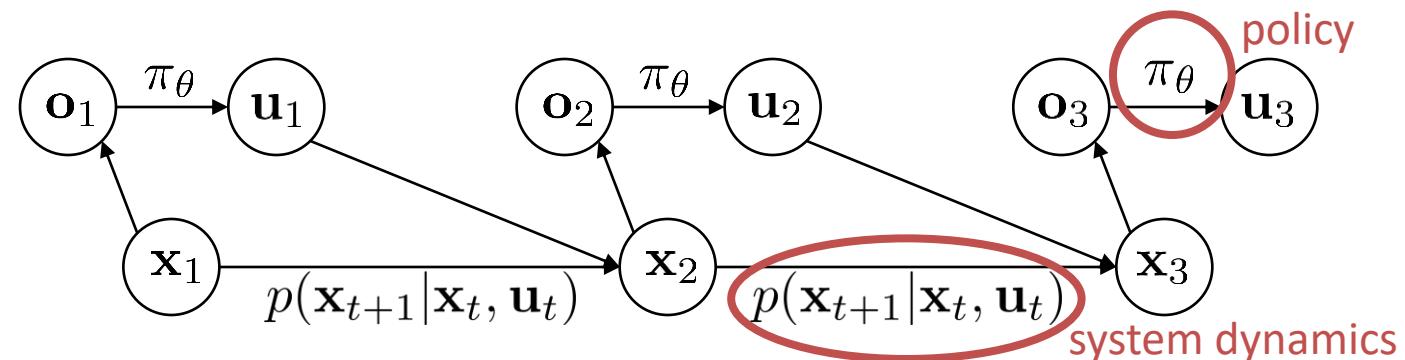
- Often we do know the dynamics
 1. Games (e.g., Atari games, chess, Go)
 2. Easily modeled systems (e.g., navigating a car)
 3. Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
 1. System identification – fit unknown parameters of a known model
 2. Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier?

Often, yes!

Model-based reinforcement learning

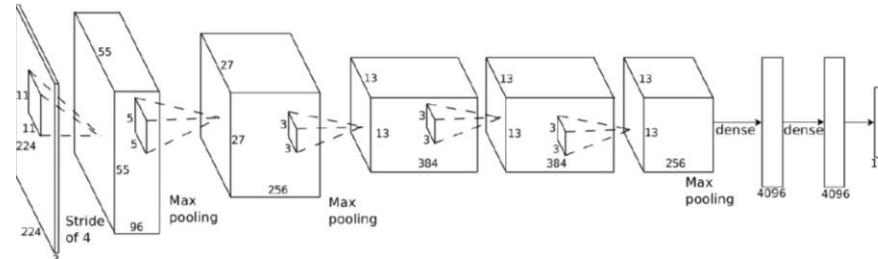
1. Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions
2. Today: how can we make decisions if we *know* the dynamics?
 - a. How can we choose actions under perfect knowledge of the system dynamics?
 - b. Optimal control, trajectory optimization, planning
3. Next week: how can we learn *unknown* dynamics?
4. How can we then also learn policies? (*e.g. by imitating optimal control*)



The objective



s_t



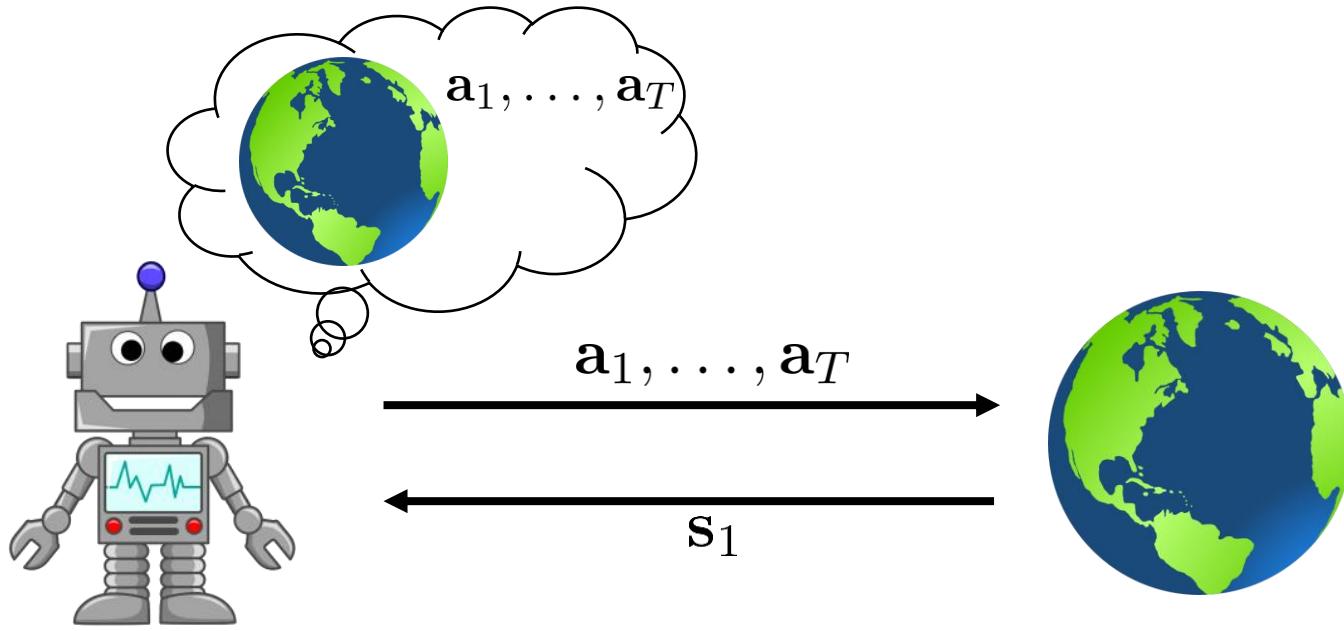
$$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$$



a_t

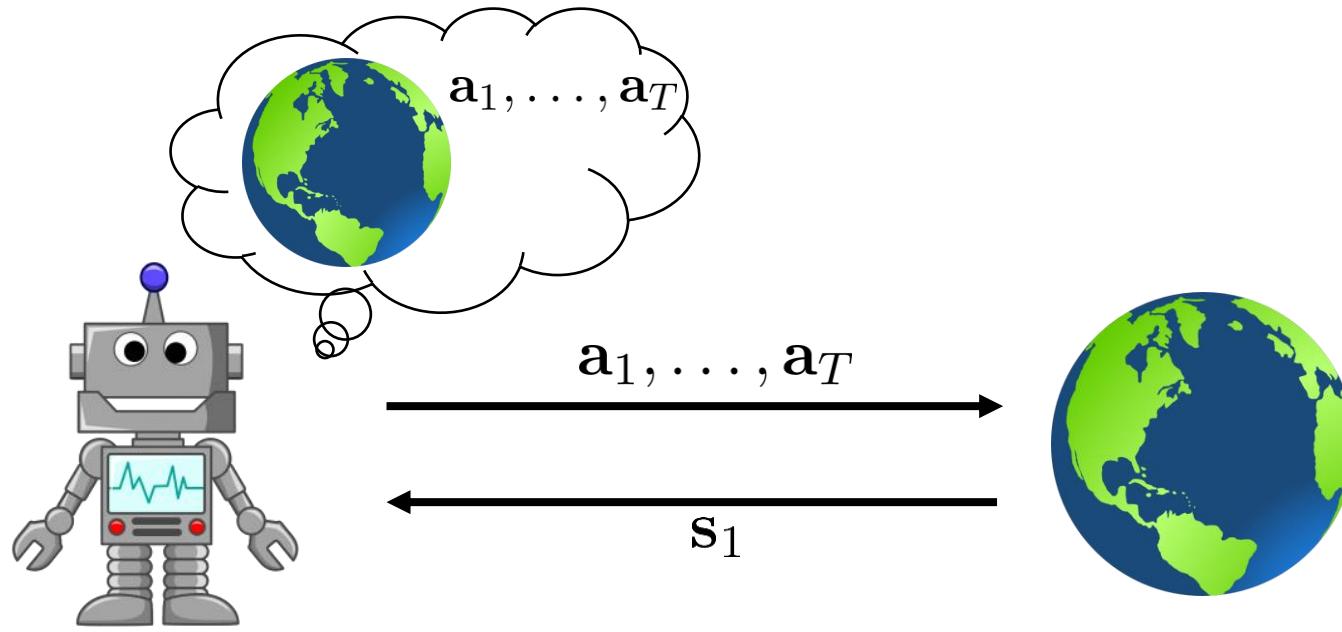
$$\min_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T \log p(\mathbf{s}_t, \mathbf{a}_t) \text{ by tiger+af, } (\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$$

The deterministic case



$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

The stochastic open-loop case



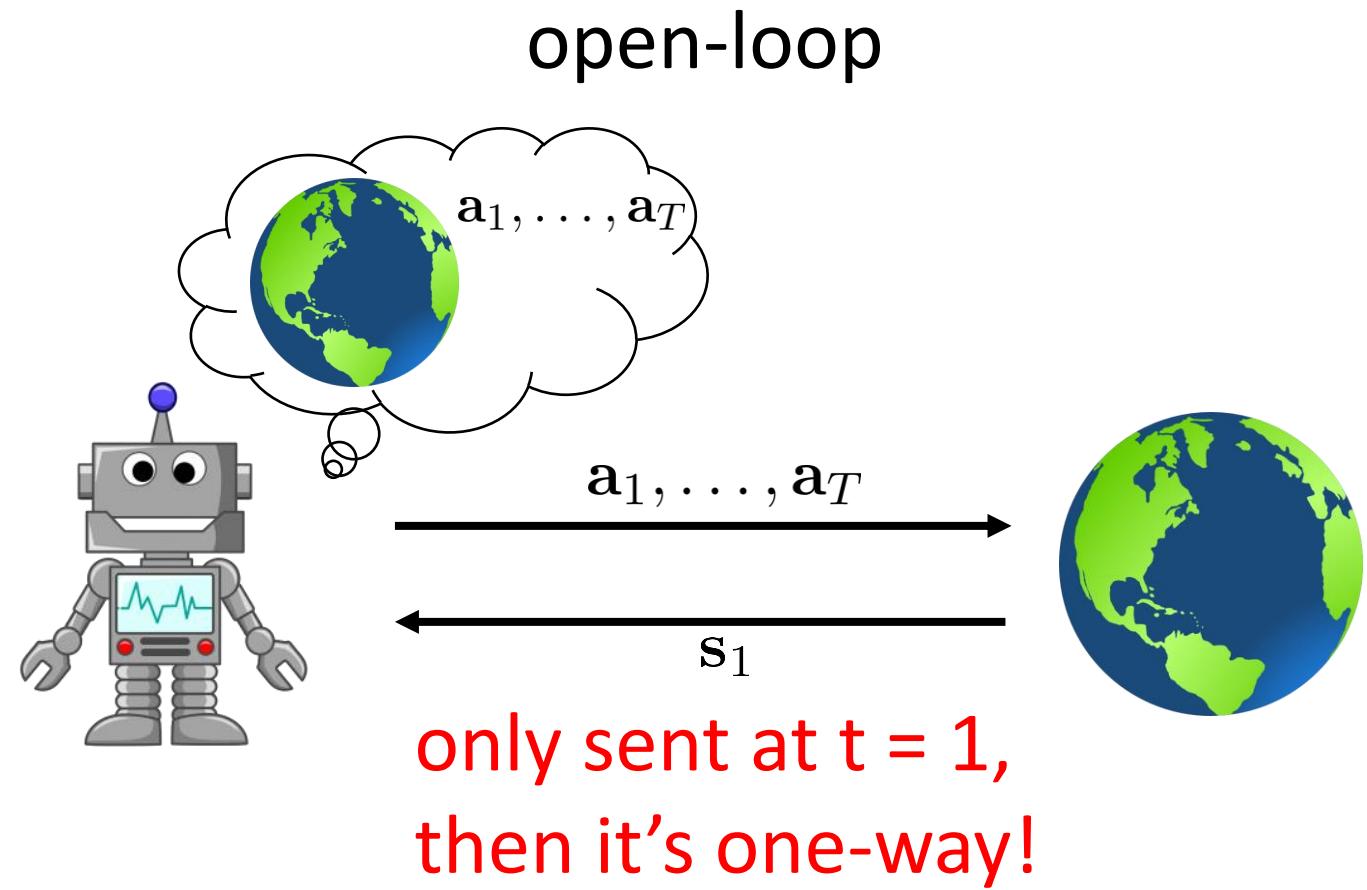
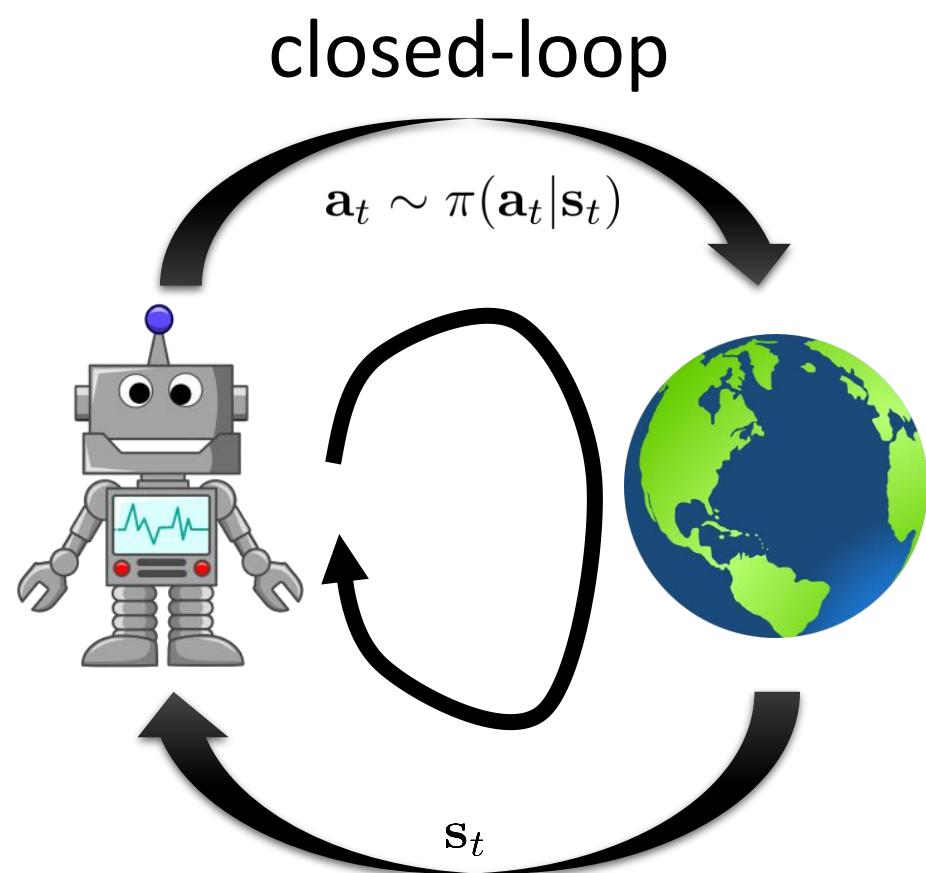
$$p_{\theta}(\mathbf{s}_1, \dots, \mathbf{s}_T | \mathbf{a}_1, \dots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} E \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \dots, \mathbf{a}_T \right]$$

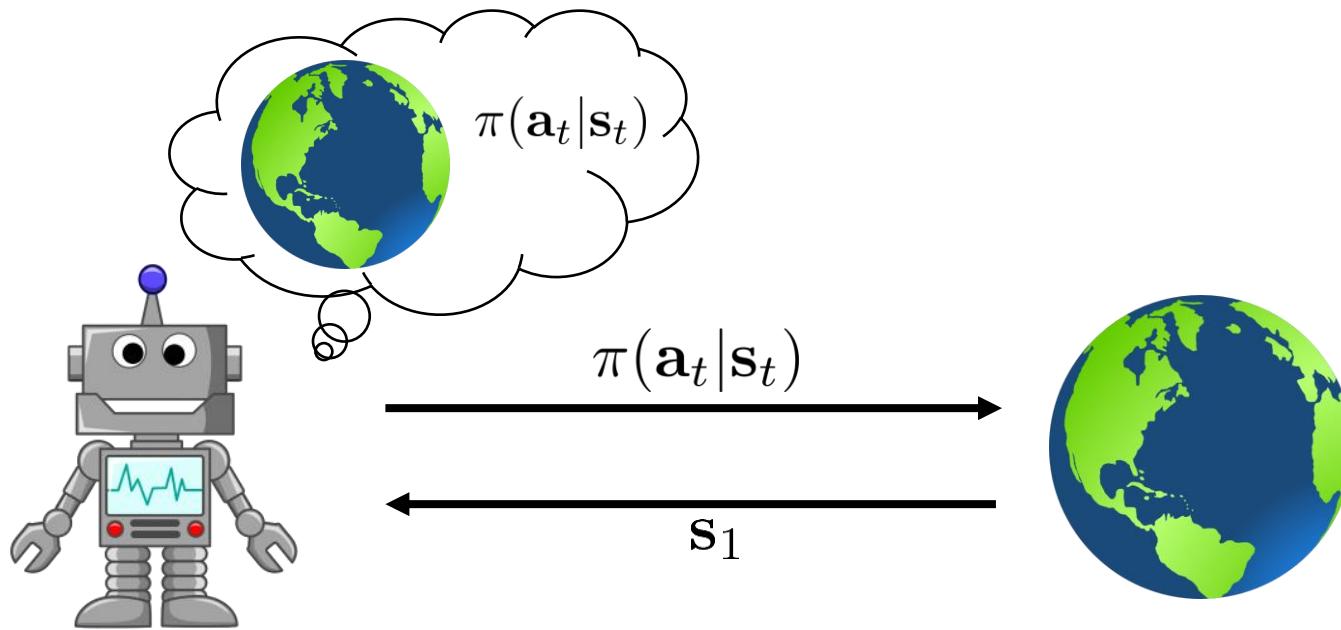
why is this suboptimal?

Aside: terminology

what is this “loop”?



The stochastic closed-loop case



$$p(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \pi(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg \max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

form of π ?

neural net

time-varying linear

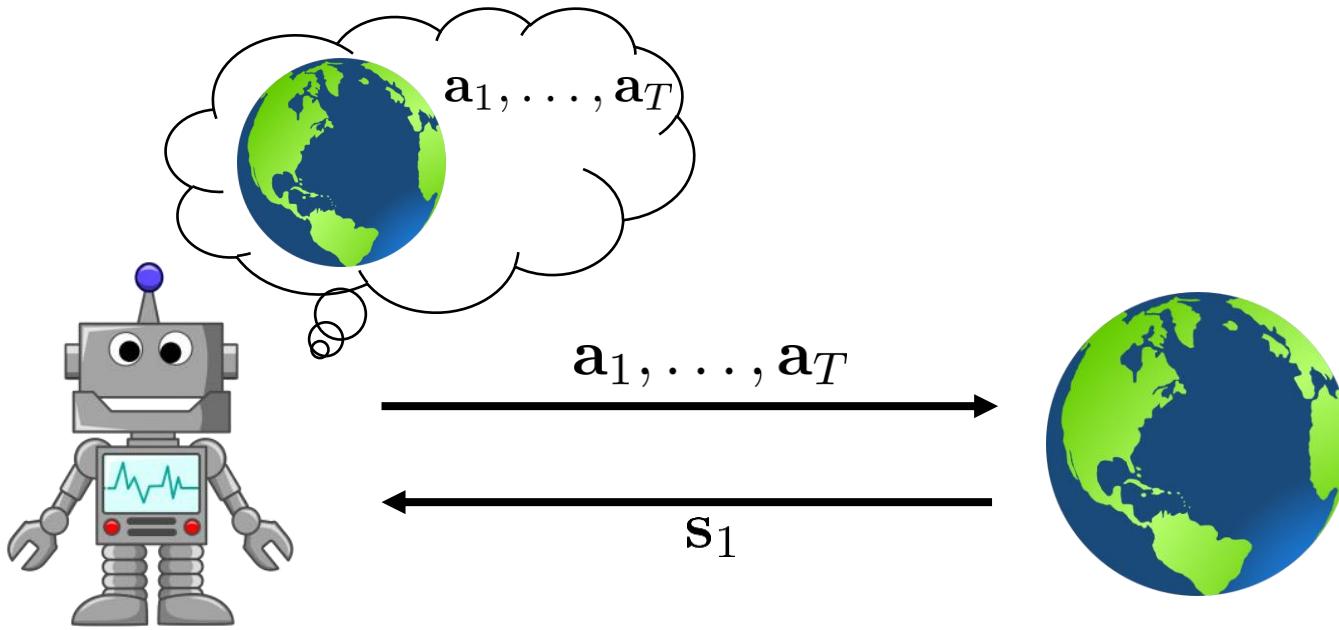
$\mathbf{K}_t \mathbf{s}_t + \mathbf{k}_t$

global
local

(more on this later)

Open-Loop Planning

But for now, open-loop planning



$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

Stochastic optimization

abstract away optimal control/planning:

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} J(\underbrace{\mathbf{a}_1, \dots, \mathbf{a}_T}_{})$$

$$\mathbf{A} = \arg \max_{\mathbf{A}} J(\mathbf{A})$$

don't care what this is

simplest method: guess & check

“random shooting method”

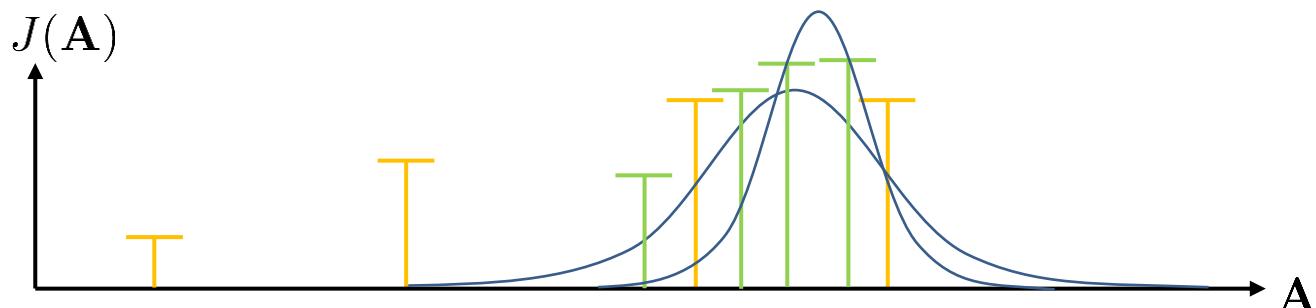
1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose \mathbf{A}_i based on $\arg \max_i J(\mathbf{A}_i)$

Cross-entropy method (CEM)

1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)

2. choose \mathbf{A}_i based on $\arg \max_i J(\mathbf{A}_i)$

can we do better?



cross-entropy method with continuous-valued inputs:

- 1. sample $\mathbf{A}_1, \dots, \mathbf{A}_N$ from $p(\mathbf{A})$
- 2. evaluate $J(\mathbf{A}_1), \dots, J(\mathbf{A}_N)$
- 3. pick the *elites* $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$
- 4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$

typically use Gaussian distribution

see also: CMA-ES (sort of like CEM with momentum)

What's the upside?

1. Very fast if parallelized
2. Extremely simple

What's the problem?

1. Very harsh dimensionality limit
2. Only open-loop planning

Discrete case: Monte Carlo tree search (MCTS)

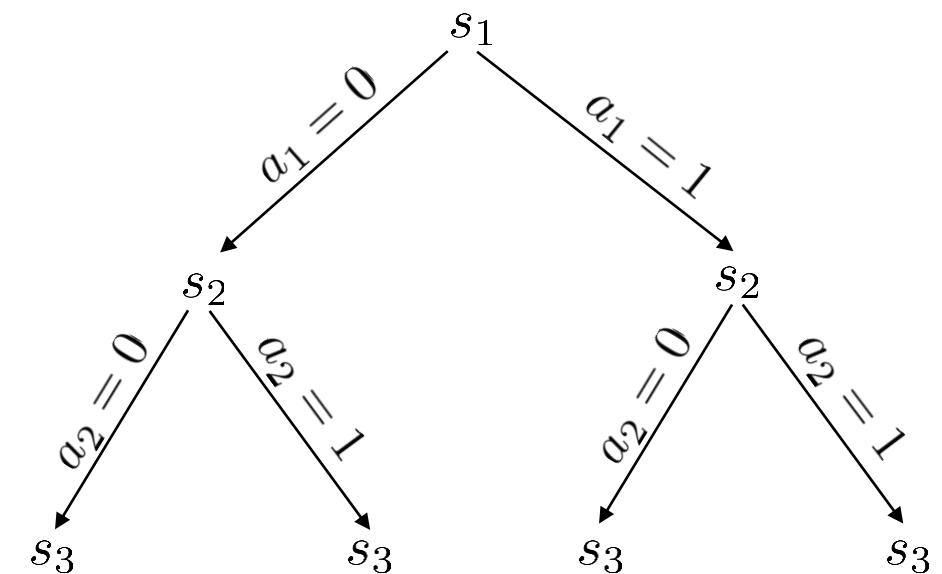
discrete planning as a search problem



s_t



a_t



Discrete case: Monte Carlo tree search (MCTS)

how to approximate value without full tree?

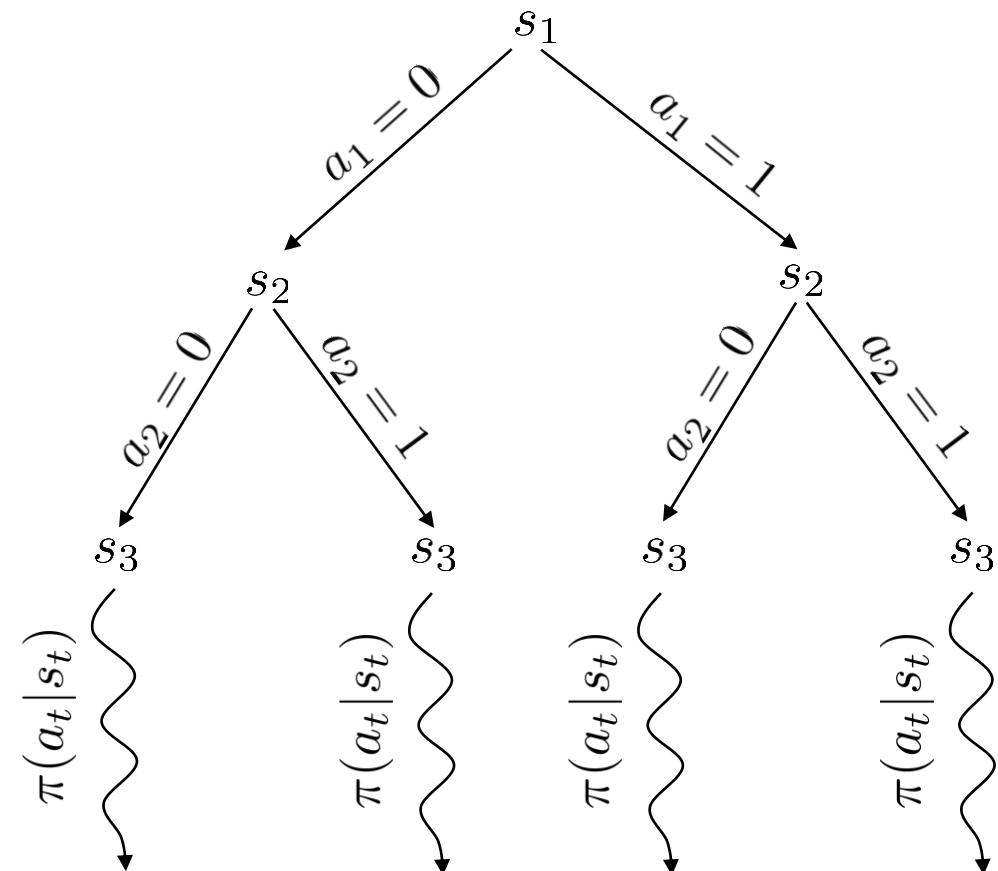


s_t

e.g., random policy



a_t



Discrete case: Monte Carlo tree search (MCTS)

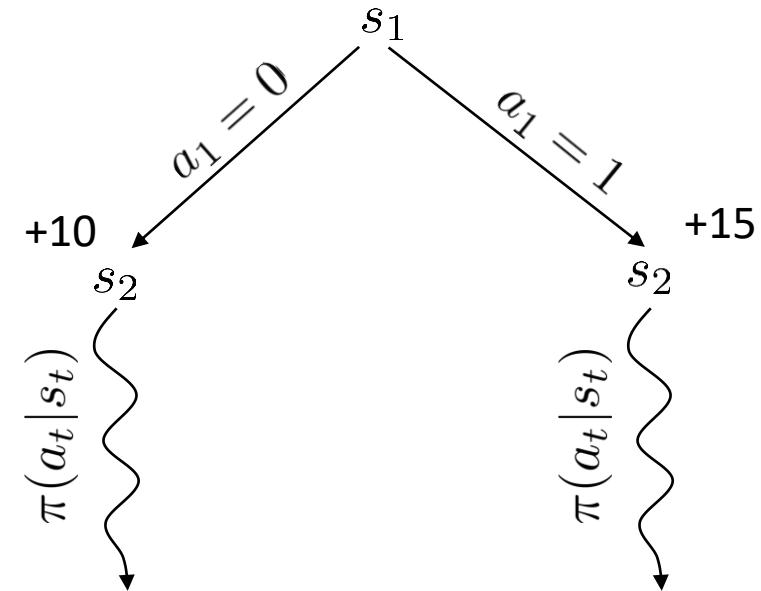
can't search all paths – where to search first?



s_t



a_t



intuition: choose nodes with best reward, but also prefer rarely visited nodes

Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

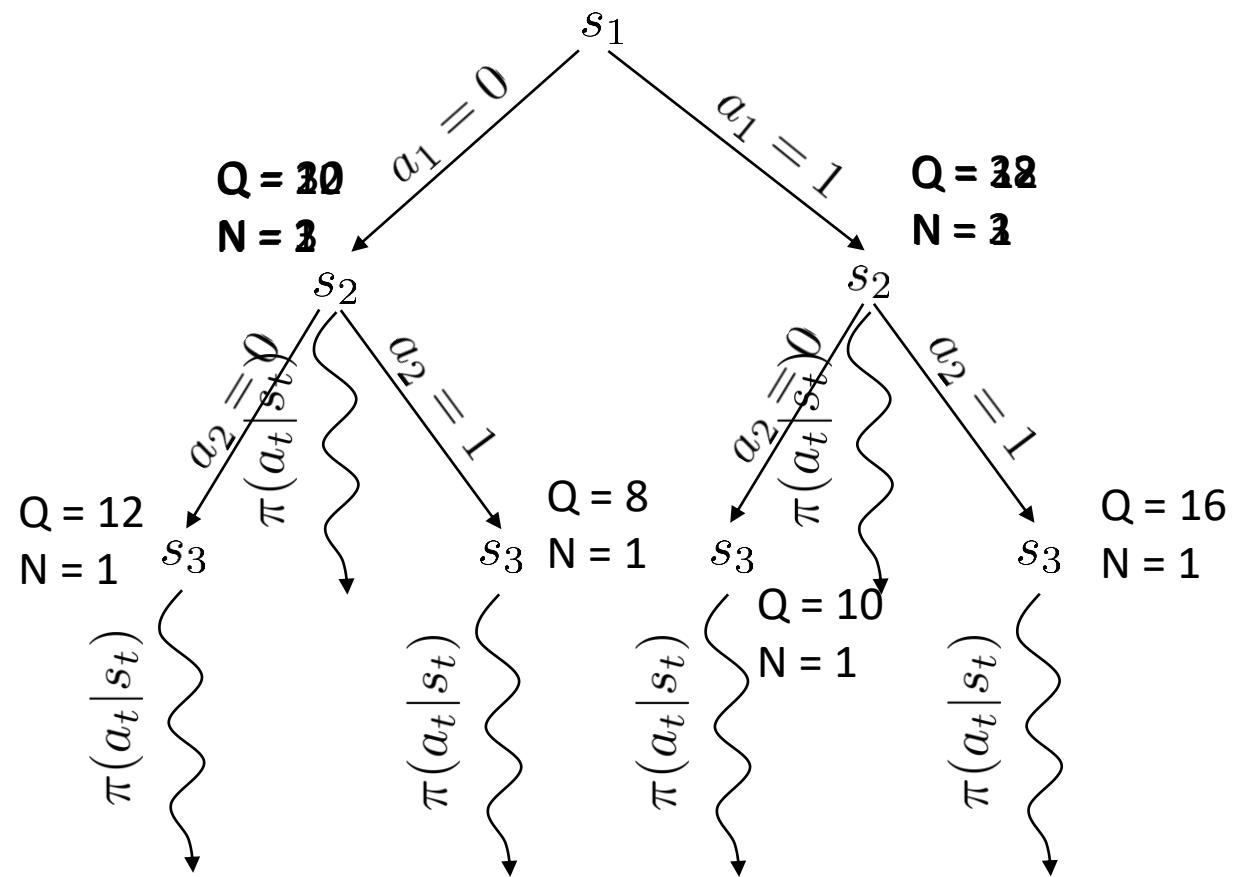
- 1. find a leaf s_l using $\text{TreePolicy}(s_1)$
- 2. evaluate the leaf using $\text{DefaultPolicy}(s_l)$
- 3. update all values in tree between s_1 and s_l

take best action from s_1

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t
else choose child with best Score(s_{t+1})

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C \sqrt{\frac{2 \ln N(s_{t-1})}{N(s_t)}}$$



Additional reading

1. Browne, Powley, Whitehouse, Lucas, Cowling, Röhlfs, Tavener, Perez, Samothrakis, Colton. (2012). A Survey of Monte Carlo Tree Search Methods.
 - Survey of MCTS methods and basic summary.

Trajectory Optimization with Derivatives

Can we use derivatives?

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$

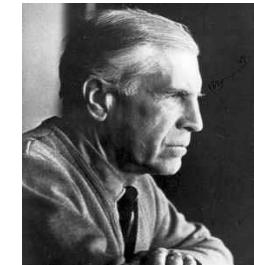
usual story: differentiate via backpropagation and optimize!

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$

\mathbf{s}_t – state
 \mathbf{a}_t – action

\mathbf{x}_t – state
 \mathbf{u}_t – action

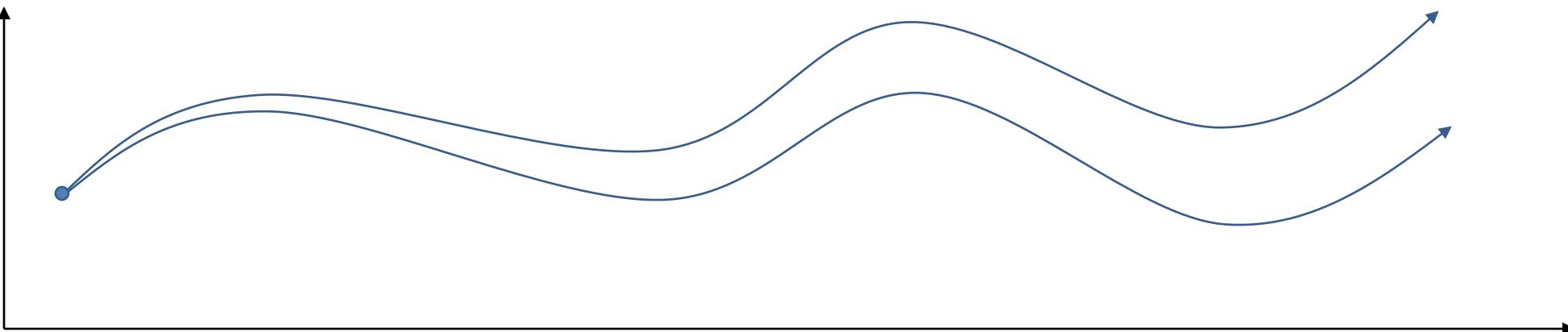
in practice, it really helps to use a 2nd order method!



Shooting methods vs collocation

shooting method: optimize over actions only

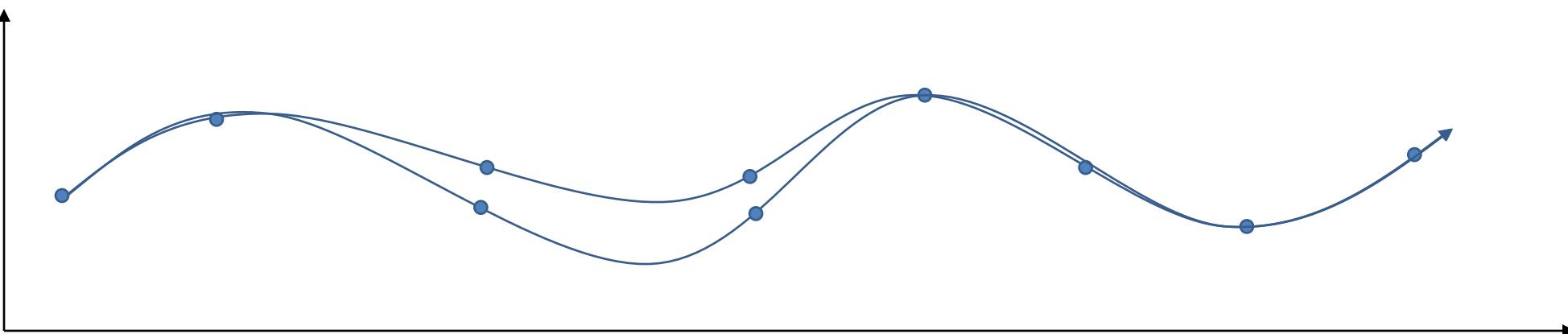
$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$



Shooting methods vs collocation

collocation method: optimize over actions and states, with constraints

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T, \mathbf{x}_1, \dots, \mathbf{x}_T} \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$



Linear case: LQR

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

linear

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

quadratic

Linear case: LQR

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + \underbrace{c(f(f(\dots) \dots), \mathbf{u}_T)}_{\mathbf{x}_T \text{ (unknown)}}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

only term that depends on \mathbf{u}_T

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

Base case: solve for \mathbf{u}_T only

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

$$\mathbf{C}_T = \begin{bmatrix} \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \\ \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \end{bmatrix}$$

$$\mathbf{c}_T = \begin{bmatrix} \mathbf{c}_{\mathbf{x}_T} \\ \mathbf{c}_{\mathbf{u}_T} \end{bmatrix}$$

$$\nabla_{\mathbf{u}_T} Q(\mathbf{x}_T, \mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T}$$

$$\mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} (\mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T}) \quad \mathbf{u}_T = \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \quad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{c}_{\mathbf{u}_T}$$

Linear case: LQR

$$\mathbf{u}_T = \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \quad \mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \quad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{c}_{\mathbf{u}_T}$$

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

Since \mathbf{u}_T is fully determined by \mathbf{x}_T , we can eliminate it via substitution!

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{c}_T$$

$$\begin{aligned} V(\mathbf{x}_T) &= \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \\ &\quad \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{k}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{k}_T + \mathbf{x}_T^T \mathbf{c}_{\mathbf{x}_T} + \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{c}_{\mathbf{u}_T} + \text{const} \end{aligned}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T$$

$$\mathbf{V}_T = \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{K}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{K}_T$$

$$\mathbf{v}_T = \mathbf{c}_{\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{k}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{k}_T$$

Linear case: LQR

Solve for \mathbf{u}_{T-1} in terms of \mathbf{x}_{T-1}

\mathbf{u}_{T-1} affects \mathbf{x}_T !

$$f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{x}_T = \mathbf{F}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \mathbf{f}_{T-1}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

$\brace{V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T}$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

Linear case: LQR

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \underbrace{\begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{Q}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{q}_{T-1}$$

$$\mathbf{Q}_{T-1} = \mathbf{C}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}$$

$$\mathbf{q}_{T-1} = \mathbf{c}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{v}_T$$

$$\nabla_{\mathbf{u}_{T-1}} Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}} \mathbf{x}_{T-1} + \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}} \mathbf{u}_{T-1} + \mathbf{q}_{\mathbf{u}_{T-1}}^T = 0$$

$$\mathbf{u}_{T-1} = \mathbf{K}_{T-1} \mathbf{x}_{T-1} + \mathbf{k}_{T-1}$$

$$\mathbf{K}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}}$$

$$\mathbf{k}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{q}_{\mathbf{u}_{T-1}}$$

Linear case: LQR

Backward recursion

for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

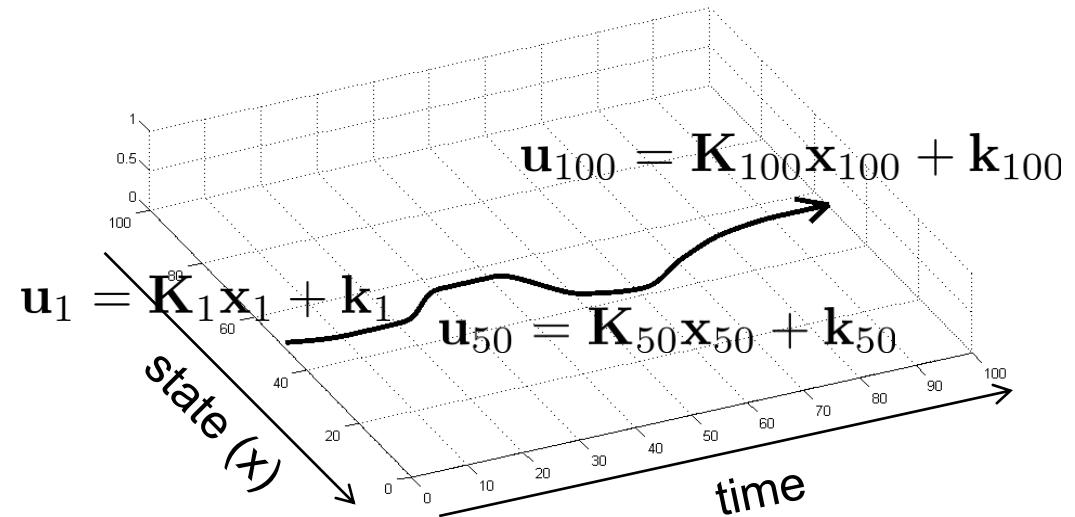
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$



we know \mathbf{x}_1 !

Forward recursion

for $t = 1$ to T :

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

Linear case: LQR

Backward recursion

for $t = T$ to 1:

total cost from now until end if we take \mathbf{u}_t from state \mathbf{x}_t

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$

total cost from now until end from state \mathbf{x}_t

$$V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t)$$

LQR for Stochastic and Nonlinear Systems

Stochastic dynamics

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$$

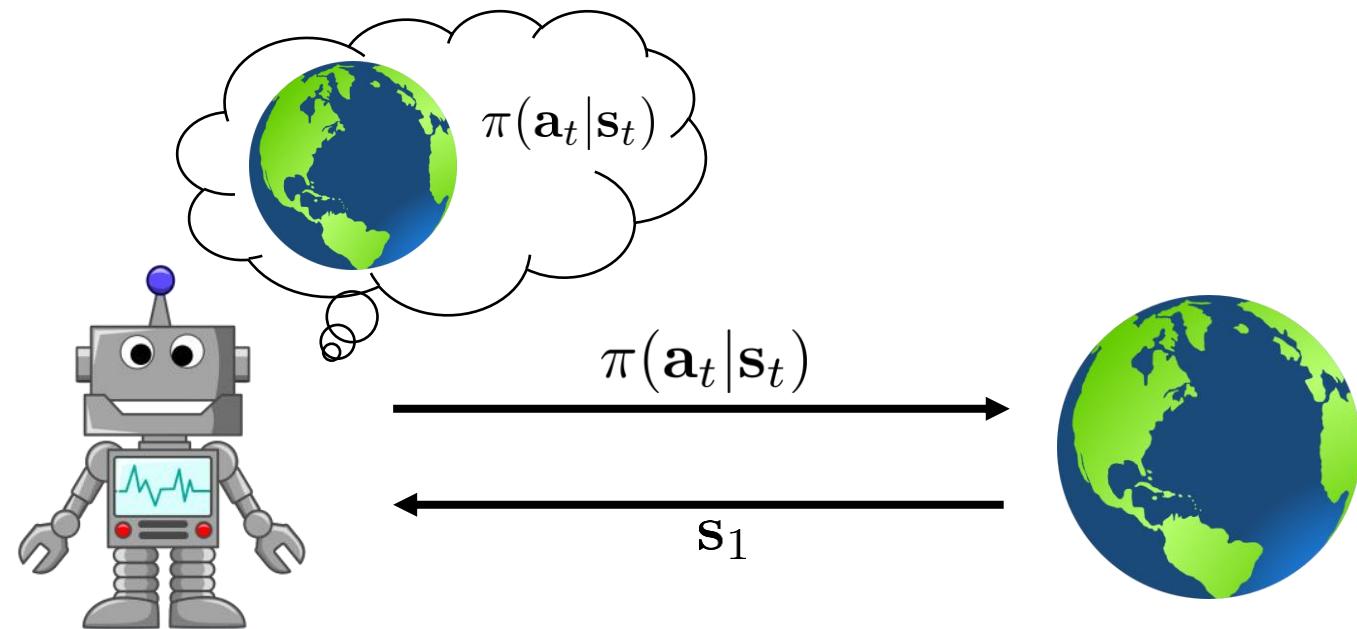
$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N} \left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \Sigma_t \right)$$

Solution: choose actions according to $\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$

$\mathbf{x}_t \sim p(\mathbf{x}_t)$, no longer deterministic, but $p(\mathbf{x}_t)$ is Gaussian

no change to algorithm! can ignore Σ_t due to symmetry of Gaussians
(checking this is left as an exercise; hint: the expectation of a quadratic under a Gaussian has an analytic solution)

The stochastic closed-loop case



$$p(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^T \pi(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

$$\pi = \arg \max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

form of π ?

time-varying linear
 $\mathbf{K}_t \mathbf{s}_t + \mathbf{k}_t$

Nonlinear case: DDP/iterative LQR

Linear-quadratic assumptions:

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t \quad c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

Can we *approximate* a nonlinear system as a linear-quadratic system?

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

Nonlinear case: DDP/iterative LQR

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$\bar{f}(\delta \mathbf{x}_t, \delta \mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}$$

$$\bar{c}(\delta \mathbf{x}_t, \delta \mathbf{u}_t) = \frac{1}{2} \underbrace{\begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}^T}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \underbrace{\mathbf{C}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} + \underbrace{\begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}^T}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \underbrace{\mathbf{c}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)}$$

$$\delta \mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$$

$$\delta \mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$$

Now we can run LQR with dynamics \bar{f} , cost \bar{c} , state $\delta \mathbf{x}_t$, and action $\delta \mathbf{u}_t$

Nonlinear case: DDP/iterative LQR

Iterative LQR (simplified pseudocode)

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta \mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta \mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with real nonlinear dynamics and $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass

Nonlinear case: DDP/iterative LQR

Why does this work?

Compare to Newton's method for computing $\min_{\mathbf{x}} g(\mathbf{x})$:

until convergence:

$$\mathbf{g} = \nabla_{\mathbf{x}} g(\hat{\mathbf{x}})$$

$$\mathbf{H} = \nabla_{\mathbf{x}}^2 g(\hat{\mathbf{x}})$$

$$\hat{\mathbf{x}} \leftarrow \arg \min_{\mathbf{x}} \frac{1}{2} (\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H} (\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T (\mathbf{x} - \hat{\mathbf{x}})$$

Iterative LQR (iLQR) is the same idea: locally approximate a complex nonlinear function via Taylor expansion

In fact, iLQR is an approximation of Newton's method for solving

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)), \mathbf{u}_T)$$

Nonlinear case: DDP/iterative LQR

In fact, iLQR is an approximation of Newton's method for solving

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)), \mathbf{u}_T)$$

To get Newton's method, need to use *second order* dynamics approximation:

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} + \frac{1}{2} \left(\nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \cdot \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} \right) \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}$$

differential dynamic programming (DDP)

Nonlinear case: DDP/iterative LQR

$$\hat{\mathbf{x}} \leftarrow \arg \min_{\mathbf{x}} \frac{1}{2} (\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H} (\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T (\mathbf{x} - \hat{\mathbf{x}})$$

why is this a bad idea?

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

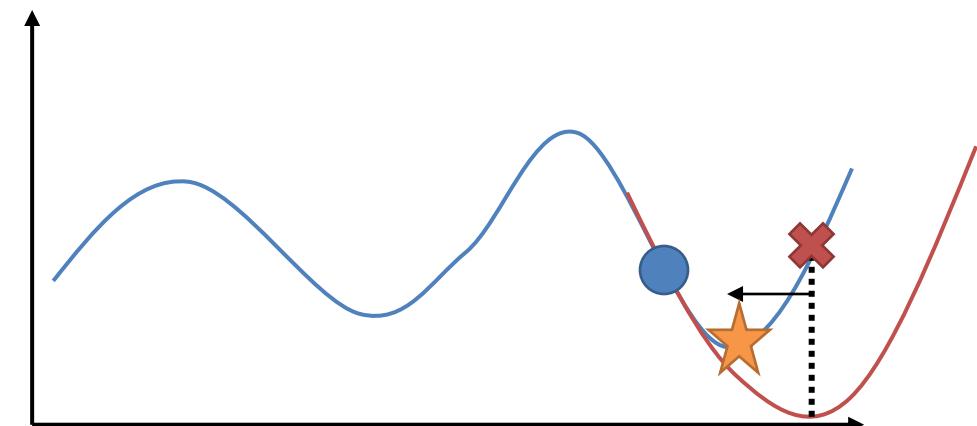
$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta \mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta \mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass



search over α
until improvement achieved



Case Study and Additional Readings

Case study: nonlinear model-predictive control

Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization

Yuval Tassa, Tom Erez and Emanuel Todorov
University of Washington

every time step:

observe the state \mathbf{x}_t

use iLQR to plan $\mathbf{u}_t, \dots, \mathbf{u}_T$ to minimize $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$

execute action \mathbf{u}_t , discard $\mathbf{u}_{t+1}, \dots, \mathbf{u}_{t+T}$

Synthesis of Complex Behaviors with Online Trajectory Optimization

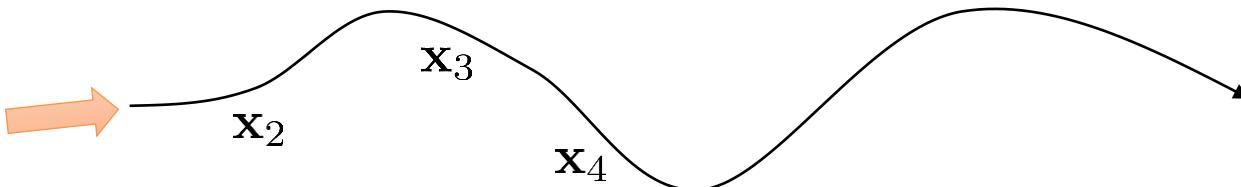
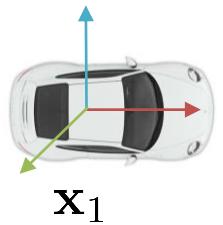
Yuval Tassa, Tom Erez & Emo Todorov

IEEE International Conference
on Intelligent Robots and Systems
2012

Additional reading

1. Mayne, Jacobson. (1970). Differential dynamic programming.
 - Original differential dynamic programming algorithm.
2. Tassa, Erez, Todorov. (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.
 - Practical guide for implementing non-linear iterative LQR.
3. Levine, Abbeel. (2014). Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics.
 - Probabilistic formulation and trust region alternative to deterministic line search.

What's wrong with known dynamics?



Next time: learning the dynamics model

Model-Based Reinforcement Learning

CS 285

Instructor: Sergey Levine
UC Berkeley



Today's Lecture

1. Basics of model-based RL: learn a model, use model for control
 - Why does naïve approach not work?
 - The effect of distributional shift in model-based RL
2. Uncertainty in model-based RL
3. Model-based RL with complex observations
4. Next time: **policy learning** with model-based RL
 - Goals:
 - Understand how to build model-based RL algorithms
 - Understand the important considerations for model-based RL
 - Understand the tradeoffs between different model class choices

Why learn the model?

If we knew $f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$, we could use the tools from last week.

(or $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ in the stochastic case)

So let's learn $f(\mathbf{s}_t, \mathbf{a}_t)$ from data, and *then* plan through it!

model-based reinforcement learning version 0.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

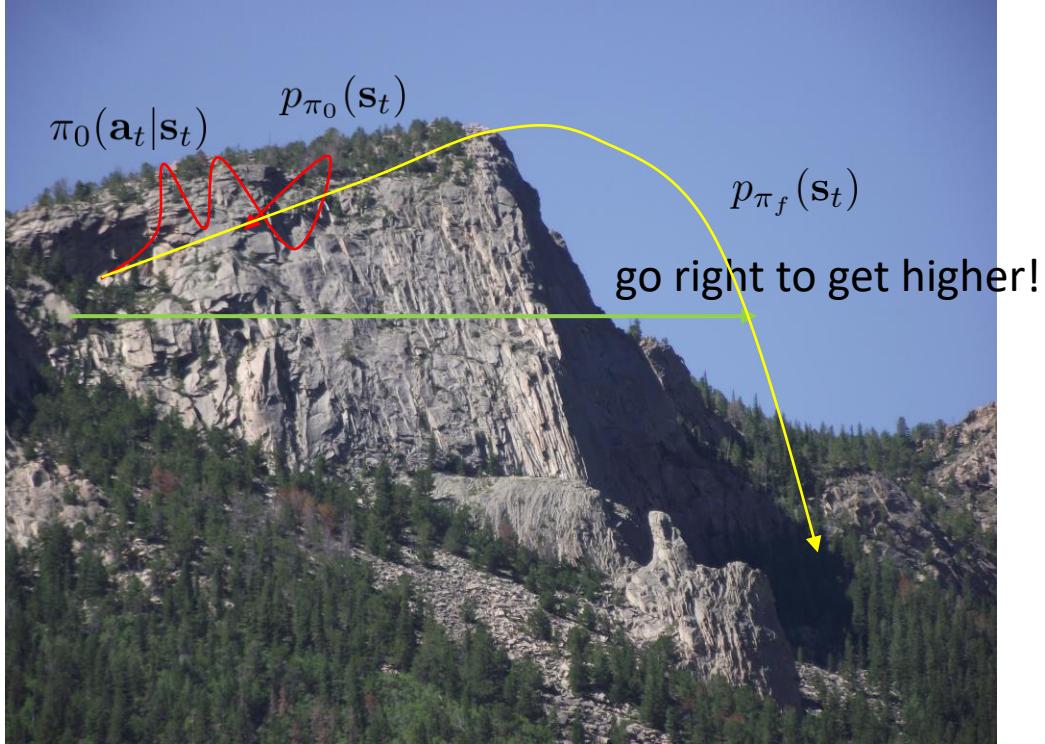
Does it work?

Yes!

- Essentially how system identification works in classical robotics
- Some care should be taken to design a good base policy
- Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters

Does it work?

No!



1. run base policy $\pi_0(\mathbf{a}_t | \mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

$$p_{\pi_f}(\mathbf{s}_t) \neq p_{\pi_0}(\mathbf{s}_t)$$

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes

Can we do better?

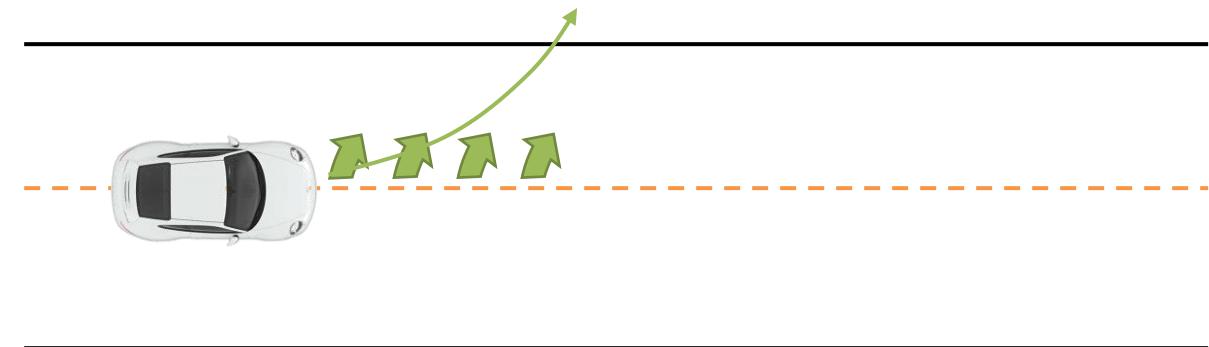
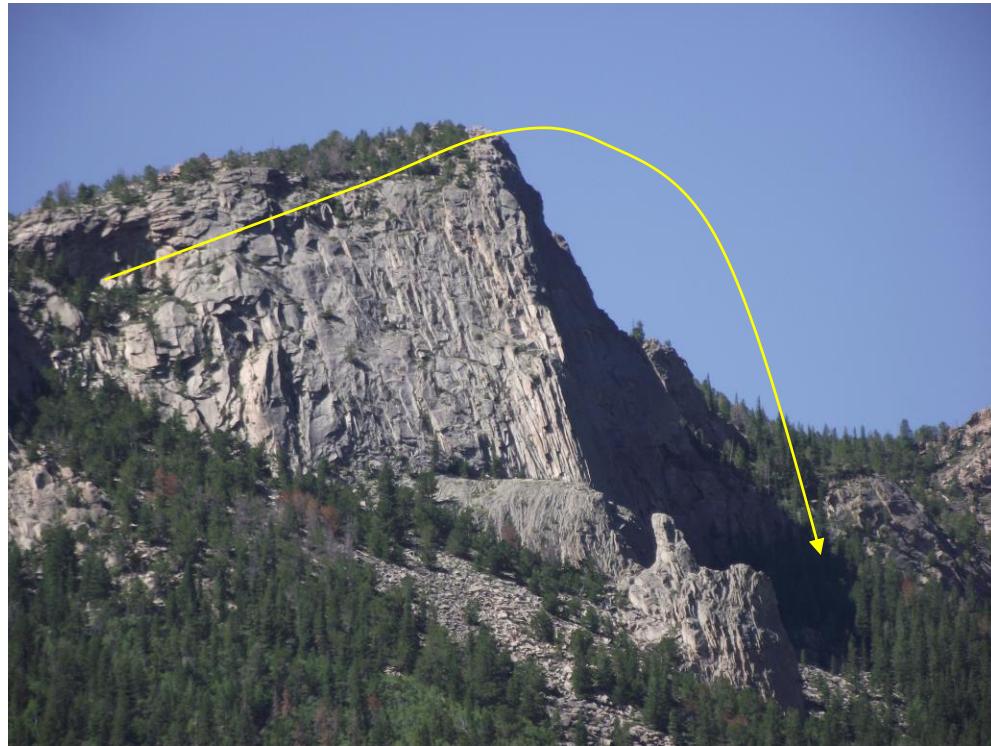
can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

where have we seen that before? need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

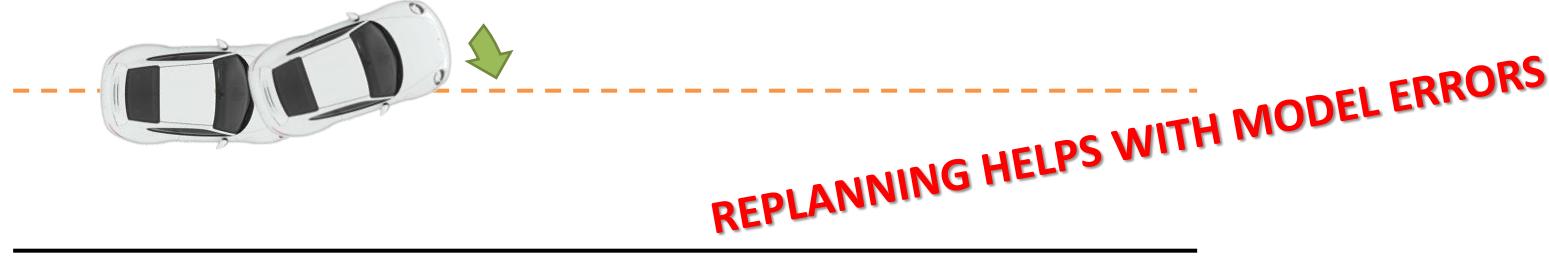
model-based reinforcement learning version 1.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute those actions and add the resulting data $\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_j\}$ to \mathcal{D}

What if we make a mistake?



Can we do better?



model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t | \mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state \mathbf{s}' (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset \mathcal{D}

every N steps

This will be on HW4!

How to replan?

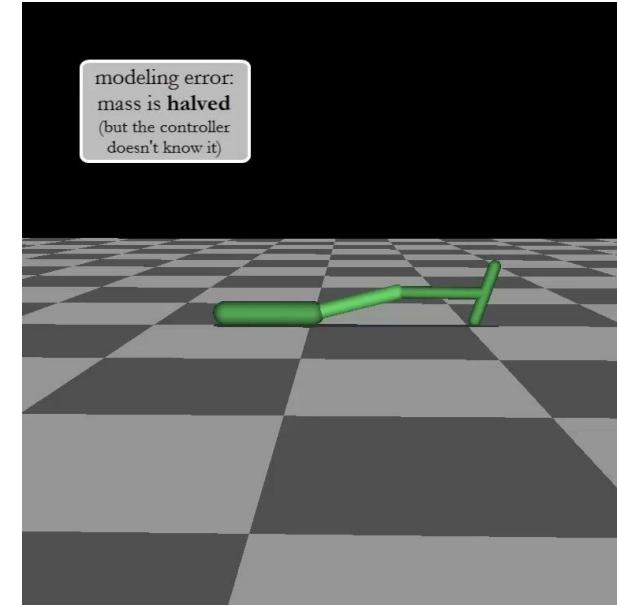
model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state \mathbf{s}' (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset \mathcal{D}

every N steps

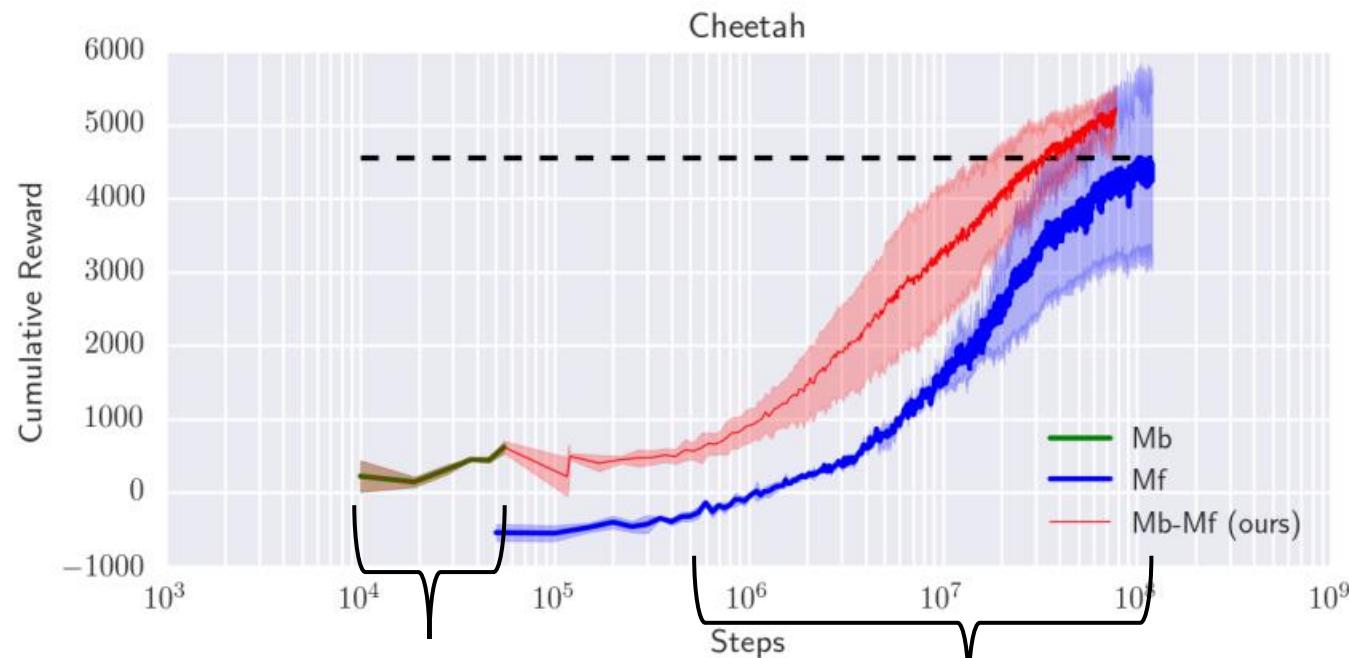
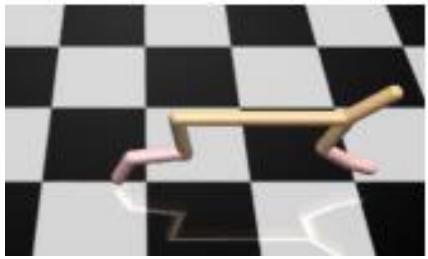


- The more you replan, the less perfect each individual plan needs to be
- Can use shorter horizons
- Even random sampling can often work well here!



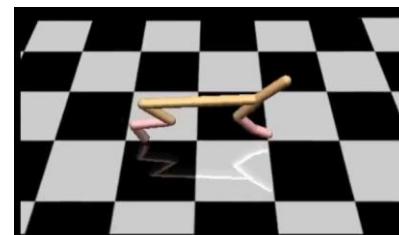
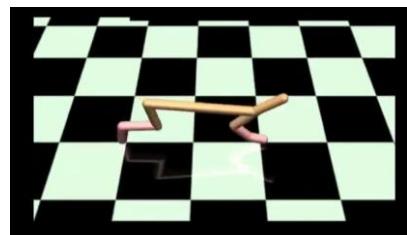
Uncertainty in Model-Based RL

A performance gap in model-based RL

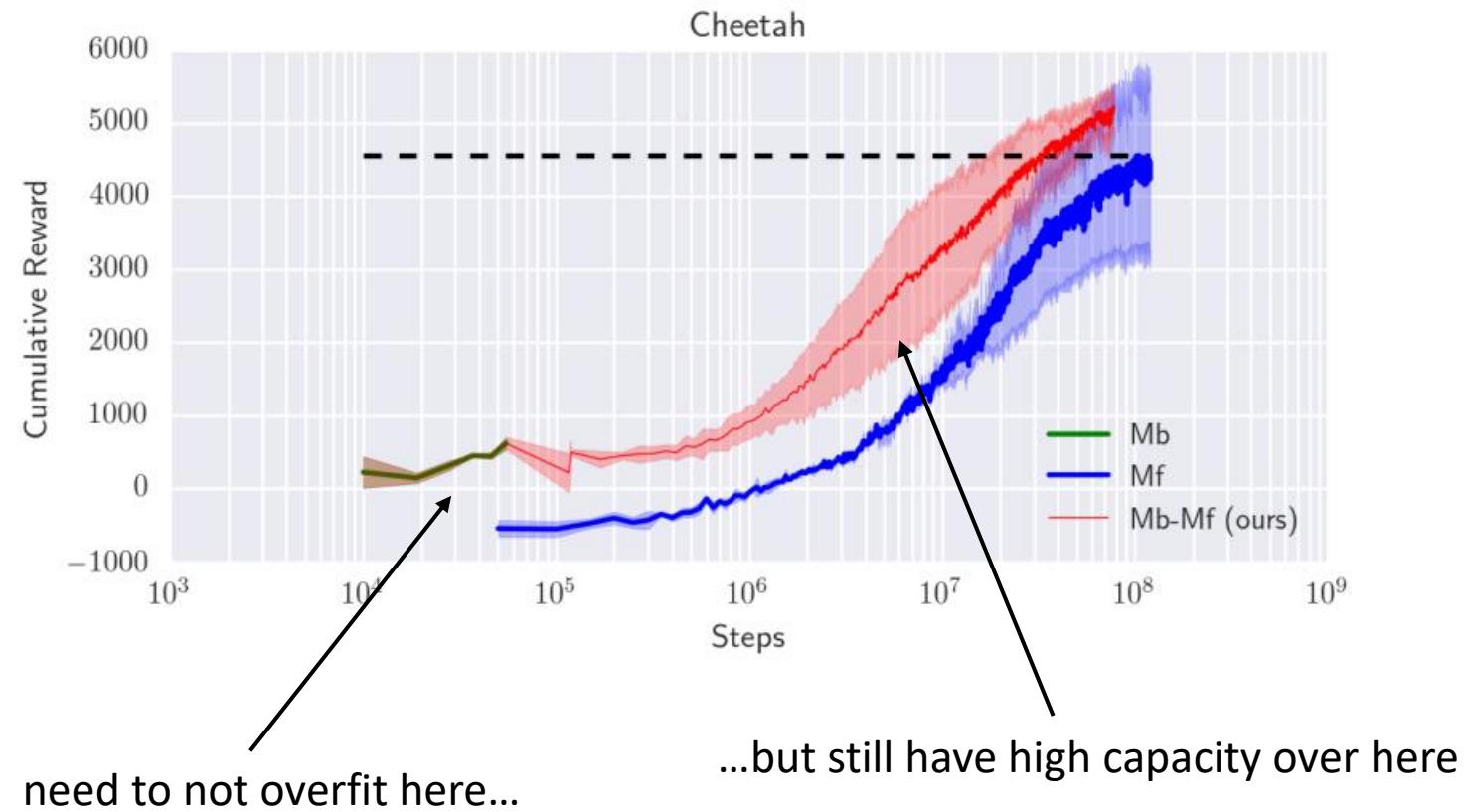
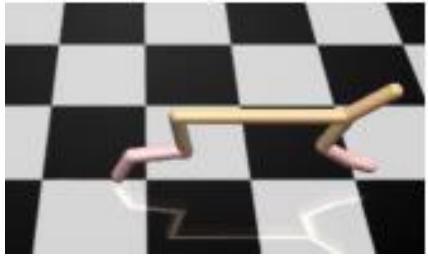


pure model-based
(about 10 minutes real time)

model-free training
(about 10 days...)



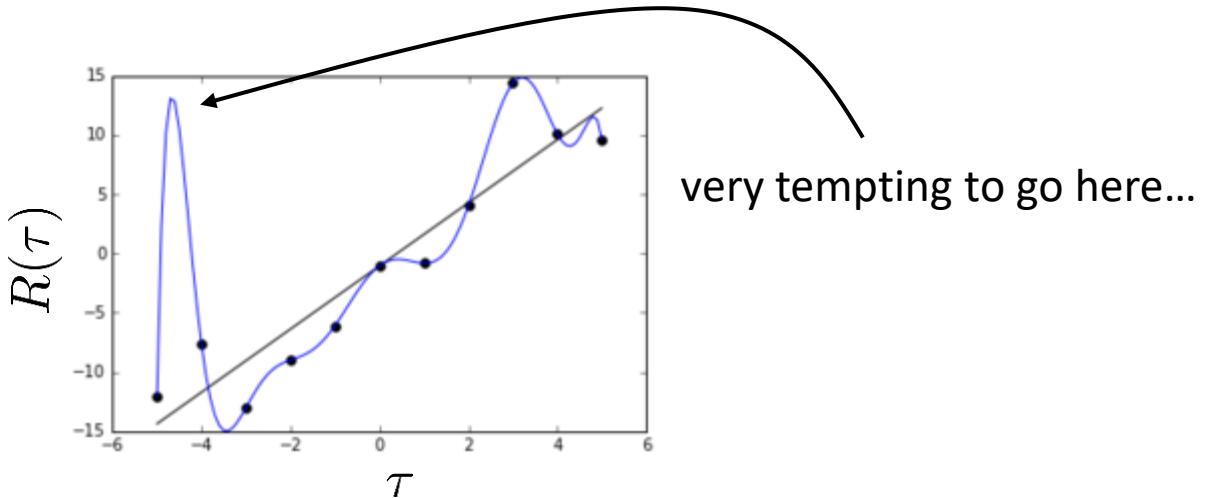
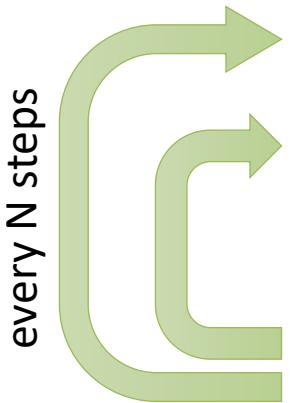
Why the performance gap?



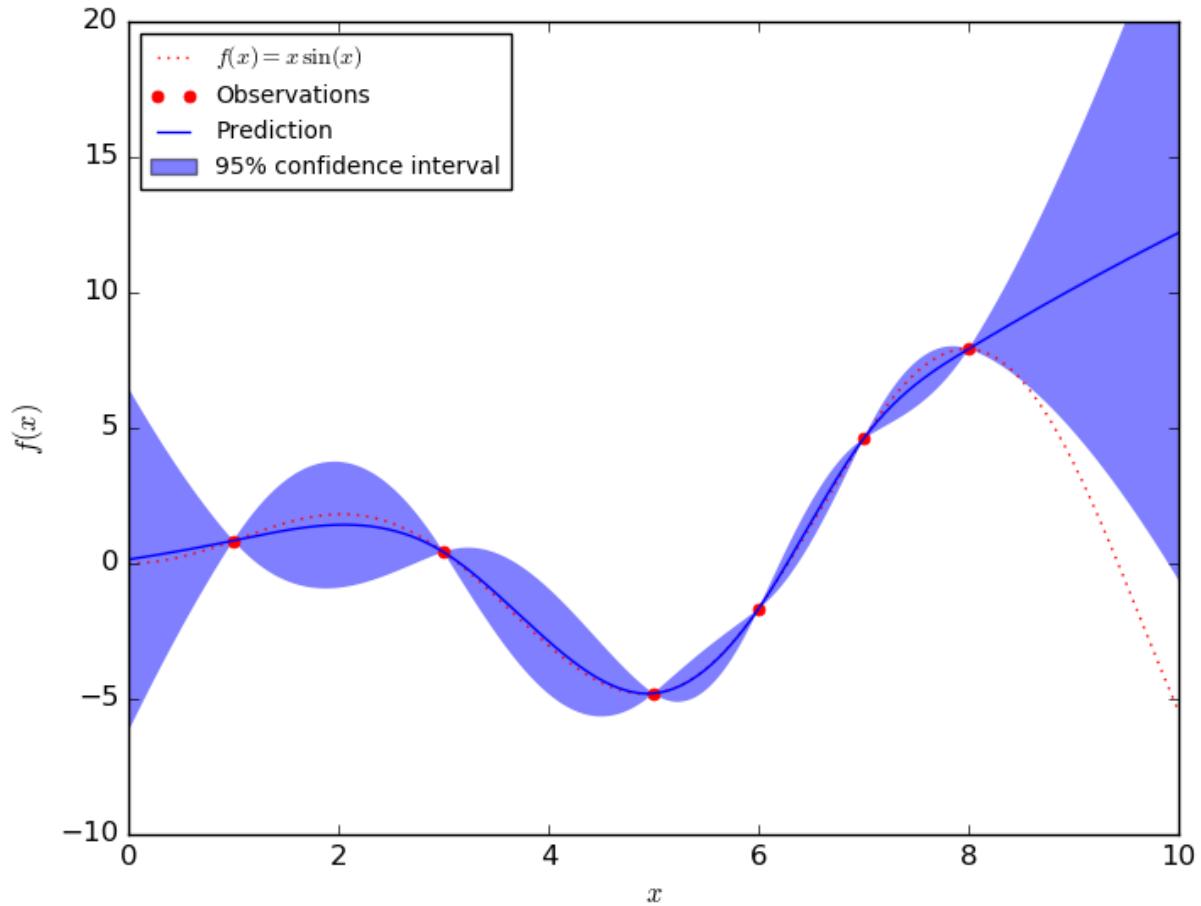
Why the performance gap?

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state \mathbf{s}' (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset \mathcal{D}



How can uncertainty estimation help?



$$p_{\pi_f}(\mathbf{s}_t) \neq p_{\pi_0}(\mathbf{s}_t)$$



expected reward under high-variance prediction
is **very** low, even though mean is the same!

Intuition behind uncertainty-aware RL

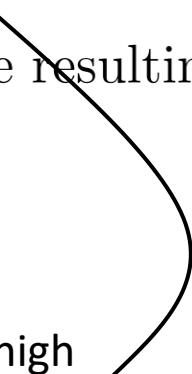
model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state \mathbf{s}' (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset \mathcal{D}



every N steps

only take actions for which we think we'll get high reward in expectation (w.r.t. uncertain dynamics)



This avoids “exploiting” the model

The model will then adapt and get better

There are a few caveats...



Need to explore to get better

Expected value is not the same as pessimistic value

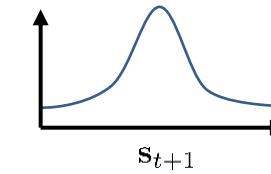
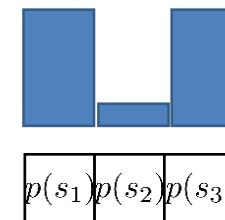
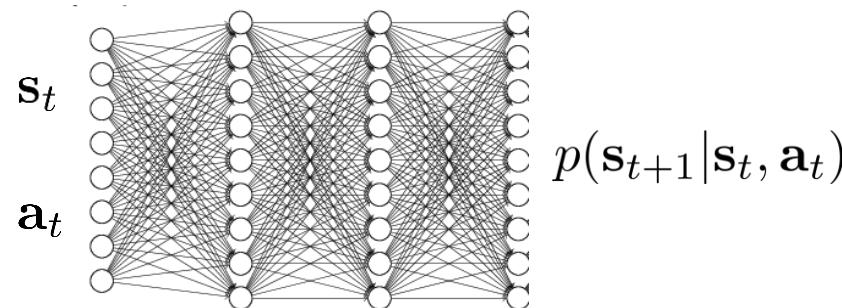
Expected value is not the same as optimistic value

...but expected value is often a good start

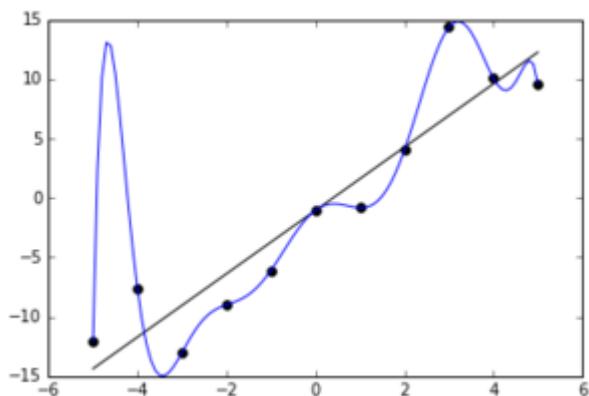
Uncertainty-Aware Neural Net Models

How can we have uncertainty-aware models?

Idea 1: use output entropy



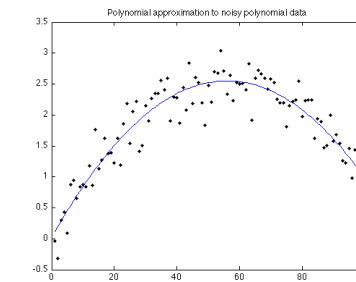
why is this not enough?



Two types of uncertainty:

aleatoric or statistical uncertainty →

← *epistemic or model uncertainty*



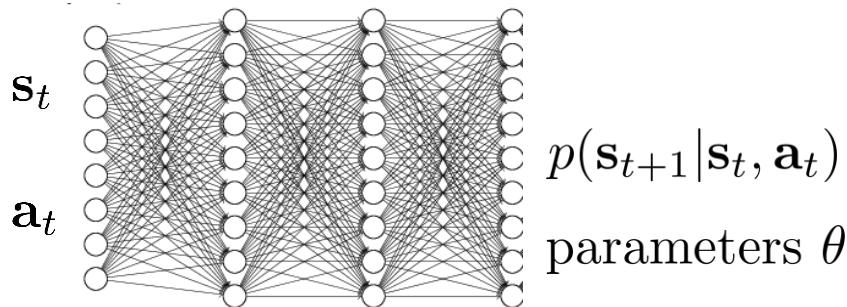
"the model is certain about the data, but we are not certain about the model"

what is the variance here?

How can we have uncertainty-aware models?

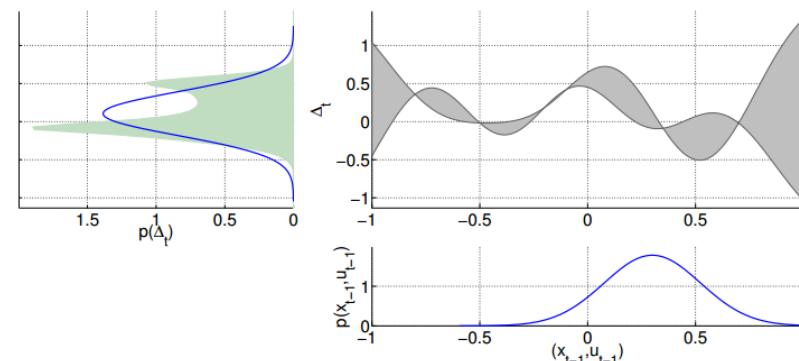
Idea 2: estimate model uncertainty

“the model is certain about the data, but we are not certain about the model”



predict according to:

$$\int p(s_{t+1}|s_t, a_t, \theta) p(\theta|\mathcal{D}) d\theta$$



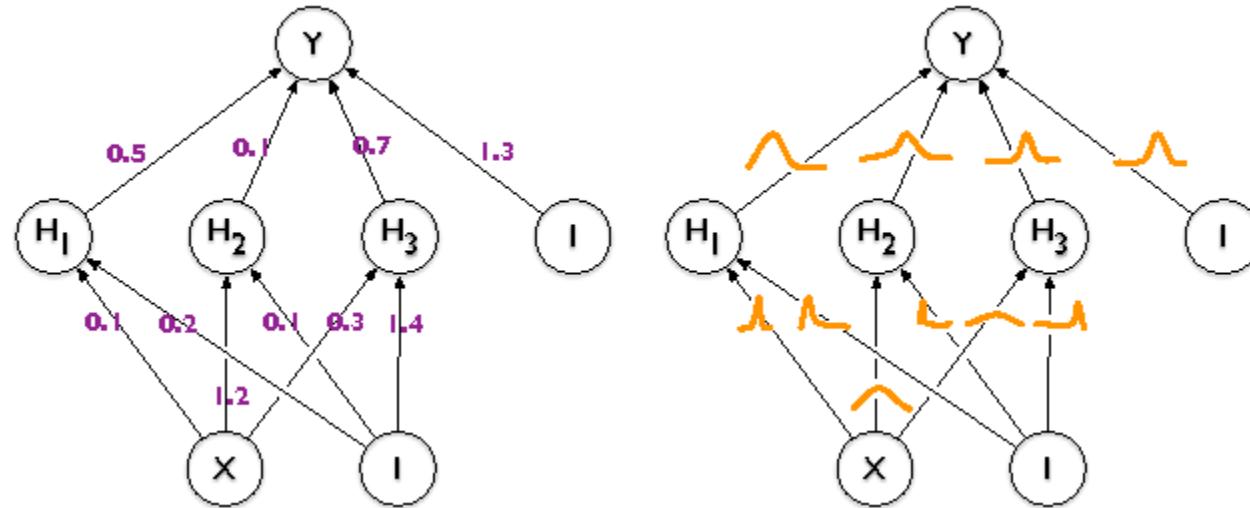
usually, we estimate

$$\arg \max_{\theta} \log p(\theta|\mathcal{D}) = \arg \max_{\theta} \log p(\mathcal{D}|\theta)$$

can we instead estimate $p(\theta|\mathcal{D})$?

the entropy of this tells us
the model uncertainty!

Quick overview of Bayesian neural networks



common approximation:

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

expected weight

uncertainty
about the weight

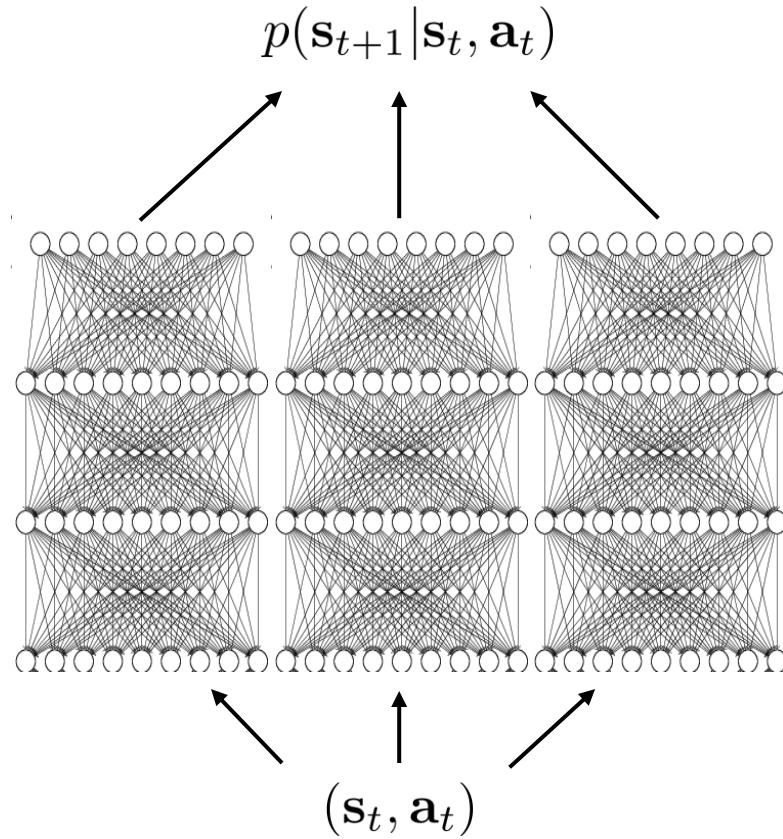
For more, see:

Blundell et al., Weight Uncertainty in Neural Networks

Gal et al., Concrete Dropout

We'll learn more about variational inference later!

Bootstrap ensembles



Train multiple models and see if they agree!

formally: $p(\theta|\mathcal{D}) \approx \frac{1}{N} \sum_i \delta(\theta_i)$

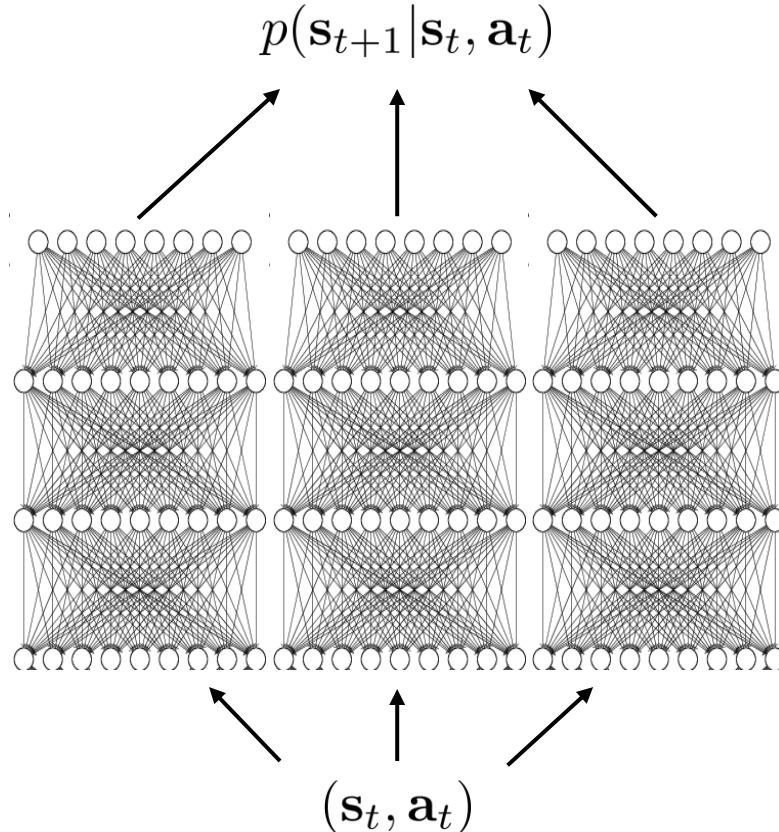
$$\int p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t, \theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{N} \sum_i p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t, \theta_i)$$

How to train?

Main idea: need to generate
“independent” datasets to get
“independent” models

θ_i is trained on \mathcal{D}_i , sampled *with replacement* from \mathcal{D}

Bootstrap ensembles in deep learning



This basically works

Very crude approximation, because the number of models is usually small (< 10)

Resampling with replacement is usually unnecessary, because SGD and random initialization usually makes the models sufficiently independent

Planning with Uncertainty, Examples

How to plan with uncertainty

Before: $J(\mathbf{a}_1, \dots, \mathbf{a}_H) = \sum_{t=1}^H r(\mathbf{s}_t, \mathbf{a}_t)$, where $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$

Now: $J(\mathbf{a}_1, \dots, \mathbf{a}_H) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^H r(\mathbf{s}_{t,i}, \mathbf{a}_t)$, where $\mathbf{s}_{t+1,i} = f_i(\mathbf{s}_{t,i}, \mathbf{a}_t)$

In general, for candidate action sequence $\mathbf{a}_1, \dots, \mathbf{a}_H$:

distribution over
deterministic models

Step 1: sample $\theta \sim p(\theta|\mathcal{D})$

Step 2: at each time step t , sample $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$

Step 3: calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

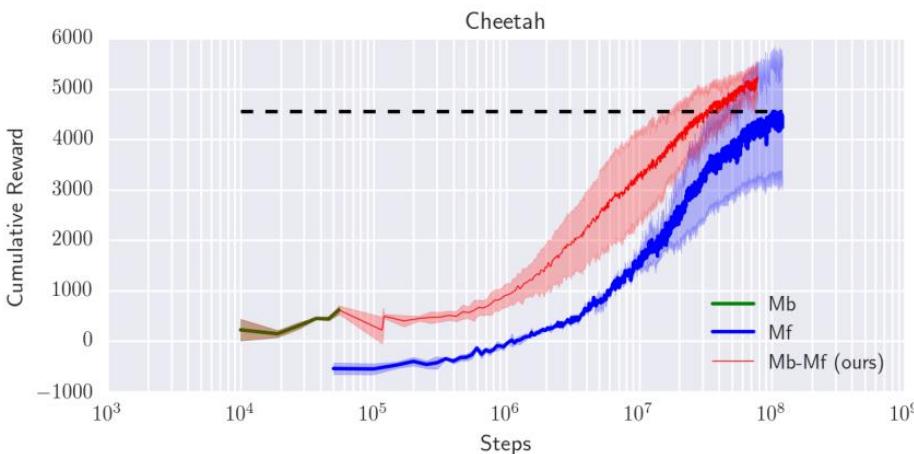
Step 4: repeat steps 1 to 3 and accumulate the average reward

Other options: moment matching, more complex posterior estimation
with BNNs, etc.

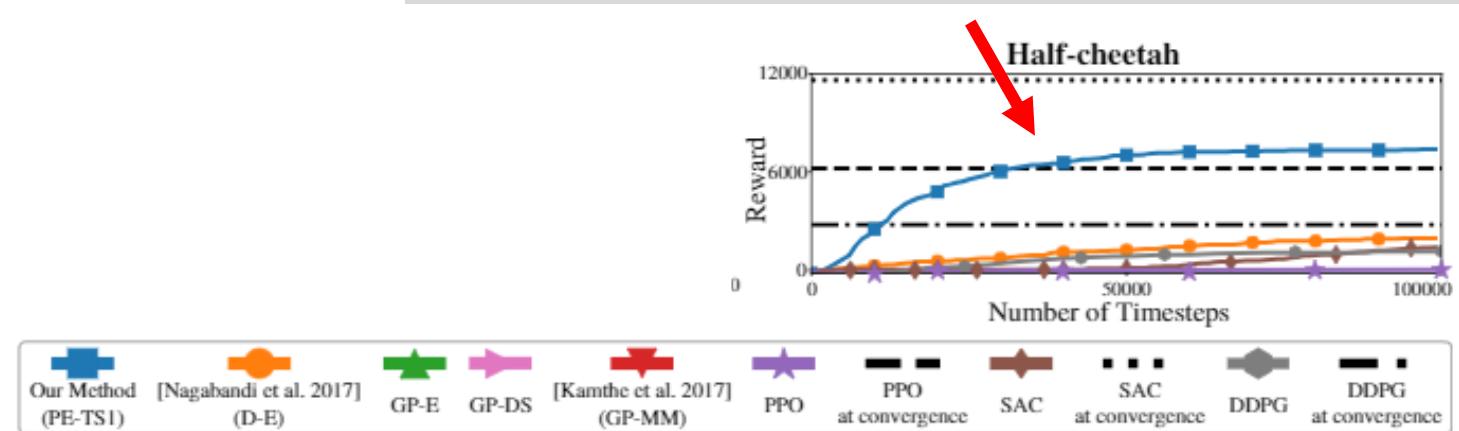
Example: model-based RL with ensembles

Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

exceeds performance of model-free after 40k steps
(about 10 minutes of real time)



before



after

More recent example: PDDM



Further readings

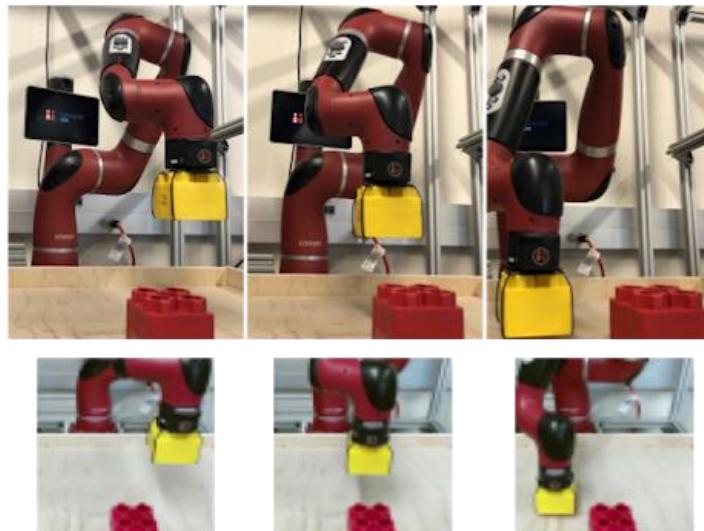
- Deisenroth et al. PILCO: A Model-Based and Data-Efficient Approach to Policy Search.

Recent papers:

- Nagabandi et al. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning.
- Chua et al. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models.
- Feinberg et al. Model-Based Value Expansion for Efficient Model-Free Reinforcement Learning.
- Buckman et al. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion.

Model-Based RL with Images

What about complex observations?



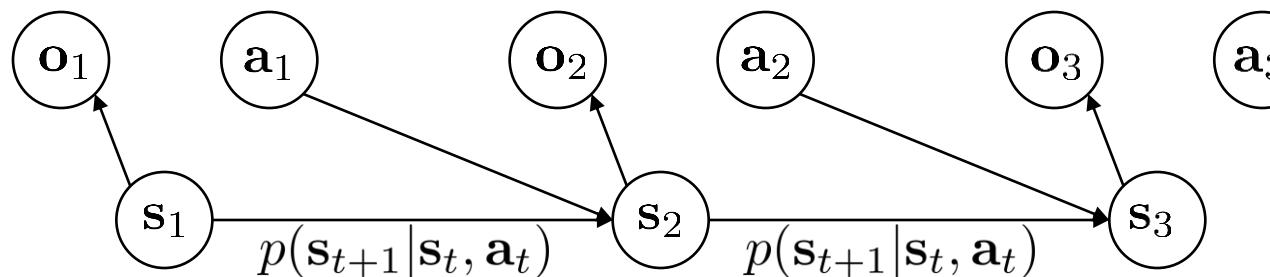
$$f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$$

What is hard about this?

- High dimensionality
- Redundancy
- Partial observability

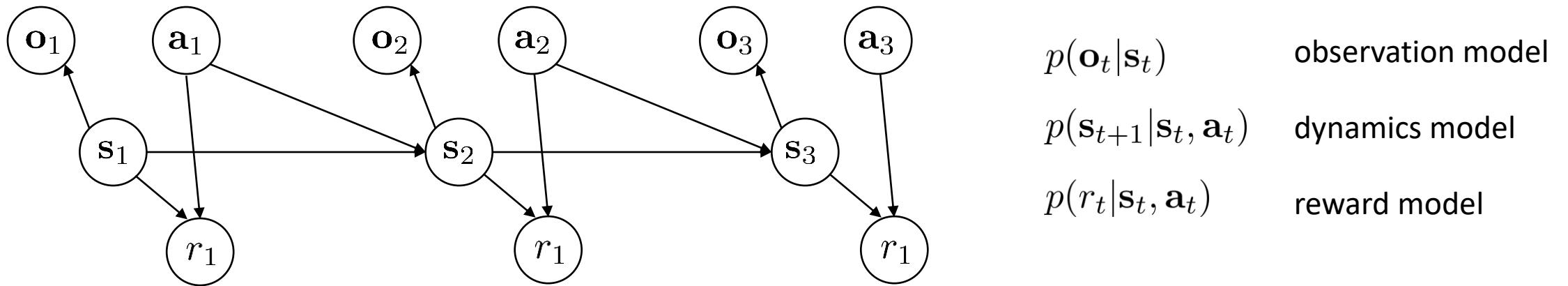
high-dimensional
but not dynamic

low-dimensional
but dynamic



separately learn $p(\mathbf{o}_t|\mathbf{s}_t)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$?

State space (latent space) models



How to train?

standard (fully observed) model: $\max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log p_{\phi}(s_{t+1,i} | s_{t,i}, \mathbf{a}_{t,i})$

latent space model: $\max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T E [\log p_{\phi}(s_{t+1,i} | s_{t,i}, \mathbf{a}_{t,i}) + \log p_{\phi}(\mathbf{o}_{t,i} | s_{t,i})]$

expectation w.r.t. $(\mathbf{s}_t, \mathbf{s}_{t+1}) \sim p(\mathbf{s}_t, \mathbf{s}_{t+1} | \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$

Model-based RL with latent space models

$$\max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T E [\log p_{\phi}(\mathbf{s}_{t+1,i} | \mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_{\phi}(\mathbf{o}_{t,i} | \mathbf{s}_{t,i})]$$

↑
expectation w.r.t. $(\mathbf{s}_t, \mathbf{s}_{t+1}) \sim p(\mathbf{s}_t, \mathbf{s}_{t+1} | \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$

learn *approximate* posterior $q_{\psi}(\mathbf{s}_t | \mathbf{o}_{1:t}, \mathbf{a}_{1:t})$ “encoder”

many other choices for approximate posterior:

$q_{\psi}(\mathbf{s}_t, \mathbf{s}_{t+1} | \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$ full smoothing posterior

+ most accurate
- most complicated

$q_{\psi}(\mathbf{s}_t | \mathbf{o}_t)$ single-step encoder

+ simplest
- least accurate

we'll talk about this one for now

We will discuss variational inference in more detail
next week!

Model-based RL with latent space models

$$\max_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T E [\log p_{\phi}(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_{\phi}(\mathbf{o}_{t,i}|\mathbf{s}_{t,i})]$$

↑
expectation w.r.t. $\mathbf{s}_t \sim q_{\psi}(\mathbf{s}_t|\mathbf{o}_t), \mathbf{s}_{t+1} \sim q_{\psi}(\mathbf{s}_{t+1}|\mathbf{o}_{t+1})$

$$q_{\psi}(\mathbf{s}_t|\mathbf{o}_t)$$

simple special case: $q(\mathbf{s}_t|\mathbf{o}_t)$ is *deterministic*

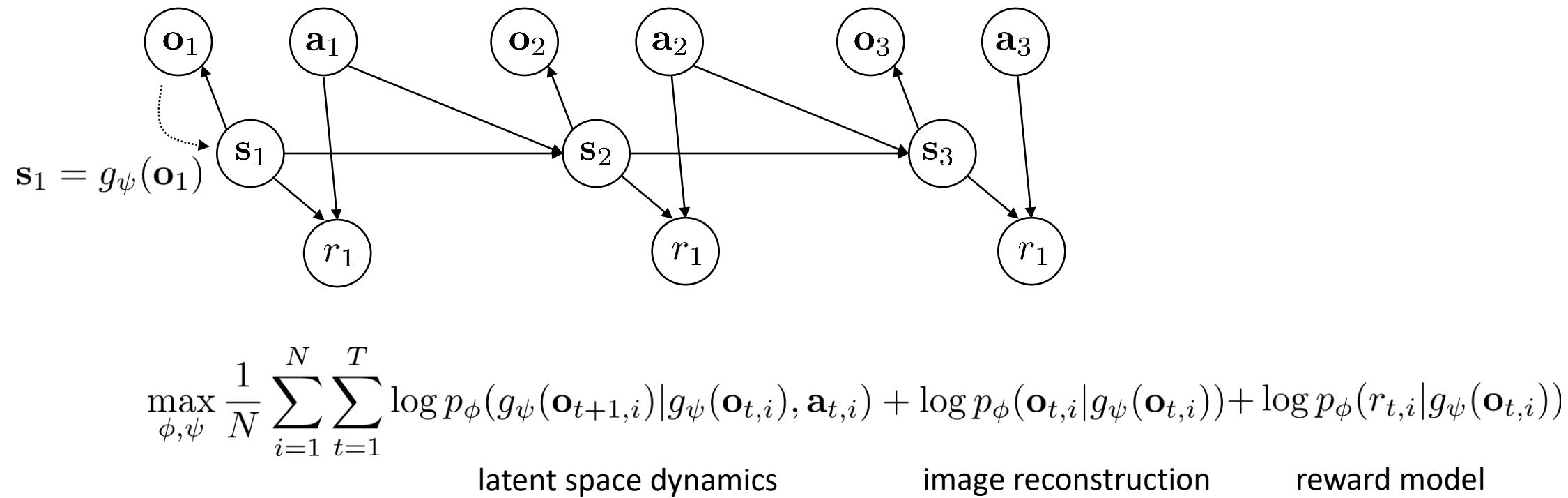
stochastic case requires variational inference (next week)

$$q_{\psi}(\mathbf{s}_t|\mathbf{o}_t) = \delta(\mathbf{s}_t = g_{\psi}(\mathbf{o}_t)) \Rightarrow \mathbf{s}_t = g_{\psi}(\mathbf{o}_t) \quad \text{deterministic encoder}$$

$$\max_{\phi, \psi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log p_{\phi}(g_{\psi}(\mathbf{o}_{t+1,i})|g_{\psi}(\mathbf{o}_{t,i}), \mathbf{a}_{t,i}) + \log p_{\phi}(\mathbf{o}_{t,i}|g_{\psi}(\mathbf{o}_{t,i}))$$

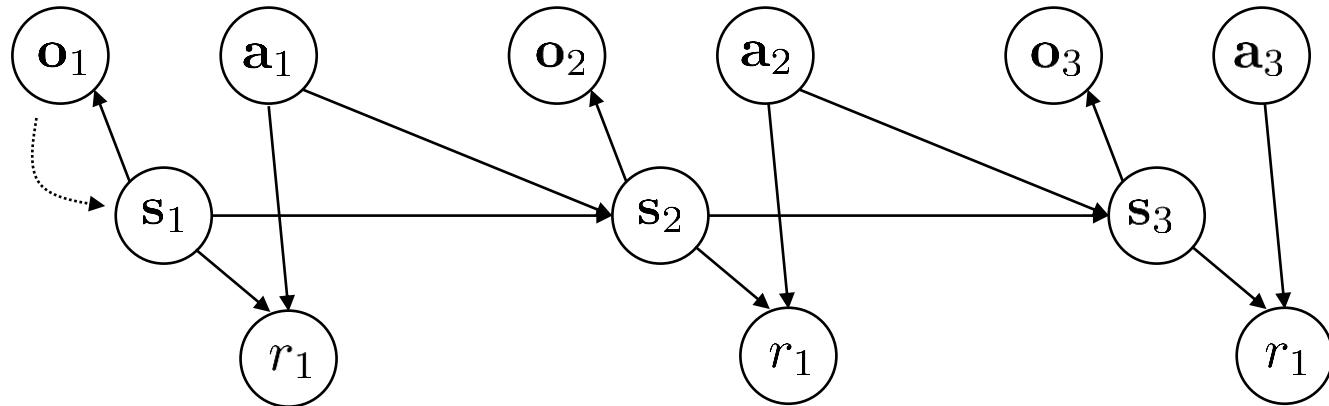
Everything is differentiable, can train with backprop

Model-based RL with latent space models



Many practical methods use a stochastic encoder to model uncertainty

Model-based RL with latent space models



model-based reinforcement learning with latent state:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{o}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{o}, \mathbf{a}, \mathbf{o}')_i\}$
2. learn $p_\phi(s_{t+1}|s_t, a_t)$, $p_\phi(r_t|s_t)$, $p(\mathbf{o}_t|s_t)$, $g_\psi(\mathbf{o}_t)$
3. plan through the model to choose actions
4. execute the first planned action, observe resulting \mathbf{o}' (MPC)
5. append $(\mathbf{o}, \mathbf{a}, \mathbf{o}')$ to dataset \mathcal{D}



Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images

Manuel Watter*

Jost Tobias Springenberg*

Joschka Boedecker

University of Freiburg, Germany

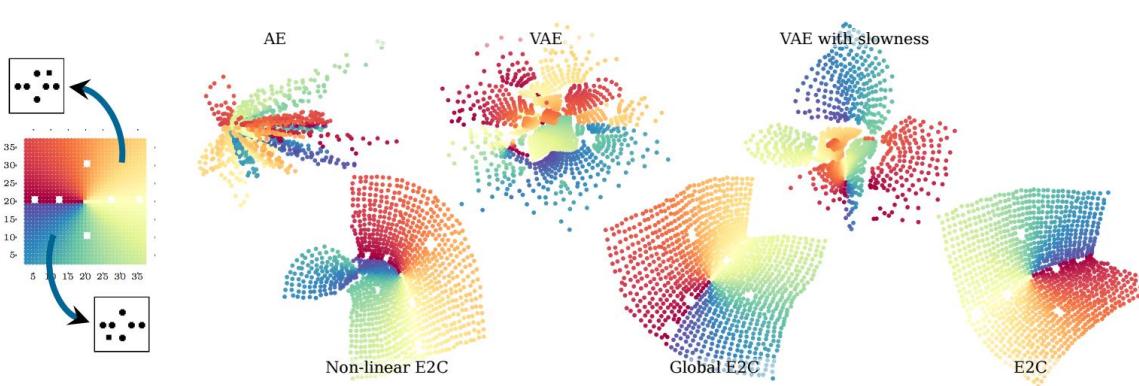
{watterm, springj, jboedeck}@cs.uni-freiburg.de

Martin Riedmiller

Google DeepMind

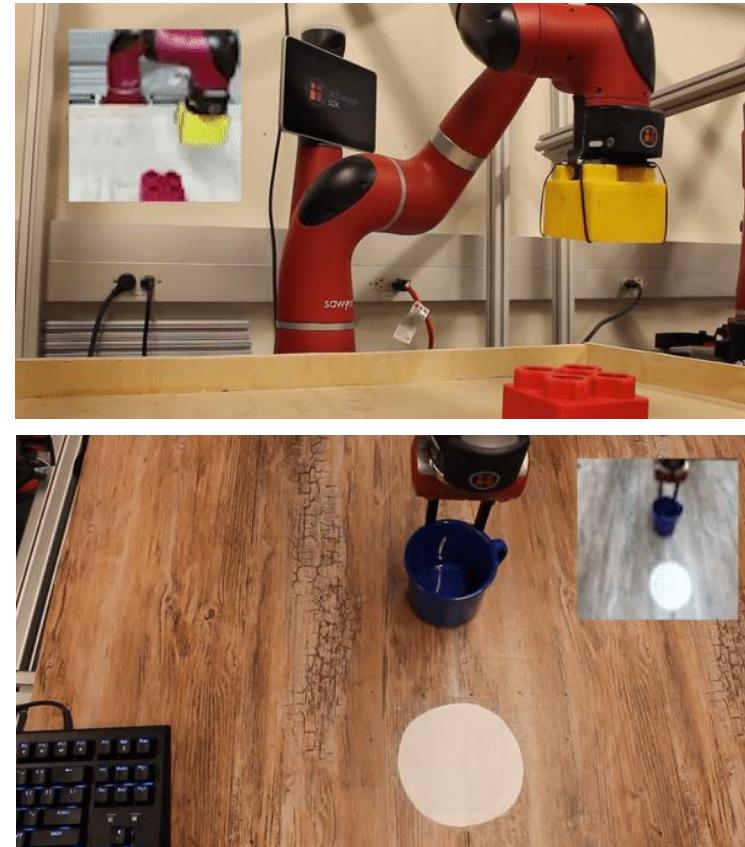
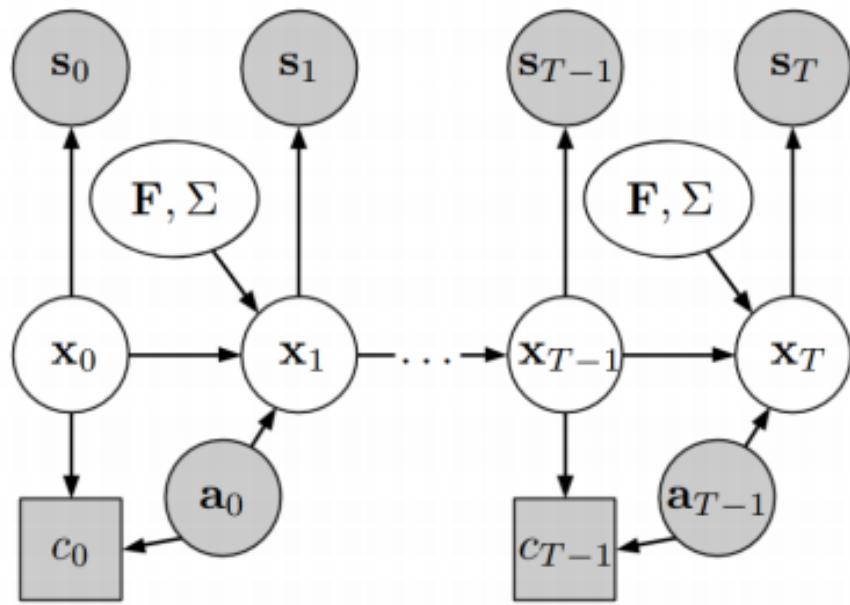
London, UK

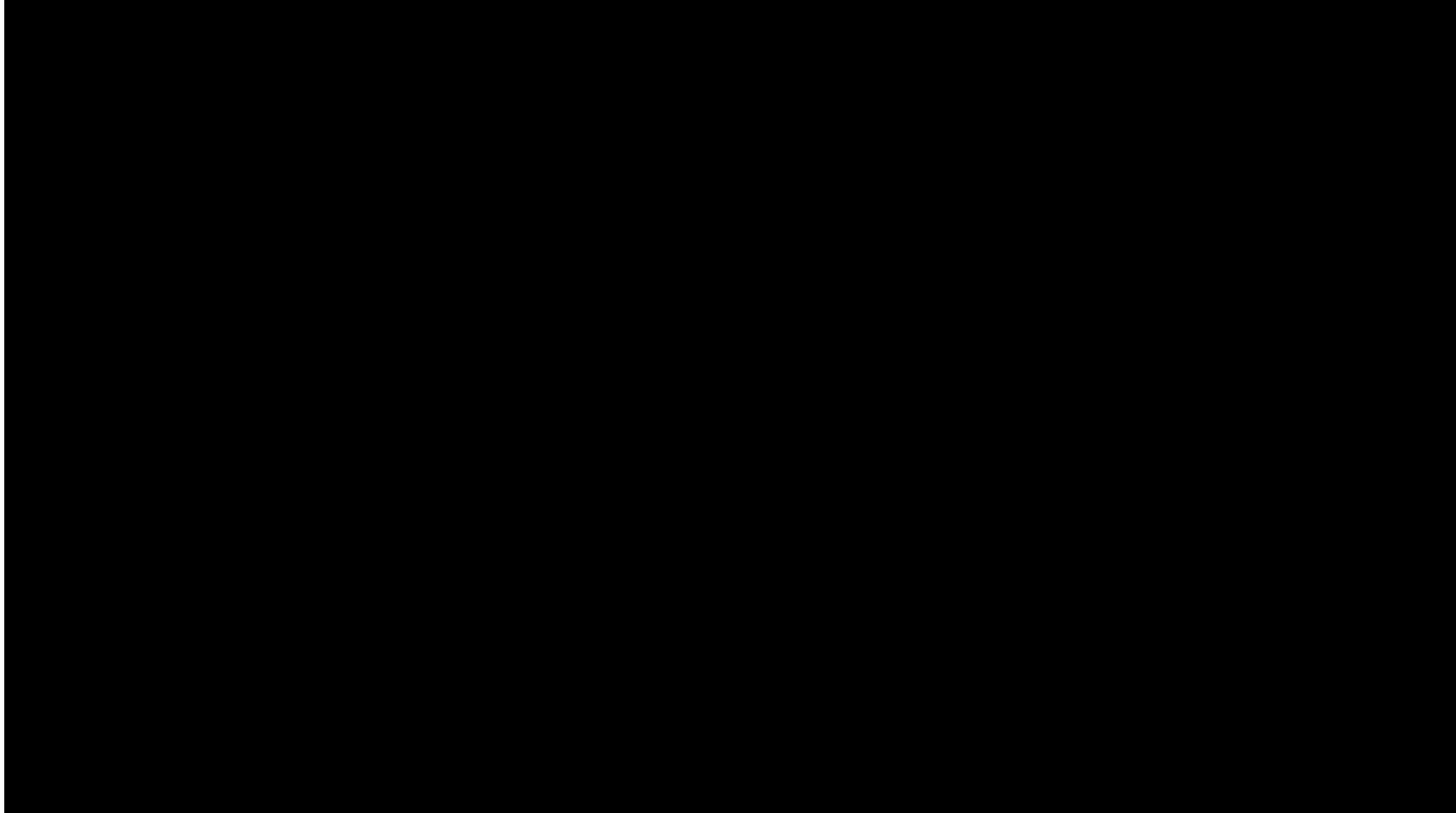
riedmiller@google.com



Swing-up with the E2C algorithm

SOLAR: Deep Structured Latent Representations for Model-Based Reinforcement Learning

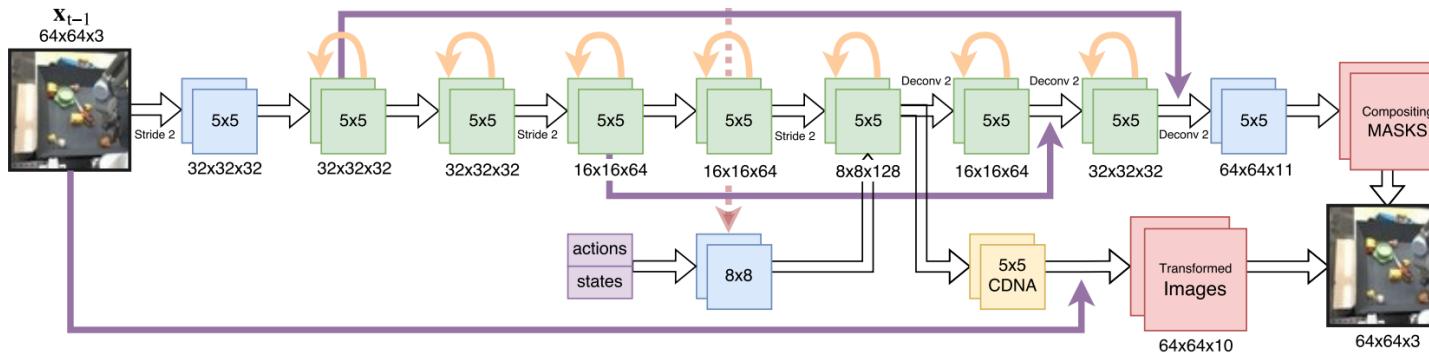




Learn directly in observation space

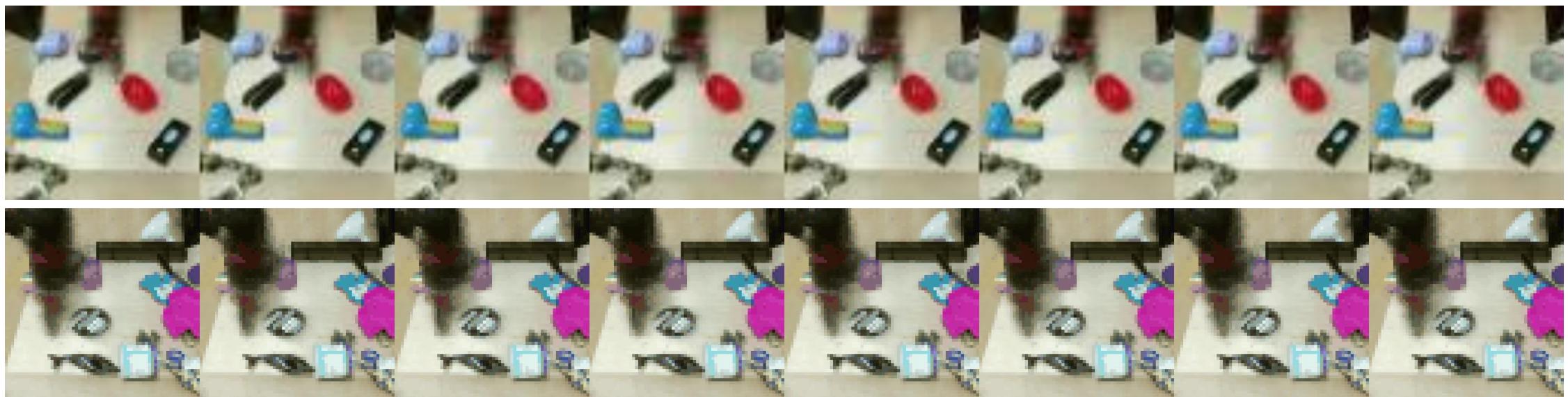
~~Key idea: learn embedding $g(\mathbf{o}_t) = s_t$~~

directly learn $p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t)$

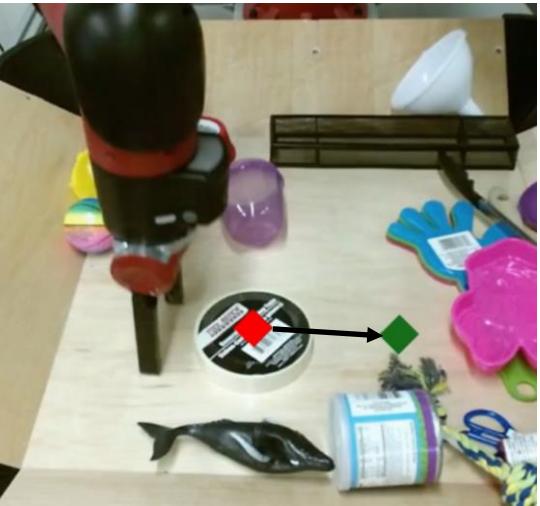


Finn, L. Deep Visual Foresight for Planning Robot Motion. ICRA 2017.

Ebert, Finn, Lee, L. Self-Supervised Visual Planning with Temporal Skip Connections. CoRL 2017.



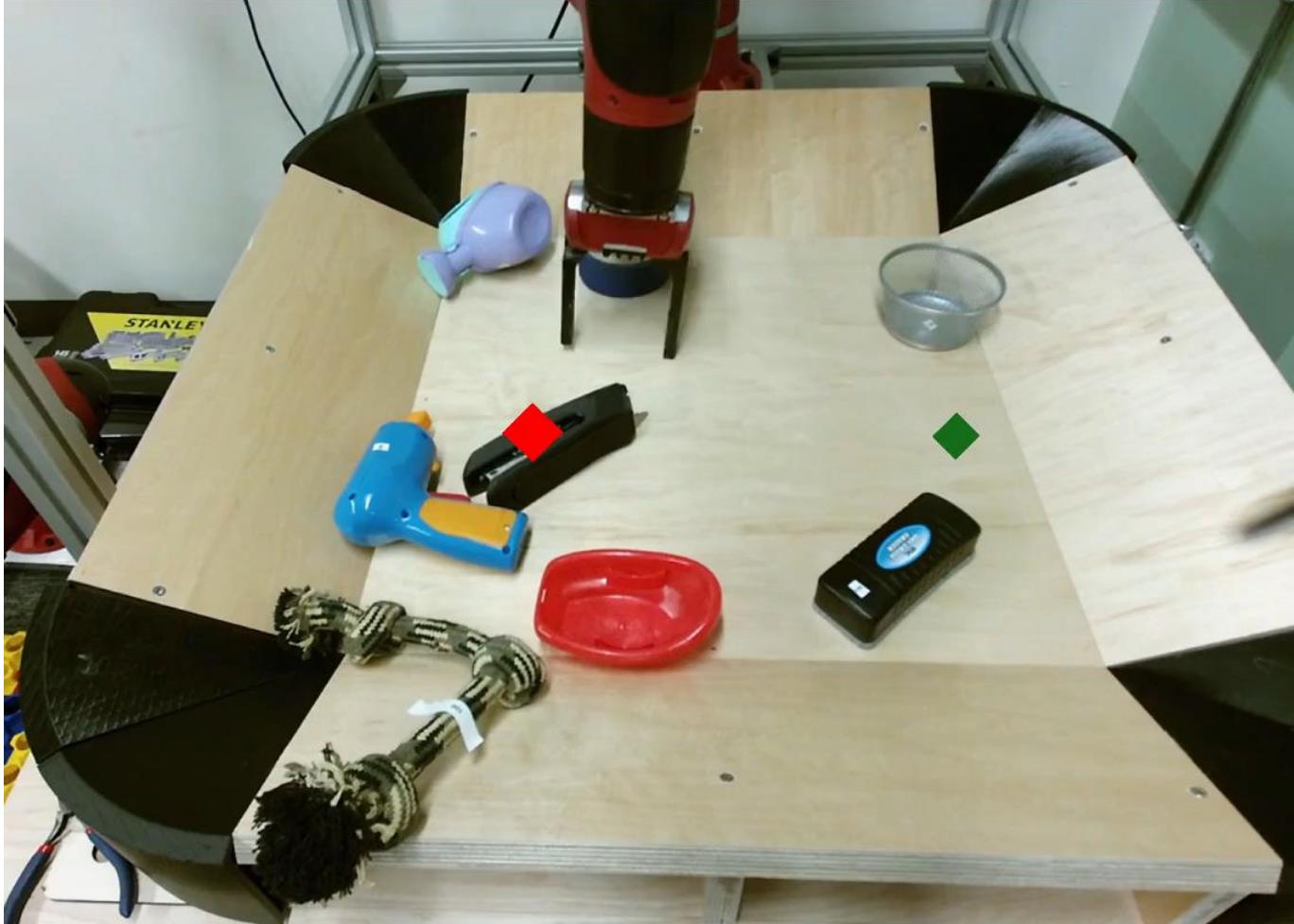
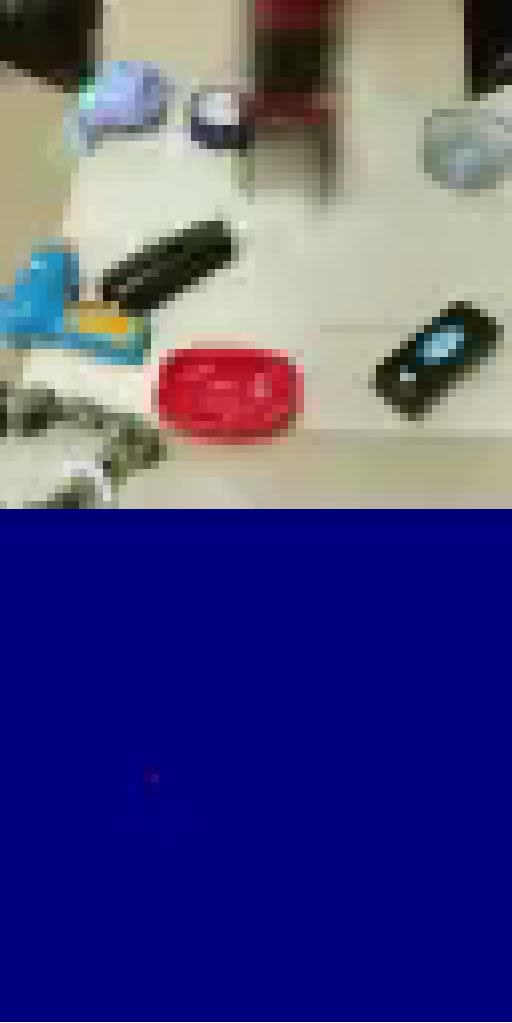
Use predictions to complete tasks



Designated Pixel ◆
Goal Pixel ◇



Task execution



Model-Based Policy Learning

CS 285

Instructor: Sergey Levine
UC Berkeley



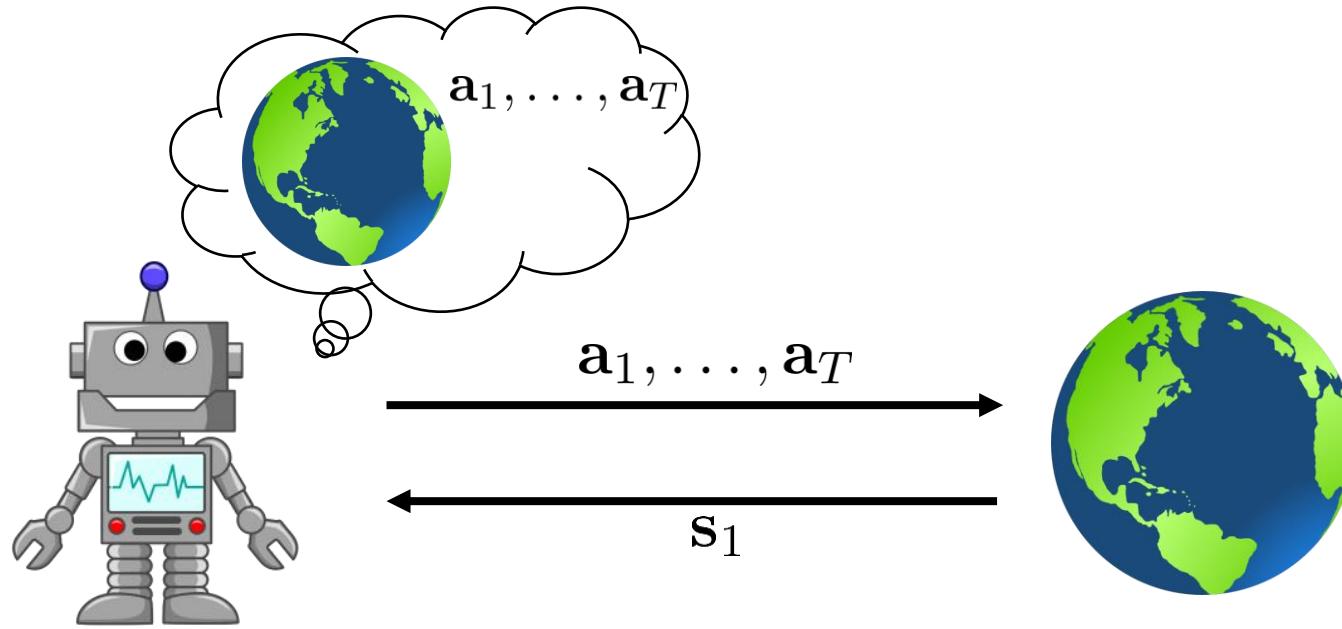
Last time: model-based RL with MPC

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state \mathbf{s}' (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset \mathcal{D}



The stochastic open-loop case

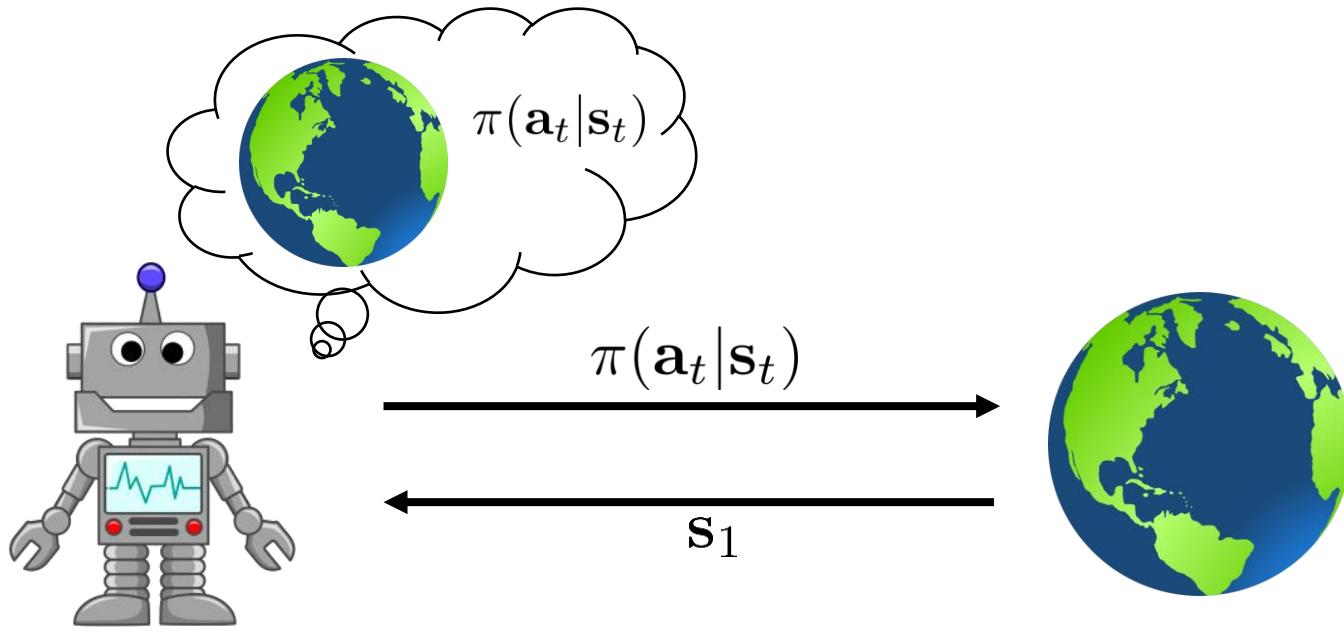


$$p_{\theta}(\mathbf{s}_1, \dots, \mathbf{s}_T | \mathbf{a}_1, \dots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} E \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \dots, \mathbf{a}_T \right]$$

why is this suboptimal?

The stochastic closed-loop case



$$p(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \pi(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg \max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

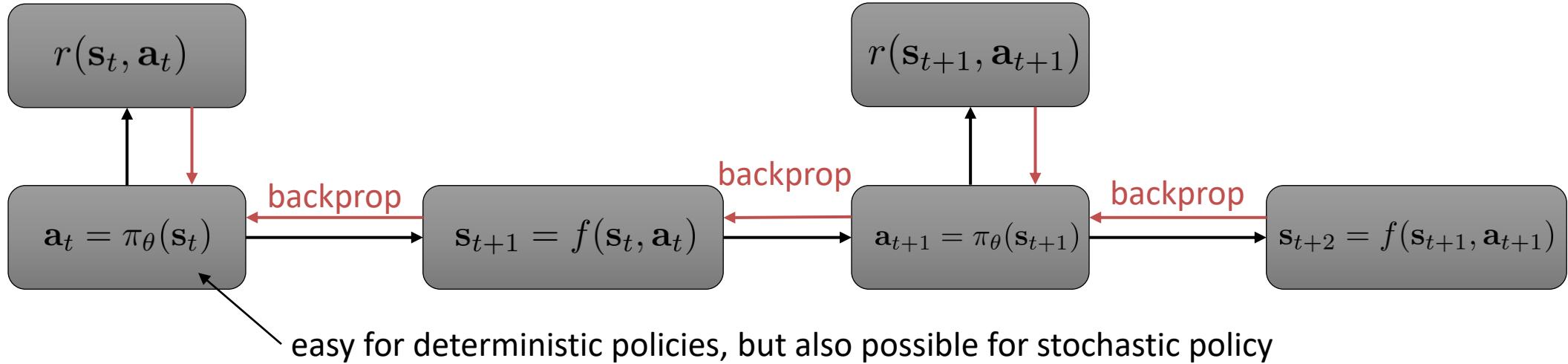
form of π ?

neural net

time-varying linear
 $\mathbf{K}_t \mathbf{s}_t + \mathbf{k}_t$

global
local

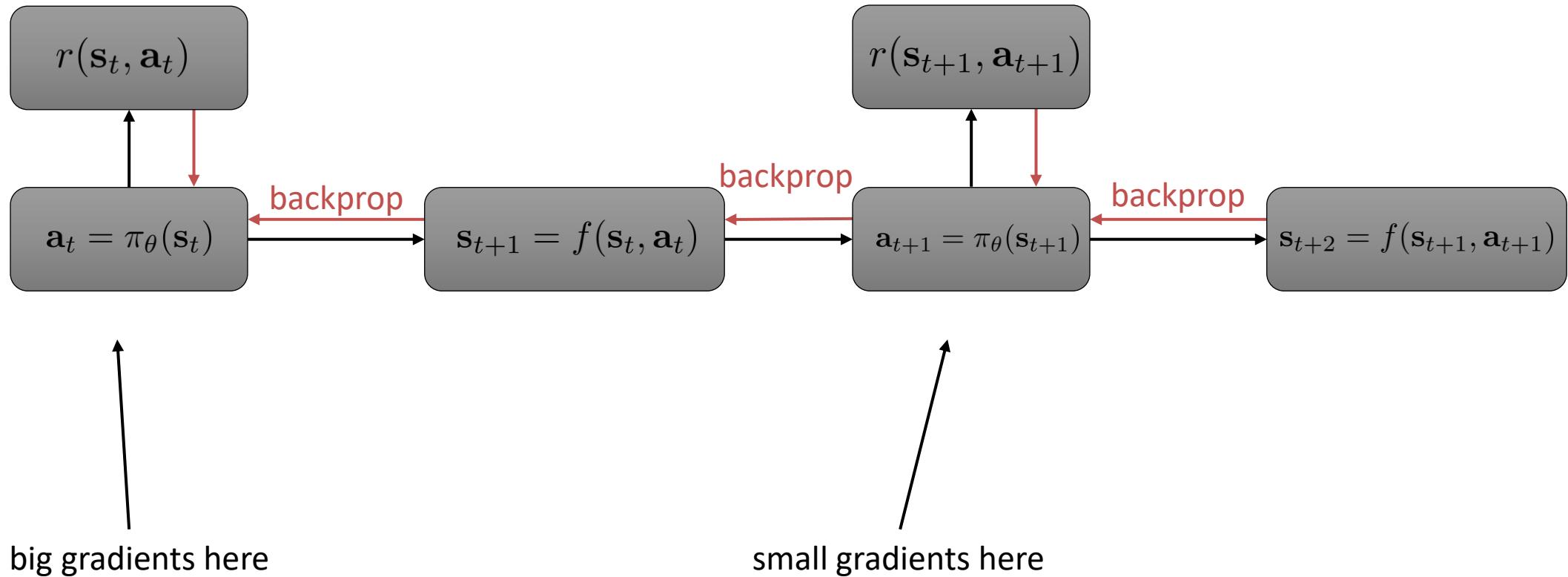
Backpropagate directly into the policy?



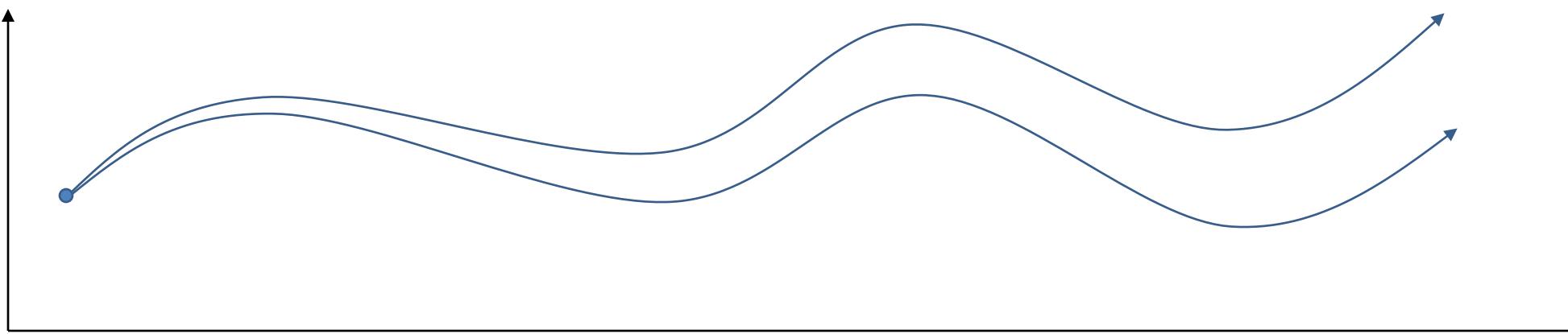
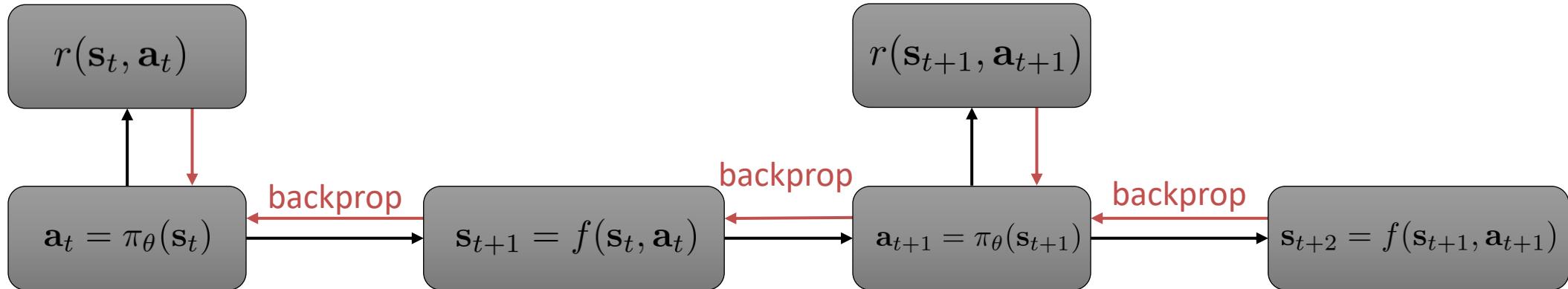
model-based reinforcement learning version 2.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')\}_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. backpropagate through $f(\mathbf{s}, \mathbf{a})$ into the policy to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to \mathcal{D}

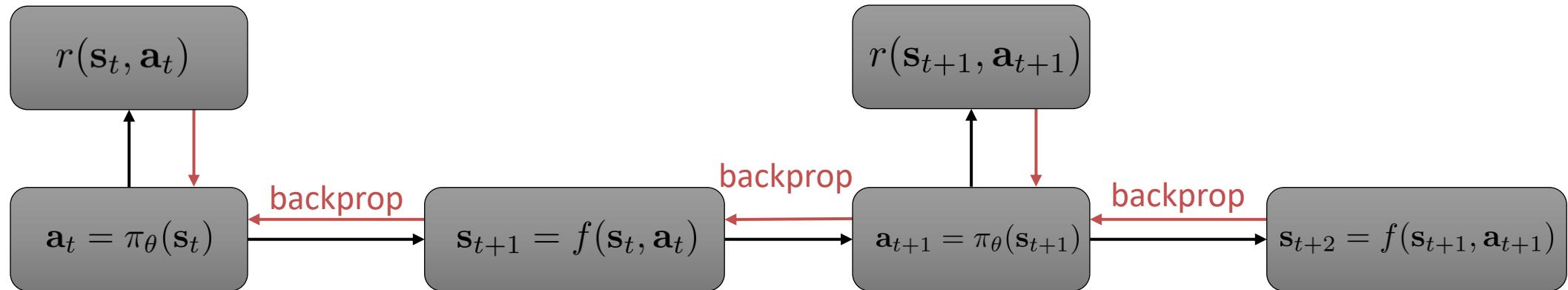
What's the problem with backprop into policy?



What's the problem with backprop into policy?



What's the problem with backprop into policy?



- Similar parameter sensitivity problems as shooting methods
 - But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming
- Similar problems to training long RNNs with BPTT
 - Vanishing and exploding gradients
 - Unlike LSTM, we can't just "choose" a simple dynamics, dynamics are chosen by nature

What's the solution?

- Use derivative-free (“model-free”) RL algorithms, with the model used to generate synthetic samples
 - Seems weirdly backwards
 - Actually works very well
 - Essentially “model-based acceleration” for model-free RL

Model-Free Learning With a Model

Model-free optimization with a model

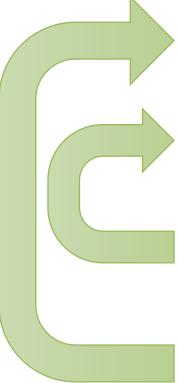
Policy gradient: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}^{\pi}$

Backprop (pathwise) gradient: $\nabla_{\theta} J(\theta) = \sum_{t=1}^T \frac{d\mathbf{a}_t}{d\theta} \frac{d\mathbf{s}_{t+1}}{d\mathbf{a}_t} \left(\sum_{t'=t+1}^T \frac{dr_{t'}}{d\mathbf{s}_{t'}} \left(\prod_{t''=t+2}^{t'} \frac{d\mathbf{s}_{t''}}{d\mathbf{a}_{t''-1}} \frac{d\mathbf{a}_{t''-1}}{d\mathbf{s}_{t''-1}} + \frac{d\mathbf{s}_{t''}}{d\mathbf{s}_{t''-1}} \right) \right)$

- Policy gradient might be more *stable* (if enough samples are used) because it does not require multiplying many Jacobians
- See a recent analysis here:
 - Parmas et al. '18: PIPP: Flexible Model-Based Policy Search Robust to the Curse of Chaos

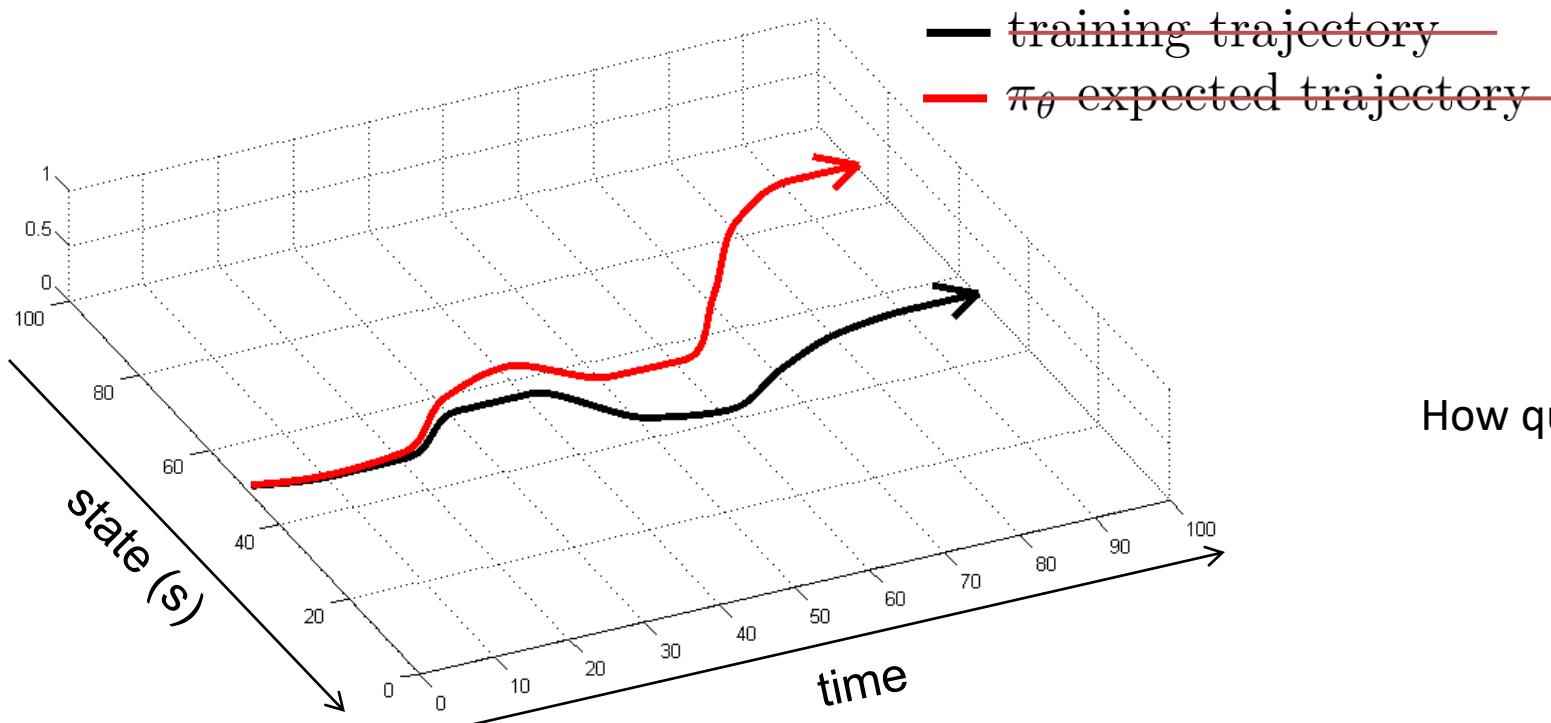
Model-based RL via policy gradient

model-based reinforcement learning version 2.5:

- 
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
 2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
 3. use $f(\mathbf{s}, \mathbf{a})$ to generate trajectories $\{\tau_i\}$ with policy $\pi_\theta(\mathbf{a}|\mathbf{s})$
 4. use $\{\tau_i\}$ to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ via policy gradient
 5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to \mathcal{D}

What's a potential **problem** with this approach?

The curse of long model-based rollouts

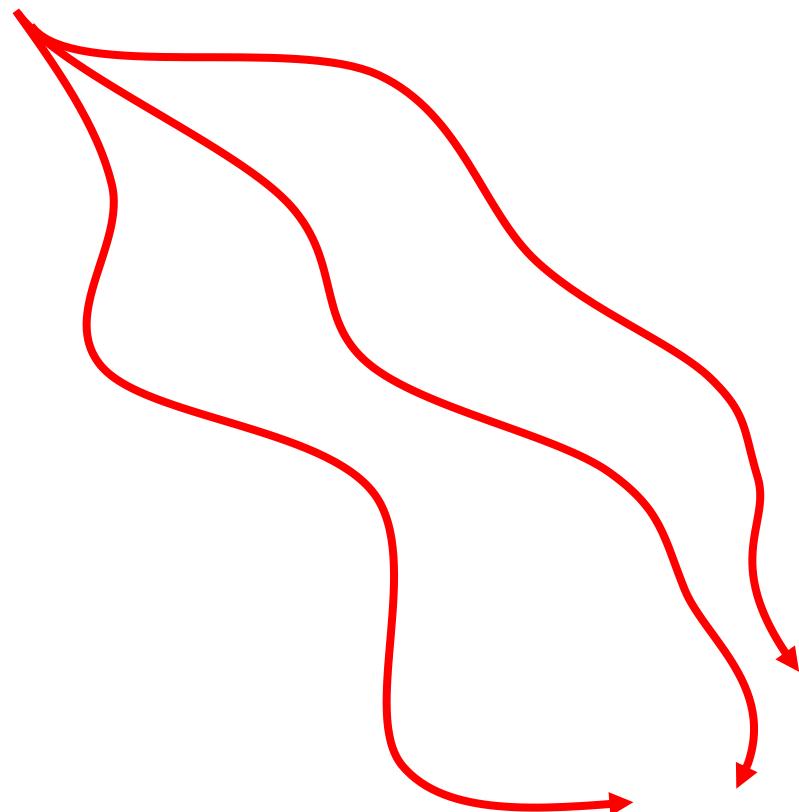


run π_θ with true dynamics
run π_θ with learned model

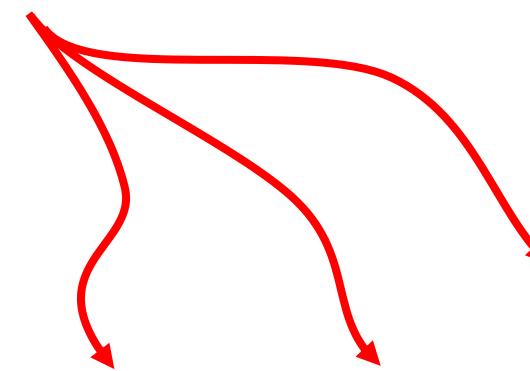
How quickly does error accumulate?

$$\mathcal{O}(\epsilon T^2)$$

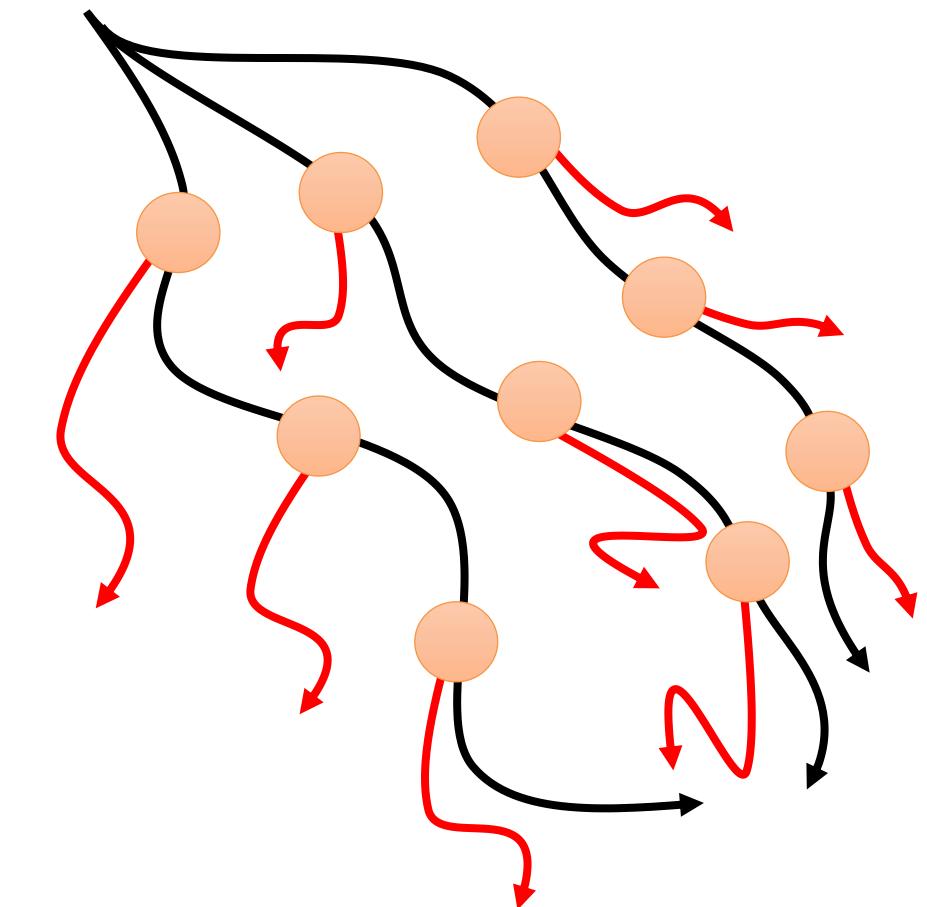
How to get away with short rollouts?



- huge accumulating error



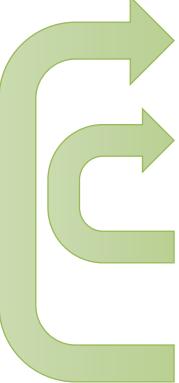
+ much lower error
- never see later time steps



+ much lower error
+ see all time steps
- wrong state distribution

Model-based RL with short rollouts

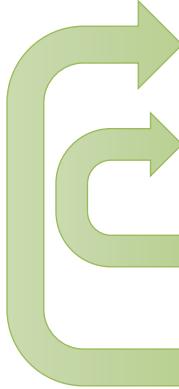
model-based reinforcement learning version 3.0:

- 
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')\}_i$
 2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
 3. pick states \mathbf{s}_i from \mathcal{D} , use $f(\mathbf{s}, \mathbf{a})$ to make *short* rollouts from them
 4. use *both* real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with *off-policy RL*
 5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to \mathcal{D}

Dyna-Style Algorithms

Model-based RL with short rollouts

model-based reinforcement learning version 3.0:

- 
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')\}_i$
 2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
 3. pick states \mathbf{s}_i from \mathcal{D} , use $f(\mathbf{s}, \mathbf{a})$ to make *short* rollouts from them
 4. use *both* real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with *off-policy RL*
 5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to \mathcal{D}

Model-free optimization with a model

Dyna

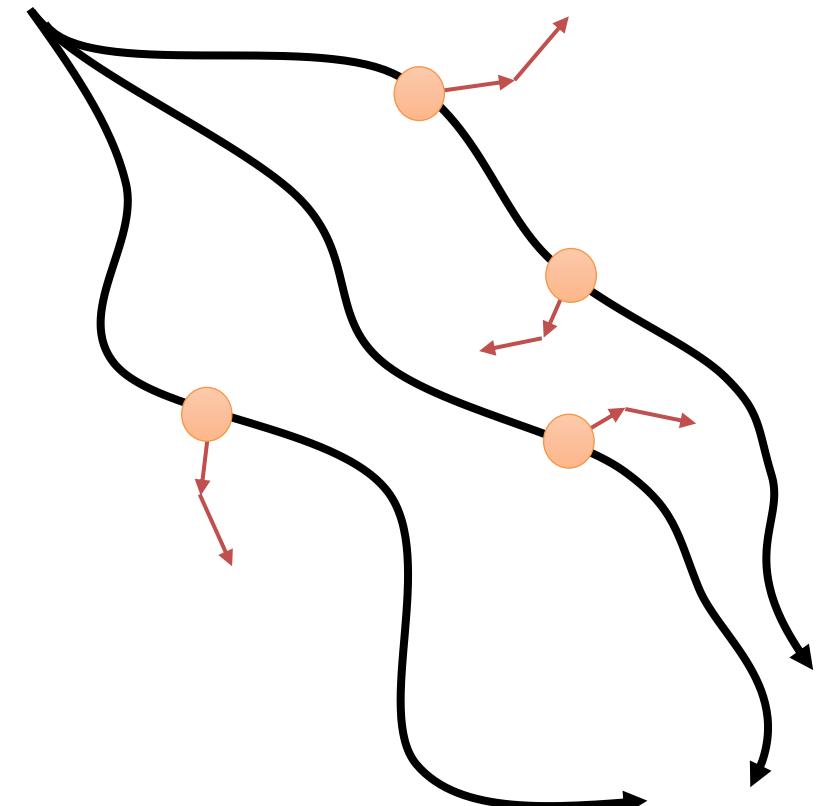
online Q-learning algorithm that performs model-free RL with a model

1. given state s , pick action a using exploration policy
2. observe s' and r , to get transition (s, a, s', r)
3. update model $\hat{p}(s'|s, a)$ and $\hat{r}(s, a)$ using (s, a, s')
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. repeat K times:
 6. sample $(s, a) \sim \mathcal{B}$ from buffer of past states and actions
 7. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

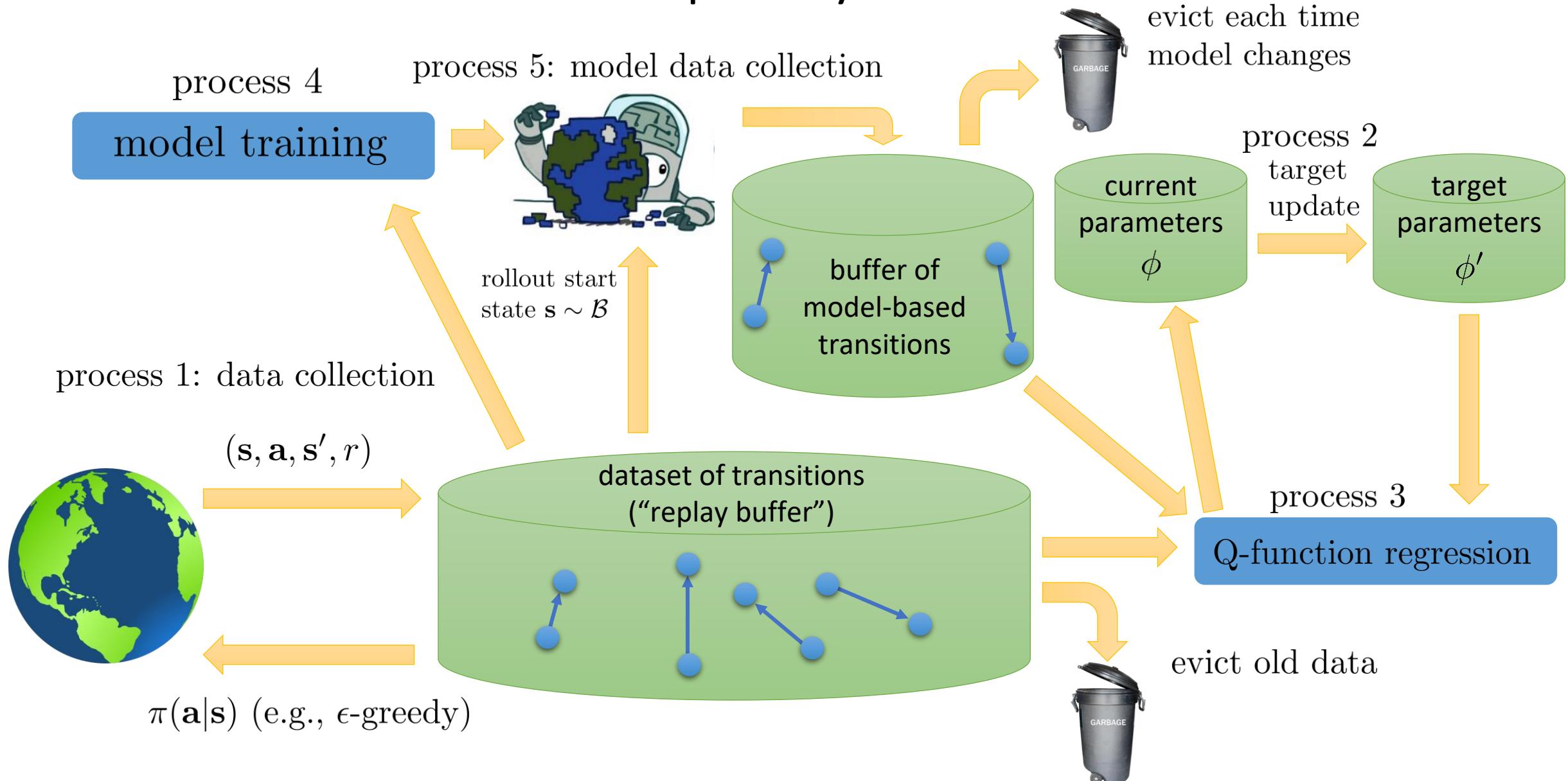
General “Dyna-style” model-based RL recipe

1. collect some data, consisting of transitions (s, a, s', r)
2. learn model $\hat{p}(s'|s, a)$ (and optionally, $\hat{r}(s, a)$)
3. repeat K times:
 4. sample $s \sim \mathcal{B}$ from buffer
 5. choose action a (from \mathcal{B} , from π , or random)
 6. simulate $s' \sim \hat{p}(s'|s, a)$ (and $r = \hat{r}(s, a)$)
 7. train on (s, a, s', r) with model-free RL
 8. (optional) take N more model-based steps

+ only requires short (as few as one step) rollouts from model
+ still sees diverse states



Model-accelerated off-policy RL



Model-Based Acceleration (MBA)

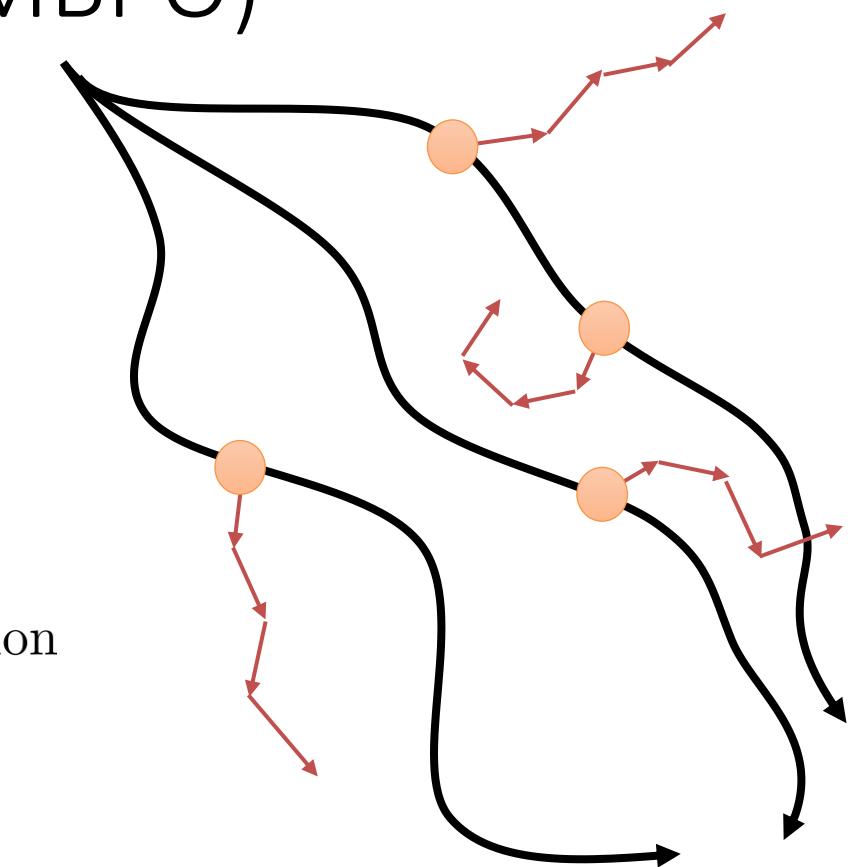
Model-Based Value Expansion (MVE)

Model-Based Policy Optimization (MBPO)

1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
3. use $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j\}$ to update model $\hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$
4. sample $\{\mathbf{s}_j\}$ from \mathcal{B}
5. for each \mathbf{s}_j , perform model-based rollout with $\mathbf{a} = \pi(\mathbf{s})$
6. use all transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ along rollout to update Q-function

+ why is this a *good* idea?

- why is this a *bad* idea?



Gu et al. Continuous deep Q-learning with model-based acceleration. '16

Feinberg et al. Model-based value expansion. '18

Janner et al. When to trust your model: model-based policy optimization. '19

Multi-Step Models & Successor Representations

What kind of model do we need to **evaluate** a policy?

The job of the model is to **evaluate** the policy

$$J(\pi) = E_{s \sim p(s_1)}[V^\pi(s_1)]$$

$$V^\pi(s_t) = \sum_{t=t'}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)} E_{\mathbf{a}_{t'} \sim \pi(\mathbf{a}_{t'}|s_{t'})} [r(\mathbf{s}_{t'}, \mathbf{a}_{t'})]$$

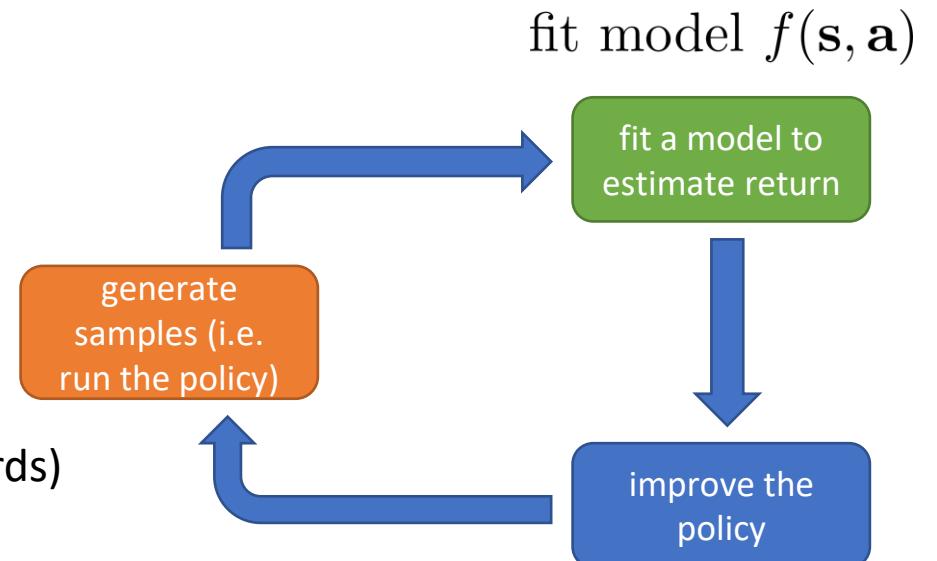
let's keep it simple

$$= \sum_{t=t'}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)} [r(\mathbf{s}_{t'})] \quad (\text{easy to re-derive for action-dependent rewards})$$

$$= \sum_{t=t'}^{\infty} \gamma^{t'-t} \sum_{\mathbf{s}} p(\mathbf{s}_{t'} = \mathbf{s} | \mathbf{s}_t) r(\mathbf{s})$$

$$= \sum_{\mathbf{s}} \left(\sum_{t=t'}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s} | \mathbf{s}_t) \right) r(\mathbf{s})$$

(if you can evaluate it, you can make it better)



What kind of model do we need to evaluate a policy?

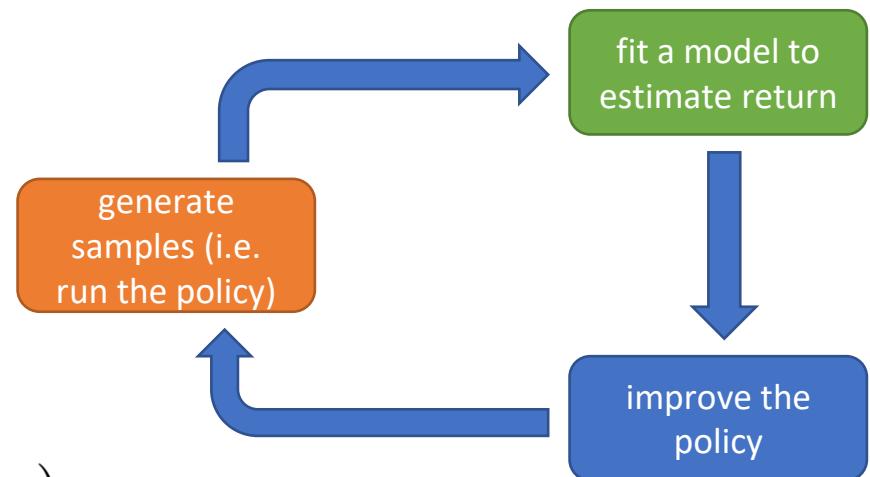
$$V^\pi(\mathbf{s}_t) = \sum_{t=t'}^{\infty} \gamma^{t'-t} E_{p(s_{t'}|s_t)}[r(\mathbf{s}_{t'})]$$

$$= \sum_{\mathbf{s}} \left(\underbrace{\sum_{t=t'}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s} | s_t)}_{p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s} | \mathbf{s}_t)} \right) r(\mathbf{s})$$

$$p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s} | \mathbf{s}_t) = \underbrace{(1 - \gamma)}_{\text{just to ensure it sums to 1}} \sum_{t'=t}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s} | \mathbf{s}_t)$$

(if you can evaluate it, you can make it better)

fit model $f(\mathbf{s}, \mathbf{a})$



What kind of model do we need to evaluate a policy?

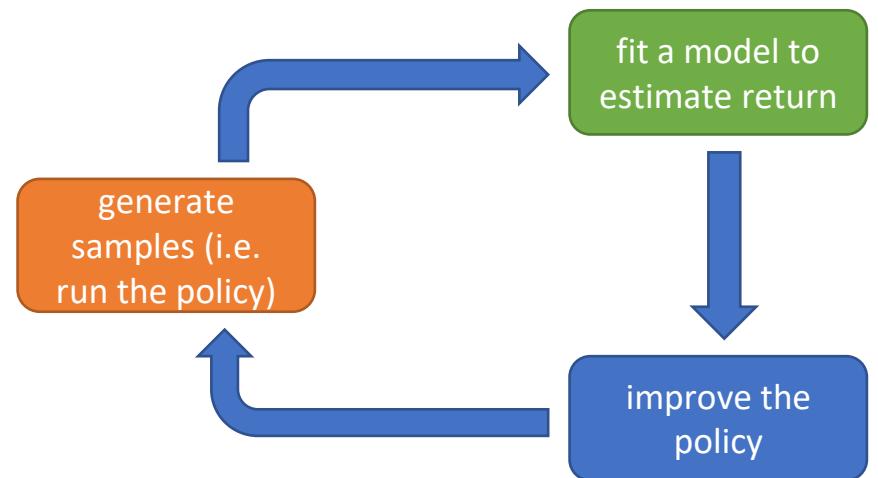
$$p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s} | \mathbf{s}_t) = (1 - \gamma) \sum_{t'=t}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = \mathbf{s} | \mathbf{s}_t)$$

(if you can evaluate it, you can make it better)

fit model $f(\mathbf{s}, \mathbf{a})$

$$V^\pi(\mathbf{s}_t) = \frac{1}{1 - \gamma} \underbrace{\sum_{\mathbf{s}} p_\pi(\mathbf{s}_{\text{future}} = \mathbf{s} | \mathbf{s}_t) r(\mathbf{s})}_{\mu^\pi(\mathbf{s}_t)^T \vec{r}}$$

$$\mu_i^\pi(\mathbf{s}_t) = p_\pi(s_{\text{future}} = i | \mathbf{s}_t)$$



This is called a **successor representation**

Successor representations

$$\begin{aligned}\mu_i^\pi(\mathbf{s}_t) &= (1 - \gamma) \sum_{t'=t}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = i | \mathbf{s}_t) \\ &= (1 - \gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\mu_i^\pi(\mathbf{s}_{t+1})]\end{aligned}$$

like a Bellman backup with “reward” $r(\mathbf{s}_t) = (1 - \gamma)\delta(\mathbf{s}_t = i)$

in practice, we can use vectorized backups for all i at once

A few issues...

- Not clear if learning successor representation is easier than model-free RL
- How to scale to large state spaces?
- How to extend to continuous state spaces?

Successor features

$$\mu_i^\pi(\mathbf{s}_t) = (1 - \gamma) \sum_{t'=t}^{\infty} \gamma^{t'-t} p(\mathbf{s}_{t'} = i | \mathbf{s}_t) \quad \psi_j^\pi(\mathbf{s}_t) = \sum_{\mathbf{s}} \mu_{\mathbf{s}}^\pi(\mathbf{s}_t) \phi_j(\mathbf{s}) \quad \psi_j^\pi(\mathbf{s}_t) = \mu^\pi(\mathbf{s}_t)^T \vec{\phi}_j$$

$$V^\pi(\mathbf{s}_t) = \mu^\pi(\mathbf{s}_t)^T \vec{r}$$

if $r(\mathbf{s}) = \sum_j \phi_j(\mathbf{s}) w_j = \phi(\mathbf{s})^T \mathbf{w}$

then $V^\pi(\mathbf{s}_t) = \psi^\pi(\mathbf{s}_t)^T \mathbf{w}$

so what?

If the number of features is much less than the number of states, learning them is much easier!

$$\begin{aligned} &= \sum_j \psi_j^\pi(\mathbf{s}_t) w_j \\ &= \sum_j \mu^\pi(\mathbf{s}_T)^T \vec{\phi}_j \mathbf{w} \\ &= \mu^\pi(\mathbf{s}_T)^T \sum_j \vec{\phi}_j \mathbf{w} = \mu^\pi(\mathbf{s}_t)^T \vec{r} \end{aligned}$$

Successor features

$$\mu_i^\pi(\mathbf{s}_t) = (1 - \gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\mu_i^\pi(\mathbf{s}_{t+1})]$$

$$\psi_j^\pi(\mathbf{s}_t) = \phi_j(\mathbf{s}_t) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\psi_j^\pi(\mathbf{s}_{t+1})]$$

special case with
 $\phi_i(\mathbf{s}_t) = (1 - \gamma)\delta(\mathbf{s}_t = i)$

can also construct a “Q-function-like” version:

$$\psi_j^\pi(\mathbf{s}_t, \mathbf{a}_t) = \phi_j(\mathbf{s}_t) + \gamma E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_{t+1} \sim \pi(\mathbf{a}_{t+1} | \mathbf{s}_{t+1})} [\psi_j^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})]$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^\pi(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w} \quad \text{when } r(\mathbf{s}_t) \approx \phi(\mathbf{s}_t)^T \mathbf{w}$$

Using successor features

Idea 1: recover a Q-function very quickly

1. Train $\psi^\pi(\mathbf{s}_t, \mathbf{a}_t)$ (via Bellman backups)
2. Get some reward samples $\{\mathbf{s}_i, r_i\}$
3. Get $\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} \sum_i \|\phi(\mathbf{s}_i)^T \mathbf{w} - r_i\|^2$
4. Recover $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^\pi(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$

Is this the **optimal** Q-function?

$$\pi'(\mathbf{s}) = \arg \max_{\mathbf{a}} \psi^\pi(\mathbf{s}, \mathbf{a})^T \mathbf{w}$$

Equivalent to **one step** of policy iteration

Better than nothing, but **not** optimal

Using successor features

Idea 2: recover many Q-functions

1. Train $\psi^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t)$ for many policies π_k (via Bellman backups)
2. Get some reward samples $\{\mathbf{s}_i, r_i\}$
3. Get $\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} \sum_i \|\phi(\mathbf{s}_i)^T \mathbf{w} - r_i\|^2$
4. Recover $Q^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t) \approx \psi^{\pi_k}(\mathbf{s}_t, \mathbf{a}_t)^T \mathbf{w}$ for every π_k

$$\pi'(\mathbf{s}) = \arg \max_{\mathbf{a}} \max_k \psi^{\pi_k}(\mathbf{s}, \mathbf{a})^T \mathbf{w}$$

Finds the highest reward policy in **each state**

Continuous successor representations

$$\mu_i^\pi(\mathbf{s}_t) = (1 - \gamma)\delta(\mathbf{s}_t = i) + \gamma E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t), \mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\mu_i^\pi(\mathbf{s}_{t+1})]$$



always zero for any sampled state if states are continuous

Framing successor representation as *classification*:

$$p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

binary classifier

$F = 1$ means $\mathbf{s}_{\text{future}}$ is a future state from $\mathbf{s}_t, \mathbf{a}_t$ under π

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t) \quad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

Continuous successor representations

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t) \quad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

$$p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

$$p^\pi(F = 0 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(\mathbf{s}_{\text{future}})}{p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t) + p^\pi(\mathbf{s}_{\text{future}})}$$

$$\frac{p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F = 0 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})} = \frac{p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t)}{p^\pi(\mathbf{s}_{\text{future}})}$$

$$\frac{p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F = 0 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})} p^\pi(\mathbf{s}_{\text{future}}) = p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t)$$

constant independent of $\mathbf{a}_t, \mathbf{s}_t$

The C-Learning algorithm

$$\mathcal{D}_+ \sim p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t) \quad \mathcal{D}_- \sim p^\pi(\mathbf{s})$$

$$p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) = \frac{p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}})}{p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{\text{future}}) + p^\pi(\mathbf{s}_{\text{future}})}$$

To train:

1. Sample $\mathbf{s} \sim p^\pi(\mathbf{s})$ (e.g., run policy, sample from trajectories)
2. Sample $\mathbf{s} \sim p^\pi(\mathbf{s}_{\text{future}} | \mathbf{s}_t, \mathbf{a}_t)$ (e.g., pick $\mathbf{s}_{t'}$ where $t' = t + \Delta$, $\Delta \sim \text{Geom}(\gamma)$)
3. Update $p^\pi(F = 1 | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s})$ using SGD with cross entropy loss

This is an **on policy** algorithm

Could also derive an **off policy** algorithm

Exploration (Part 1)

CS 285

Instructor: Sergey Levine
UC Berkeley

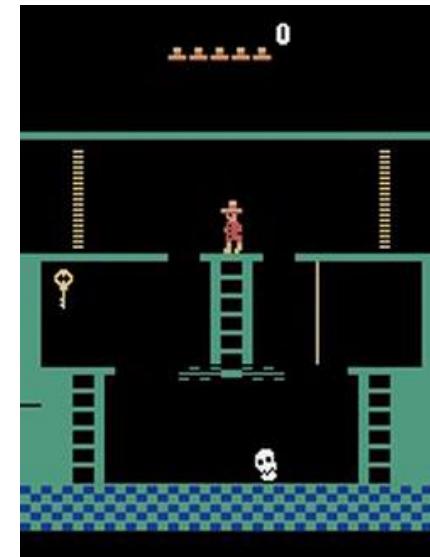


What's the problem?

this is easy (mostly)



this is impossible



Why?

Montezuma's revenge



- Getting key = reward
- Opening door = reward
- Getting killed by skull = nothing (is it good? bad?)
- Finishing the game only weakly correlates with rewarding events
- We know what to do because we **understand** what these sprites mean!

Put yourself in the algorithm's shoes



Mao

- “the only rule you may be told is this one”
- Incur a penalty when you break a rule
- Can only discover rules through trial and error
- Rules don’t always make sense to you
- Temporally extended tasks like Montezuma’s revenge become increasingly difficult based on
 - How extended the task is
 - How little you know about the rules
- Imagine if your goal in life was to win 50 games of Mao...
- (and you didn’t know this in advance)

Another example

Learned Policies

Exploration and exploitation

- Two potential definitions of exploration problem
 - How can an agent discover high-reward strategies that require a temporally extended sequence of complex behaviors that, individually, are not rewarding?
 - How can an agent decide whether to attempt new behaviors (to discover ones with higher reward) or continue to do the best thing it knows so far?
- Actually the same problem:
 - Exploitation: doing what you *know* will yield highest reward
 - Exploration: doing things you haven't done before, in the hopes of getting even higher reward

Exploration and exploitation examples

- Restaurant selection
 - **Exploitation:** go to your favorite restaurant
 - **Exploration:** try a new restaurant
- Online ad placement
 - **Exploitation:** show the most successful advertisement
 - **Exploration:** show a different random advertisement
- Oil drilling
 - **Exploitation:** drill at the best known location
 - **Exploration:** drill at a new location

Exploration is hard

Can we derive an **optimal** exploration strategy?

what does optimal even mean?

regret vs. Bayes-optimal strategy? more on this later...

multi-armed bandits
(1-step stateless
RL problems)

contextual bandits
(1-step RL problems)

small, finite MDPs
(e.g., tractable planning,
model-based RL setting)

large, infinite MDPs,
continuous spaces

theoretically tractable

theoretically intractable

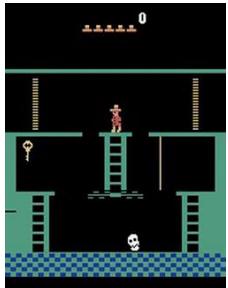
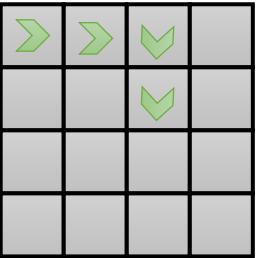
What makes an exploration problem tractable?

multi-arm bandits

contextual bandits

small, finite MDPs

large or infinite MDPs



can formalize exploration
as POMDP identification
policy learning is trivial
even with POMDP

can frame as Bayesian model
identification, reason explicitly
about value of information

optimal methods don't work
...but can take inspiration from
optimal methods in smaller settings
use hacks

Bandits

What's a bandit anyway?



the drosophila of exploration problems



$$\mathcal{A} = \{\text{pull arm}\}$$

$$r(\text{pull arm}) = ?$$



$$\mathcal{A} = \{\text{pull}_1, \text{pull}_2, \dots, \text{pull}_n\}$$

$$r(a_n) = ?$$

assume $r(a_n) \sim p(r|a_n)$

unknown *per-action* reward distribution!

How can we define the bandit?

assume $r(a_i) \sim p_{\theta_i}(r_i)$

e.g., $p(r_i = 1) = \theta_i$ and $p(r_i = 0) = 1 - \theta_i$

$\theta_i \sim p(\theta)$, but otherwise unknown

this defines a POMDP with $s = [\theta_1, \dots, \theta_n]$

belief state is $\hat{p}(\theta_1, \dots, \theta_n)$

- solving the POMDP yields the optimal exploration strategy
- but that's overkill: belief state is huge!
- we can do very well with much simpler strategies

how do we measure goodness of exploration algorithm?

regret: difference from optimal policy at time step T :

$$\text{Reg}(T) = TE[r(a^*)] - \sum_{t=1}^T r(a_t)$$

expected reward of best action ↗
(the best we can hope for in expectation) ↘
actual reward of action
actually taken

Three Classes of Exploration Methods

How can we beat the bandit?

$$\text{Reg}(T) = TE[r(a^*)] - \sum_{t=1}^T r(a_t)$$

expected reward of best action
(the best we can hope for in expectation) ↗

actual reward of action
actually taken ↘

- Variety of relatively simple strategies
- Often can provide theoretical guarantees on regret
 - Variety of optimal algorithms (up to a constant factor)
 - But empirical performance may vary...
- Exploration strategies for more complex MDP domains will be inspired by these strategies

Optimistic exploration

keep track of average reward $\hat{\mu}_a$ for each action a

exploitation: pick $a = \arg \max \hat{\mu}_a$

optimistic estimate: $a = \arg \max \hat{\mu}_a + C \overline{\sigma_a}$
some sort of variance estimate

intuition: try each arm until you are *sure* it's not great

example (Auer et al. Finite-time analysis of the multiarmed bandit problem):

$$a = \arg \max \hat{\mu}_a + \sqrt{\frac{2 \ln T}{N(a)}}$$

← number of times we
picked this action

$\text{Reg}(T)$ is $O(\log T)$, provably as good as any algorithm

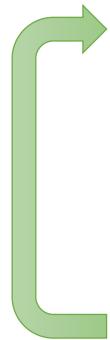
Probability matching/posterior sampling

assume $r(a_i) \sim p_{\theta_i}(r_i)$

this defines a POMDP with $\mathbf{s} = [\theta_1, \dots, \theta_n]$

belief state is $\hat{p}(\theta_1, \dots, \theta_n)$

this is a *model* of our bandit

- 
- idea: sample $\theta_1, \dots, \theta_n \sim \hat{p}(\theta_1, \dots, \theta_n)$
 - pretend the model $\theta_1, \dots, \theta_n$ is correct
 - take the optimal action
 - update the model

- This is called posterior sampling or Thompson sampling
- Harder to analyze theoretically
- Can work very well empirically

See: Chapelle & Li, “An Empirical Evaluation of Thompson Sampling.”

Information gain

Bayesian experimental design:

say we want to determine some latent variable z (e.g., z might be the optimal action, or its value)
which action do we take?

let $\mathcal{H}(\hat{p}(z))$ be the current entropy of our z estimate

let $\mathcal{H}(\hat{p}(z)|y)$ be the entropy of our z estimate after observation y (e.g., y might be $r(a)$)

the lower the entropy, the more precisely we know z

$$\text{IG}(z, y) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z)|y)]$$

typically depends on action, so we have $\text{IG}(z, y|a)$

Information gain example

$$\text{IG}(z, y|a) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z)|y)|a]$$

how much we learn about z from action a , given current beliefs

Example bandit algorithm:

Russo & Van Roy “Learning to Optimize via Information-Directed Sampling”

$$y = r_a, z = \theta_a \text{ (parameters of model } p(r_a))$$

$$g(a) = \text{IG}(\theta_a, r_a|a) - \text{information gain of } a$$

$$\Delta(a) = E[r(a^*) - r(a)] - \text{expected suboptimality of } a$$

choose a according to $\arg \min_a \frac{\Delta(a)^2}{g(a)}$

don't take actions that you're
sure are suboptimal

don't bother taking actions if
you won't learn anything

General themes

UCB:

$$a = \arg \max \hat{\mu}_a + \sqrt{\frac{2 \ln T}{N(a)}}$$

Thompson sampling:

$$\begin{aligned}\theta_1, \dots, \theta_n &\sim \hat{p}(\theta_1, \dots, \theta_n) \\ a &= \arg \max_a E_{\theta_a}[r(a)]\end{aligned}$$

Info gain:

$$\text{IG}(z, y|a)$$

- Most exploration strategies require some kind of uncertainty estimation (even if it's naïve)
- Usually assumes some value to new information
 - Assume unknown = good (optimism)
 - Assume sample = truth
 - Assume information gain = good

Why should we care?

- Bandits are easier to analyze and understand
- Can derive foundations for exploration methods
- Then apply these methods to more complex MDPs
- Not covered here:
 - Contextual bandits (bandits with state, essentially 1-step MDPs)
 - Optimal exploration in small MDPs
 - Bayesian model-based reinforcement learning (similar to information gain)
 - Probably approximately correct (PAC) exploration

Exploration in Deep RL

Recap: classes of exploration methods in deep RL

- Optimistic exploration:
 - new state = good state
 - requires estimating state visitation frequencies or novelty
 - typically realized by means of exploration bonuses
- Thompson sampling style algorithms:
 - learn distribution over Q-functions or policies
 - sample and act according to sample
- Information gain style algorithms
 - reason about information gain from visiting new states

Optimistic exploration in RL

$$\text{UCB: } a = \arg \max \hat{\mu}_a + \sqrt{\frac{2 \ln T}{N(a)}}$$

“exploration bonus”

lots of functions work, so long as they decrease with $N(a)$

can we use this idea with MDPs?

count-based exploration: use $N(\mathbf{s}, \mathbf{a})$ or $N(\mathbf{s})$ to add *exploration bonus*

use $r^+(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \mathcal{B}(N(\mathbf{s}))$



bonus that decreases with $N(\mathbf{s})$

use $r^+(\mathbf{s}, \mathbf{a})$ instead of $r(\mathbf{s}, \mathbf{a})$ with any model-free algorithm

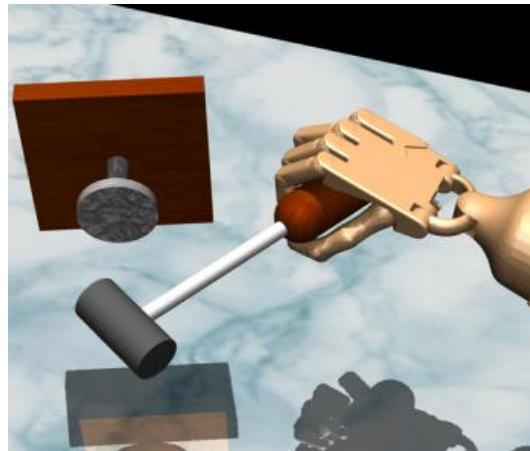
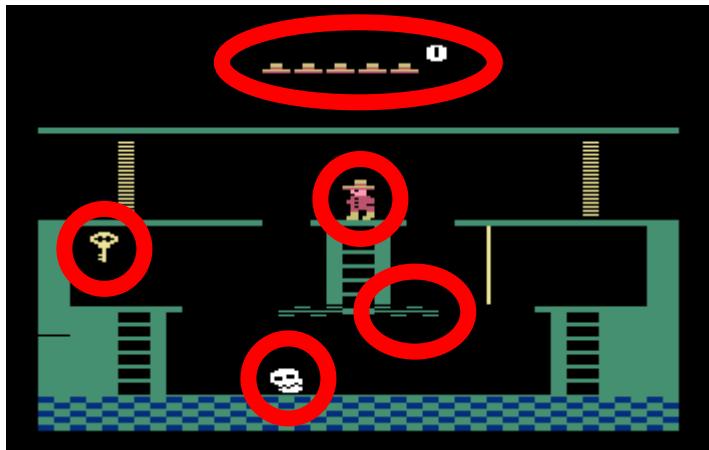
+ simple addition to any RL algorithm

- need to tune bonus weight

The trouble with counts

use $r^+(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \mathcal{B}(N(\mathbf{s}))$

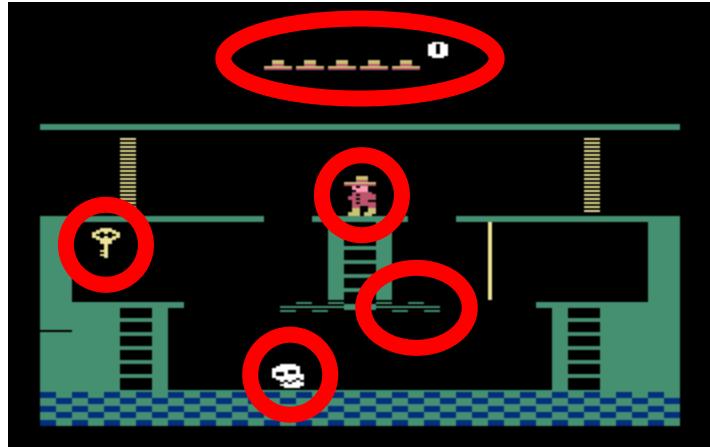
But wait... what's a count?



Uh oh... we never see the same thing twice!

But some states are more similar than others

Fitting generative models

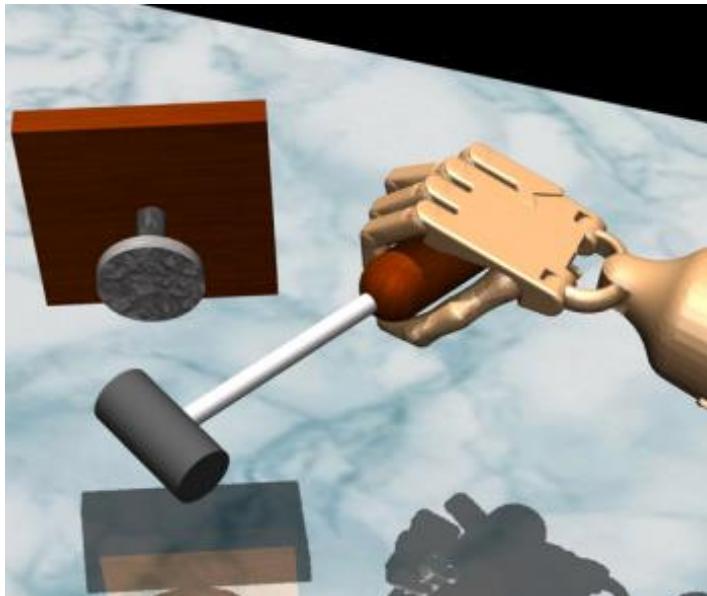


idea: fit a density model $p_\theta(\mathbf{s})$ (or $p_\theta(\mathbf{s}, \mathbf{a})$)

$p_\theta(\mathbf{s})$ might be high even for a new \mathbf{s}
if \mathbf{s} is similar to previously seen states

can we use $p_\theta(\mathbf{s})$ to get a “pseudo-count”?

if we have small MDPs
the true probability is:



after we see \mathbf{s} , we have:

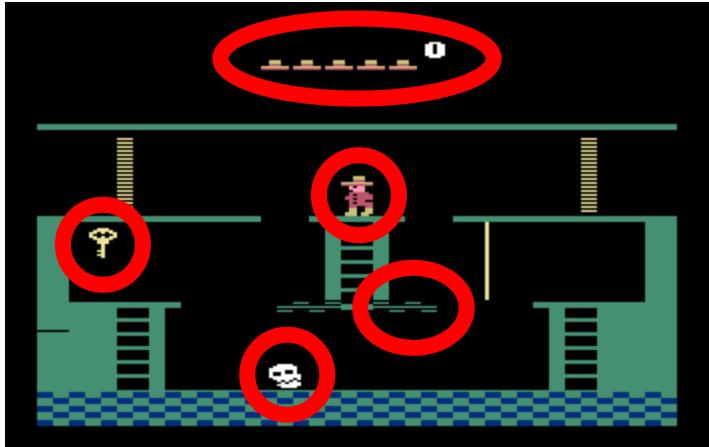
$$P(\mathbf{s}) = \frac{N(\mathbf{s})}{n}$$

↑ ← count
probability/density total states visited

$$P'(\mathbf{s}) = \frac{N(\mathbf{s}) + 1}{n + 1}$$

can we get $p_\theta(\mathbf{s})$ and $p_{\theta'}(\mathbf{s})$ to obey these equations?

Exploring with pseudo-counts



- fit model $p_{\theta}(\mathbf{s})$ to all states \mathcal{D} seen so far
take a step i and observe \mathbf{s}_i
fit new model $p_{\theta'}(\mathbf{s})$ to $\mathcal{D} \cup \mathbf{s}_i$
use $p_{\theta}(\mathbf{s}_i)$ and $p_{\theta'}(\mathbf{s}_i)$ to estimate $\hat{N}(\mathbf{s})$
set $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$ ← “pseudo-count”

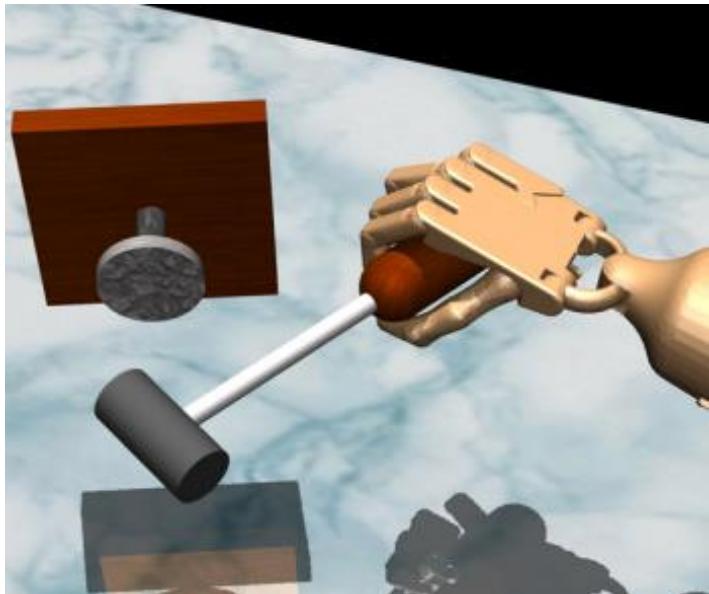
how to get $\hat{N}(\mathbf{s})$? use the equations

$$p_{\theta}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}}$$

$$p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i) + 1}{\hat{n} + 1}$$

two equations and two unknowns!

$$\hat{N}(\mathbf{s}_i) = \hat{n} p_{\theta}(\mathbf{s}_i) \quad \hat{n} = \frac{1 - p_{\theta'}(\mathbf{s}_i)}{p_{\theta'}(\mathbf{s}_i) - p_{\theta}(\mathbf{s}_i)} p_{\theta}(\mathbf{s}_i)$$



What kind of bonus to use?

Lots of functions in the literature, inspired by optimal methods for bandits or small MDPs

UCB:

$$\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{2 \ln n}{N(\mathbf{s})}}$$

MBIE-EB (Strehl & Littman, 2008):

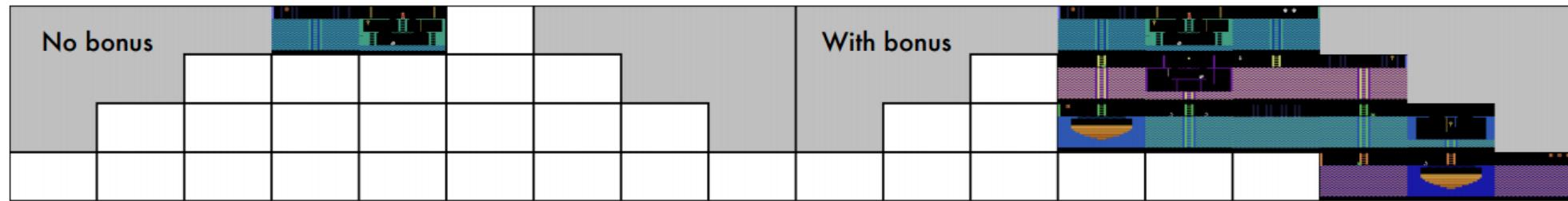
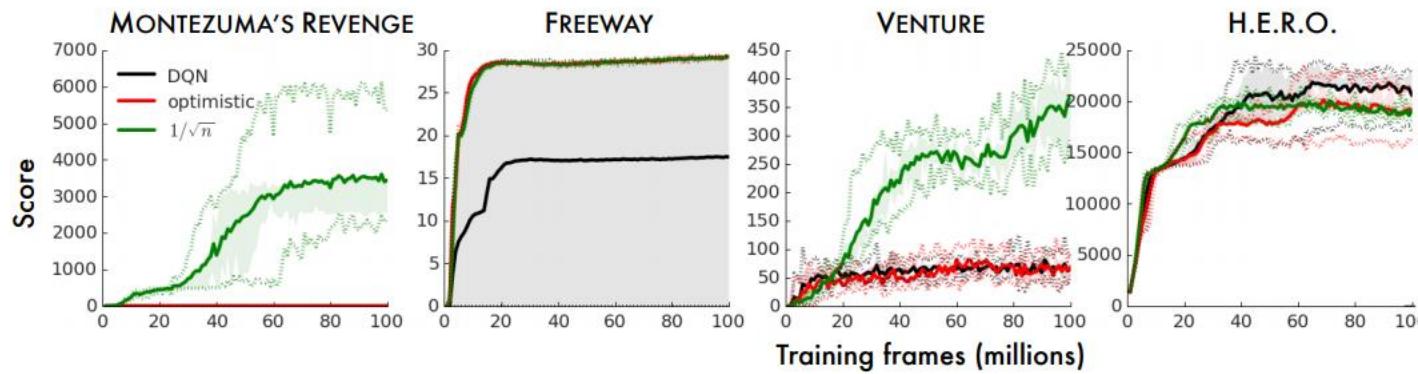
$$\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{1}{N(\mathbf{s})}}$$

BEB (Kolter & Ng, 2009):

$$\mathcal{B}(N(\mathbf{s})) = \frac{1}{N(\mathbf{s})}$$

this is the one used by Bellemare et al. '16

Does it work?



What kind of model to use?

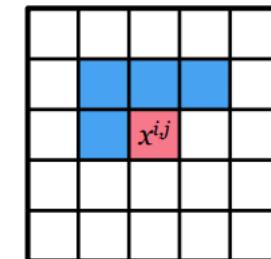


$$p_{\theta}(\mathbf{s})$$

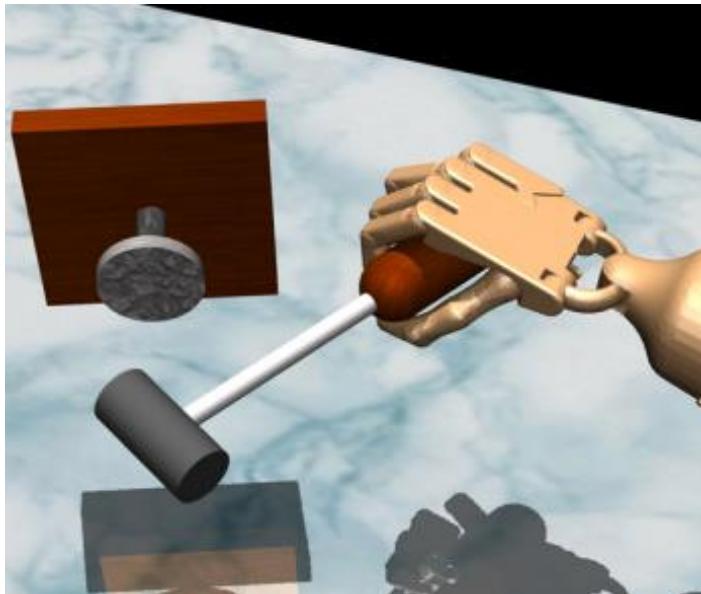
need to be able to output densities, but doesn't necessarily need to produce great samples

opposite considerations from many popular generative models in the literature (e.g., GANs)

Bellemare et al.: "CTS" model:
condition each pixel on its top-left neighborhood



Other models: stochastic neural networks, compression length, EX2



More Novelty-Seeking Exploration

Counting with hashes

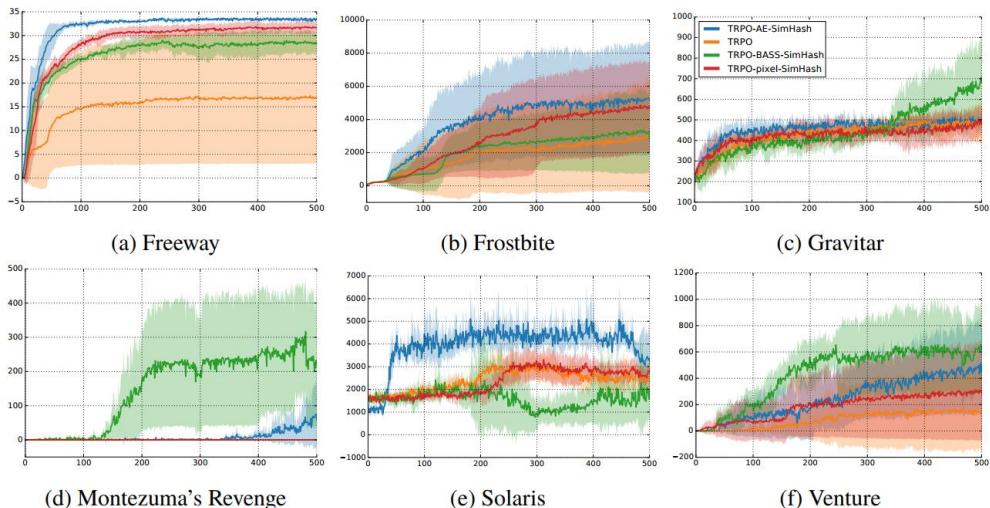
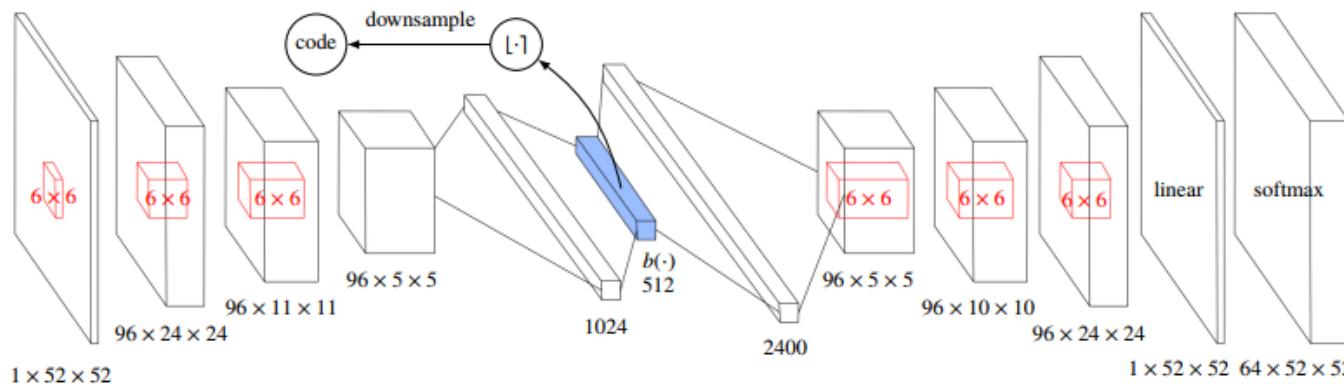
What if we still count states, but in a different space?

idea: compress \mathbf{s} into a k -bit code via $\phi(\mathbf{s})$, then count $N(\phi(\mathbf{s}))$

shorter codes = more hash collisions

similar states get the same hash? maybe

improve the odds by *learning* a compression:



Implicit density modeling with exemplar models

$$p_{\theta}(\mathbf{s})$$

need to be able to output densities, but doesn't necessarily need to produce great samples

Can we explicitly compare the new state to past states?

Intuition: the state is **novel** if it is **easy** to distinguish from all previous seen states by a classifier

for each observed state \mathbf{s} , fit a classifier to classify that state against all past states \mathcal{D} , use classifier error to obtain density

$$p_{\theta}(\mathbf{s}) = \frac{1 - D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$$

probability that classifier assigns that \mathbf{s} is “positive”
positives: $\{\mathbf{s}\}$
negatives: \mathcal{D}

Implicit density modeling with exemplar models

hang on... aren't we just checking if $\mathbf{s} = \mathbf{s}$?

if $\mathbf{s} \in \mathcal{D}$, then the optimal $D_{\mathbf{s}}(\mathbf{s}) \neq 1$

in fact: $D_{\mathbf{s}}^*(\mathbf{s}) = \frac{1}{1 + p(\mathbf{s})}$



$$p_{\theta}(\mathbf{s}) = \frac{1 - D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$$

in reality, each state is unique, so we *regularize* the classifier

isn't one classifier per state a bit much?

train one *amortized* model: single network that takes in exemplar as input!

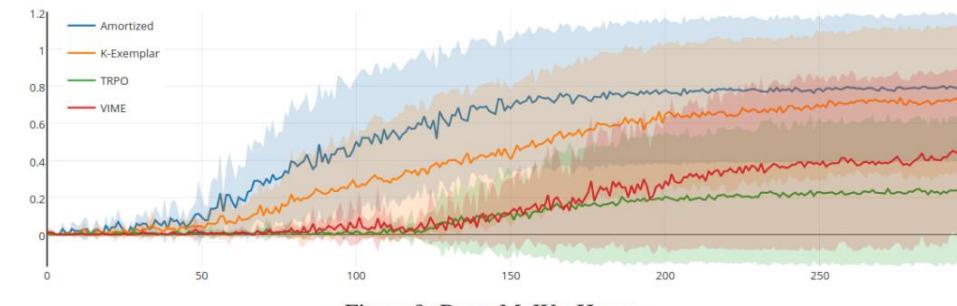
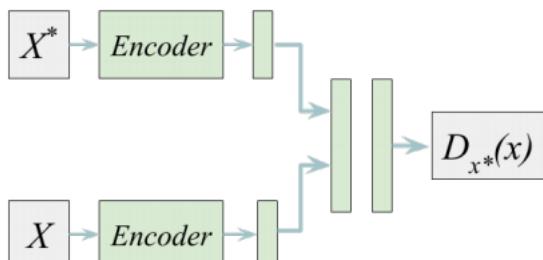
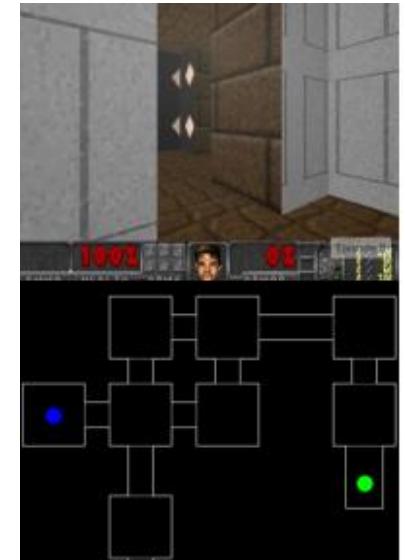


Figure 9: DoomMyWayHome+



Heuristic estimation of counts via errors

$p_\theta(\mathbf{s})$

need to be able to output densities, but doesn't necessarily need to produce great samples

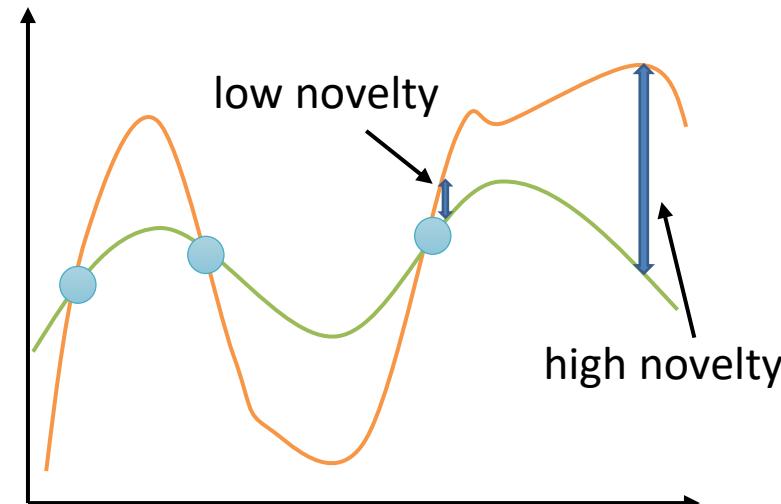
...and doesn't even need to output great densities

...just need to tell if state is **novel** or not!

let's say we have some **target** function $f^*(\mathbf{s}, \mathbf{a})$

given our buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$, fit $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$

use $\mathcal{E}(\mathbf{s}, \mathbf{a}) = \|\hat{f}_\theta(\mathbf{s}, \mathbf{a}) - f^*(\mathbf{s}, \mathbf{a})\|^2$ as bonus



Heuristic estimation of counts via errors

let's say we have some **target** function $f^*(\mathbf{s}, \mathbf{a})$

given our buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$, fit $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$

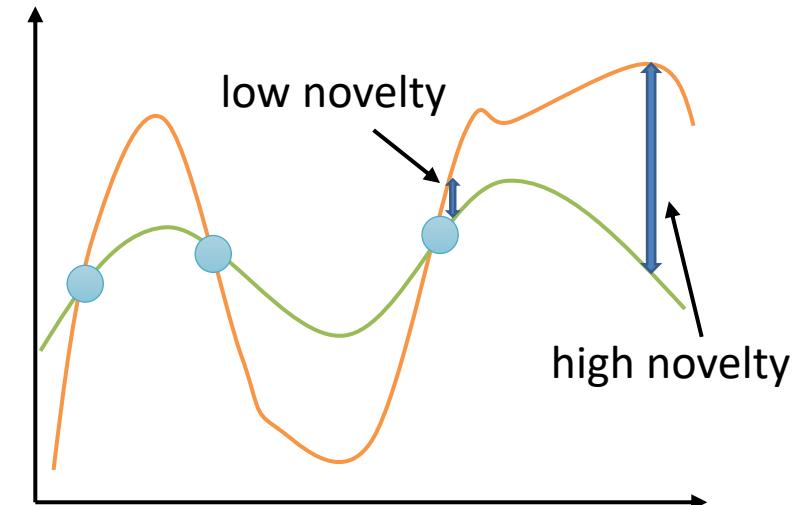
use $\mathcal{E}(\mathbf{s}, \mathbf{a}) = \|\hat{f}_\theta(\mathbf{s}, \mathbf{a}) - f^*(\mathbf{s}, \mathbf{a})\|^2$ as bonus

what should we use for $f^*(\mathbf{s}, \mathbf{a})$?

one common choice: set $f^*(\mathbf{s}, \mathbf{a}) = \mathbf{s}'$ – i.e., next state prediction

- also related to information gain, which we'll discuss next time!

even simpler: $f^*(\mathbf{s}, \mathbf{a}) = f_\phi(\mathbf{s}, \mathbf{a})$, where ϕ is a *random* parameter vector



this will be in HW5!

Posterior Sampling in Deep RL

Posterior sampling in deep RL

Thompson sampling:

$$\theta_1, \dots, \theta_n \sim \hat{p}(\theta_1, \dots, \theta_n)$$

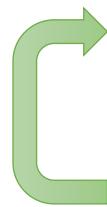
$$a = \arg \max_a E_{\theta_a}[r(a)]$$

What do we sample?

How do we represent the distribution?

bandit setting: $\hat{p}(\theta_1, \dots, \theta_n)$ is distribution over *rewards*

MDP analog is the *Q*-function!



1. sample Q-function Q from $p(Q)$
2. act according to Q for one episode
3. update $p(Q)$

since Q-learning is off-policy, we don't care
which Q-function was used to collect data

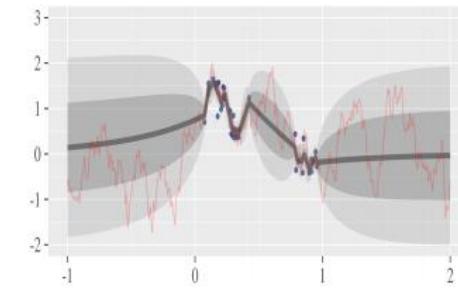
how can we represent a distribution over functions?

Bootstrap

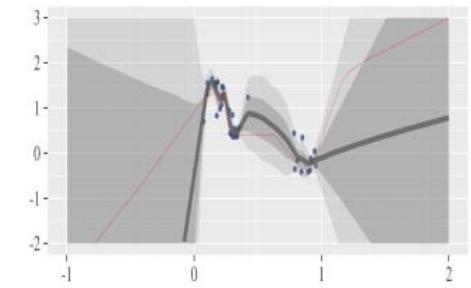
given a dataset \mathcal{D} , resample with replacement N times to get $\mathcal{D}_1, \dots, \mathcal{D}_N$

train each model f_{θ_i} on \mathcal{D}_i

to sample from $p(\theta)$, sample $i \in [1, \dots, N]$ and use f_{θ_i}

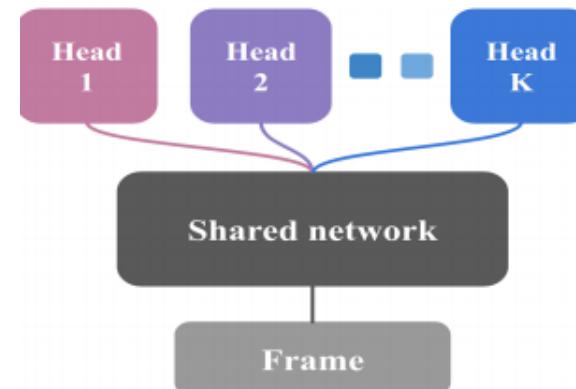


(b) Gaussian process posterior



(c) Bootstrapped neural nets

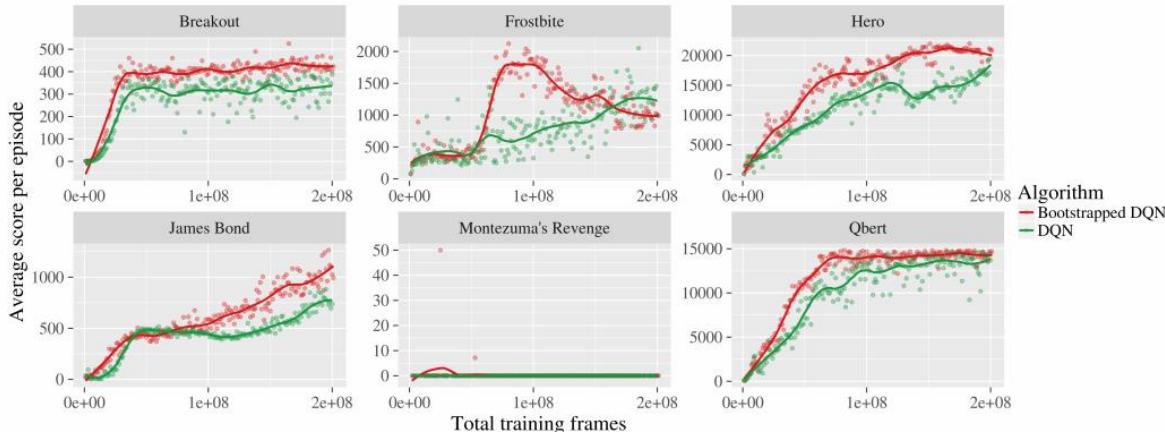
training N big neural nets is expensive, can we avoid it?



Why does this work?

Exploring with random actions (e.g., epsilon-greedy): oscillate back and forth, might not go to a coherent or interesting place

Exploring with random Q-functions: commit to a randomized but internally consistent strategy for an entire episode



- + no change to original reward function
- very good bonuses often do better

Information Gain in Deep RL

Reasoning about information gain (approximately)

Info gain: $\text{IG}(z, y|a)$

information gain about *what*?

information gain about reward $r(\mathbf{s}, \mathbf{a})$?

not very useful if reward is sparse

state density $p(\mathbf{s})$?

a bit strange, but somewhat makes sense!

information gain about dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$?

good proxy for *learning* the MDP, though still heuristic

Generally intractable to use exactly, regardless of what is being estimated!

Reasoning about information gain (approximately)

Generally intractable to use exactly, regardless of what is being estimated

A few approximations:

prediction gain: $\log p_{\theta'}(\mathbf{s}) - \log p_{\theta}(\mathbf{s})$ (Schmidhuber '91, Bellemare '16)

intuition: if density changed a lot, the state was novel

variational inference:

(Houthooft et al. "VIME")

IG can be equivalently written as $D_{\text{KL}}(p(z|y) \| p(z))$

learn about *transitions* $p_{\theta}(s_{t+1}|s_t, a_t)$: $z = \theta$

$y = (s_t, a_t, s_{t+1})$

$$D_{\text{KL}}(p(\theta|h, s_t, a_t, s_{t+1}) \| p(\theta|h))$$

↑
model parameters for $p_{\theta}(s_{t+1}|s_t, a_t)$ ↑
newly observed transition
history of all prior transitions

intuition: a transition is more informative if it causes belief over θ to change

idea: use variational inference to estimate $q(\theta|\phi) \approx p(\theta|h)$

given new transition (s, a, s') , update ϕ to get ϕ'

Reasoning about information gain (approximately)

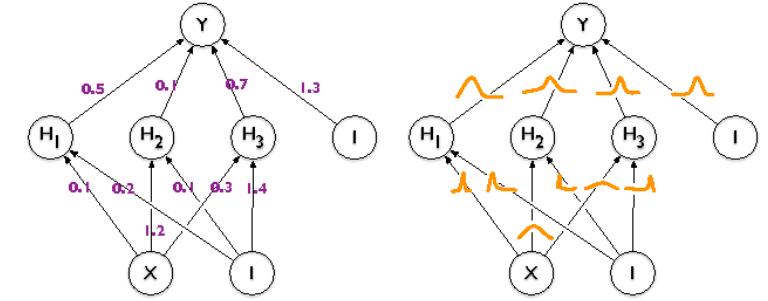
VIME implementation:

IG can be equivalently written as $D_{\text{KL}}(p(\theta|h, s_t, a_t, s_{t+1}) \| p(\theta|h))$

model parameters for $p_\theta(s_{t+1}|s_t, a_t)$

history of all prior transitions

newly observed transition



$$q(\theta|\phi) \approx p(\theta|h)$$

specifically, optimize variational lower bound $D_{\text{KL}}(q(\theta|\phi) \| p(h|\theta)p(\theta))$

represent $q(\theta|\phi)$ as product of independent Gaussian parameter distributions

with mean ϕ (see Blundell et al. “Weight uncertainty in neural networks”)

given new transition (s, a, s') , update ϕ to get ϕ'
i.e., update the network weight means and variances

use $D_{\text{KL}}(q(\theta|\phi') \| q(\theta|\phi))$ as approximate bonus

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

$$\uparrow \uparrow \\ \phi$$

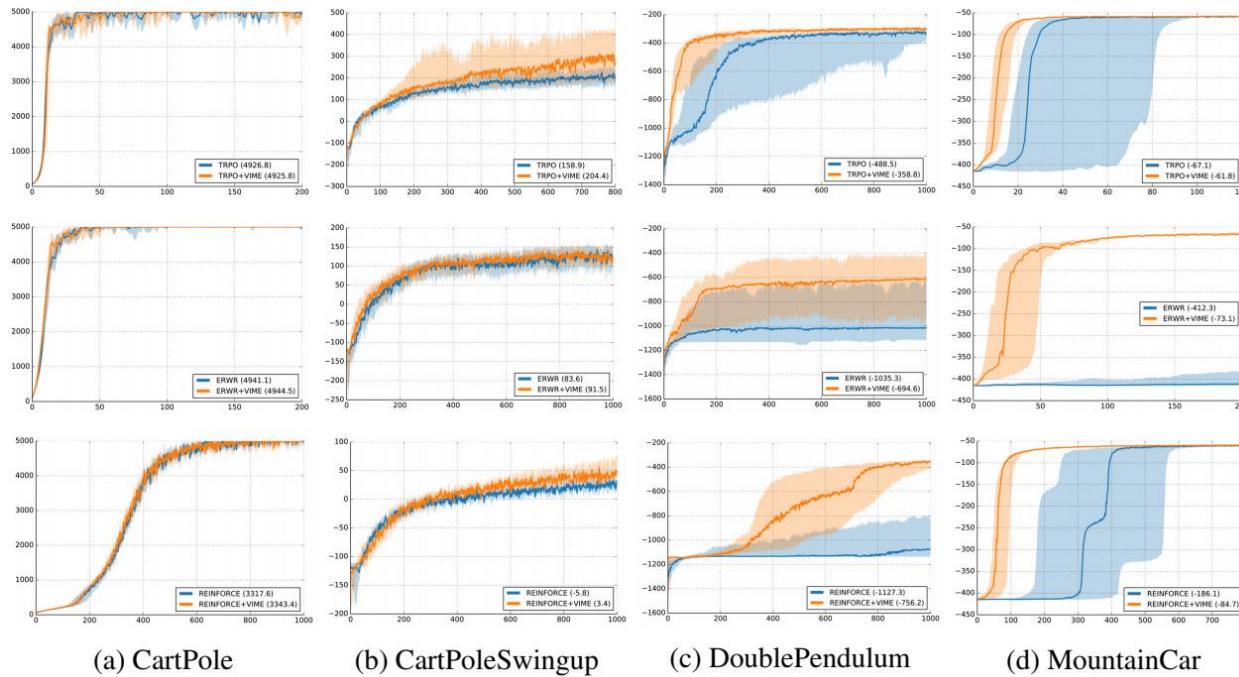
Reasoning about information gain (approximately)

VIME implementation:

IG can be equivalently written as $D_{\text{KL}}(p(\theta|h, s_t, a_t, s_{t+1}) \| p(\theta|h))$

$q(\theta|\phi) \approx p(\theta|h)$ specifically, optimize variational lower bound $D_{\text{KL}}(q(\theta|\phi) \| p(h|\theta)p(\theta))$

use $D_{\text{KL}}(q(\theta|\phi') \| q(\theta|\phi))$ as approximate bonus



Approximate IG:

- + appealing mathematical formalism
- models are more complex, generally harder to use effectively

Exploration with model errors

$D_{\text{KL}}(q(\theta|\phi')\|q(\theta|\phi))$ can be seen as change in network (mean) parameters ϕ
if we forget about IG, there are many other ways to measure this

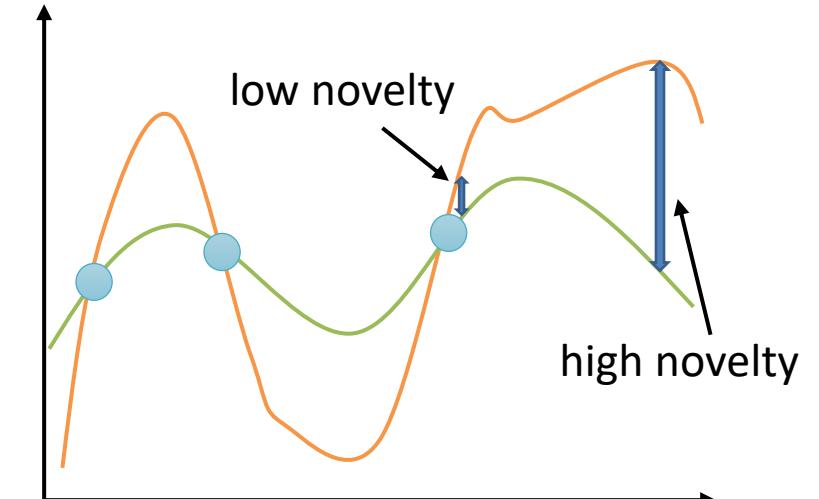
Stadie et al. 2015:

- encode image observations using auto-encoder
- build predictive model on auto-encoder latent states
- use model error as exploration bonus

Schmidhuber et al. (see, e.g. “Formal Theory of Creativity, Fun, and Intrinsic Motivation”):

- exploration bonus for model error
- exploration bonus for model gradient
- many other variations

Many others!



Recap: classes of exploration methods in deep RL

- Optimistic exploration:
 - Exploration with counts and pseudo-counts
 - Different models for estimating densities
- Thompson sampling style algorithms:
 - Maintain a distribution over models via bootstrapping
 - Distribution over Q-functions
- Information gain style algorithms
 - Generally intractable
 - Can use variational approximation to information gain

Suggested readings

Schmidhuber. (1992). **A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers.**

Stadie, Levine, Abbeel (2015). **Incentivizing Exploration in Reinforcement Learning with Deep Predictive Models.**

Osband, Blundell, Pritzel, Van Roy. (2016). **Deep Exploration via Bootstrapped DQN.**

Houthooft, Chen, Duan, Schulman, De Turck, Abbeel. (2016). **VIME: Variational Information Maximizing Exploration.**

Bellemare, Srinivasan, Ostrovski, Schaul, Saxton, Munos. (2016). **Unifying Count-Based Exploration and Intrinsic Motivation.**

Tang, Houthooft, Foote, Stooke, Chen, Duan, Schulman, De Turck, Abbeel. (2016). **#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning.**

Fu, Co-Reyes, Levine. (2017). **EX2: Exploration with Exemplar Models for Deep Reinforcement Learning.**

Exploration (Part 2)

CS 285

Instructor: Sergey Levine
UC Berkeley

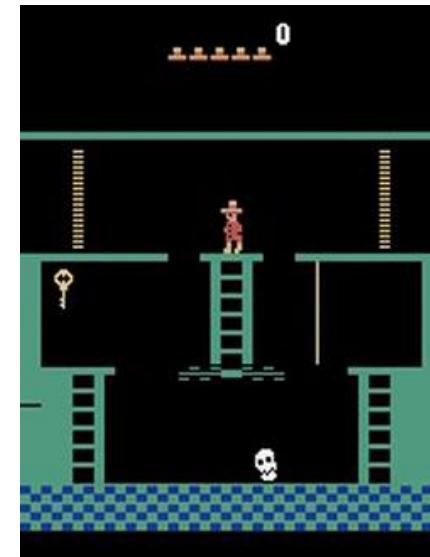


Recap: what's the problem?

this is easy (mostly)



this is impossible



Why?

Unsupervised learning of diverse behaviors

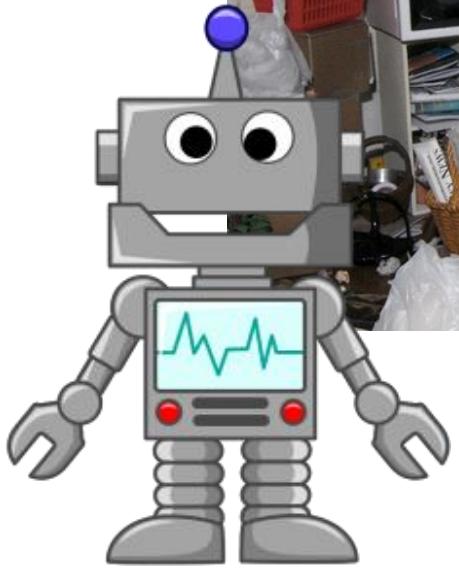
What if we want to recover diverse behavior **without any reward function at all?**



Why?

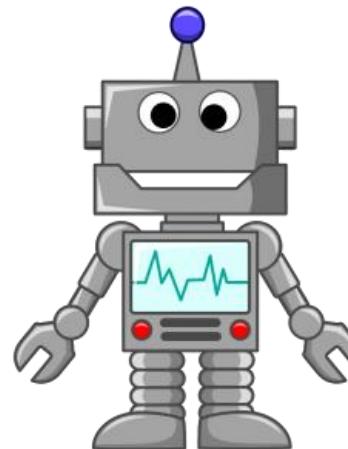
- *Learn skills without supervision, then use them to accomplish goals*
- *Learn sub-skills to use with hierarchical reinforcement learning*
- *Explore the space of possible behaviors*

An Example Scenario



How can you prepare for an **unknown** future goal?

training time: unsupervised



In this lecture...

- Definitions & concepts from information theory
- Learning without a reward function by reaching goals
- A *state distribution-matching* formulation of reinforcement learning
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills*

In this lecture...

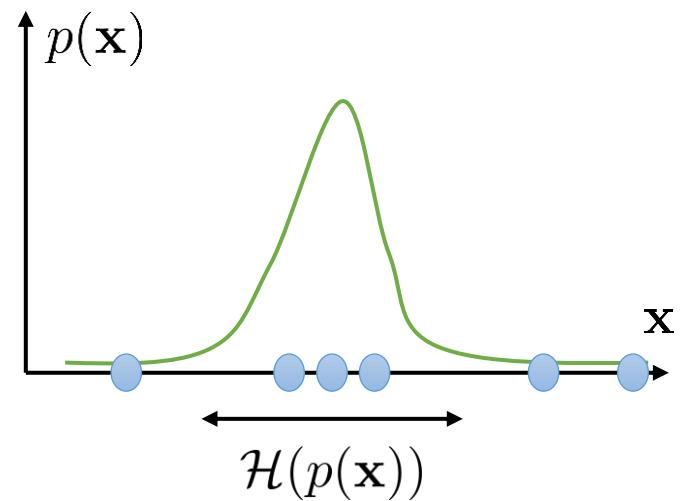
- Definitions & concepts from information theory
- Learning without a reward function by reaching goals
- A *state distribution-matching* formulation of reinforcement learning
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills*

Some useful identities

$p(\mathbf{x})$ distribution (e.g., over observations \mathbf{x})

$$\mathcal{H}(p(\mathbf{x})) = -E_{\mathbf{x} \sim p(\mathbf{x})}[\log p(\mathbf{x})]$$

entropy – how “broad” $p(\mathbf{x})$ is



Some useful identities

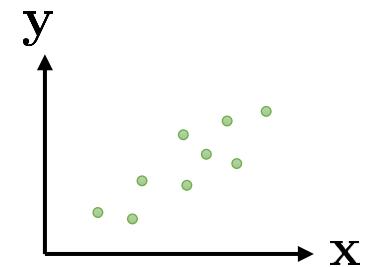
entropy – how “broad” $p(\mathbf{x})$ is

$$\mathcal{H}(p(\mathbf{x})) = -E_{\mathbf{x} \sim p(\mathbf{x})}[\log p(\mathbf{x})]$$

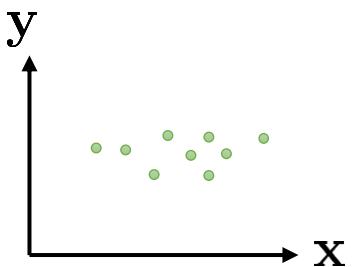
$$\mathcal{I}(\mathbf{x}; \mathbf{y}) = D_{\text{KL}}(p(\mathbf{x}, \mathbf{y}) || p(\mathbf{x})p(\mathbf{y}))$$

$$= E_{(\mathbf{x}, \mathbf{y}) \sim p(\mathbf{x}, \mathbf{y})} \left[\log \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})p(\mathbf{y})} \right]$$

$$= \mathcal{H}(p(\mathbf{y})) - \mathcal{H}(p(\mathbf{y}|\mathbf{x}))$$



high MI: \mathbf{x} and \mathbf{y} are *dependent*



low MI: \mathbf{x} and \mathbf{y} are *independent*

Information theoretic quantities in RL

$\pi(\mathbf{s})$ state *marginal* distribution of policy π

$\mathcal{H}(\pi(\mathbf{s}))$ state *marginal* entropy of policy π

quantifies *coverage*

example of mutual information: “empowerment” (Polani et al.)

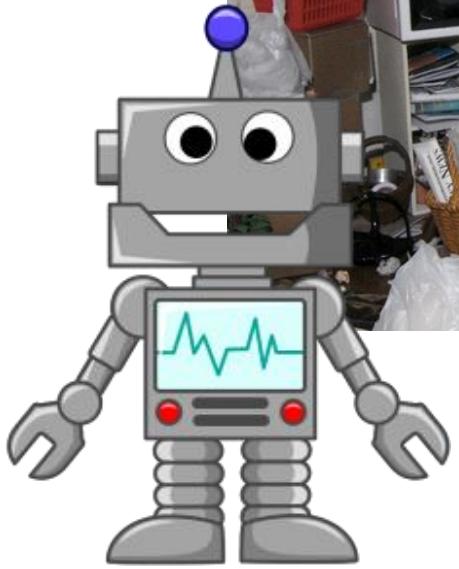
$$\mathcal{I}(\mathbf{s}_{t+1}; \mathbf{a}_t) = \mathcal{H}(\mathbf{s}_{t+1}) - \mathcal{H}(\mathbf{s}_{t+1} | \mathbf{a}_t)$$

can be viewed as quantifying “control authority” in an information-theoretic way

In this lecture...

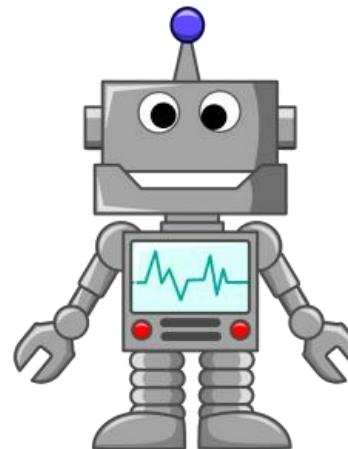
- Definitions & concepts from information theory
- Learning without a reward function by reaching goals
- A *state distribution-matching* formulation of reinforcement learning
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills*

An Example Scenario

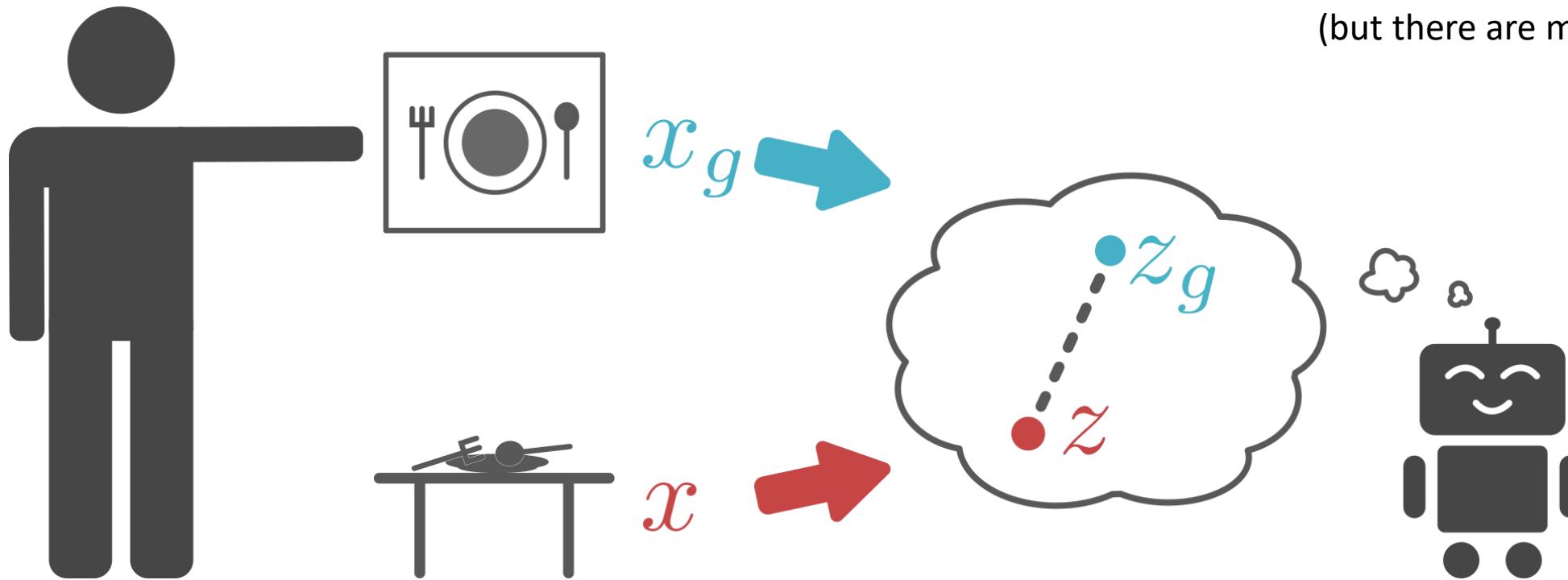


How can you prepare for an **unknown** future goal?

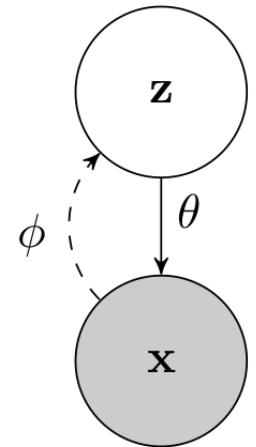
training time: unsupervised



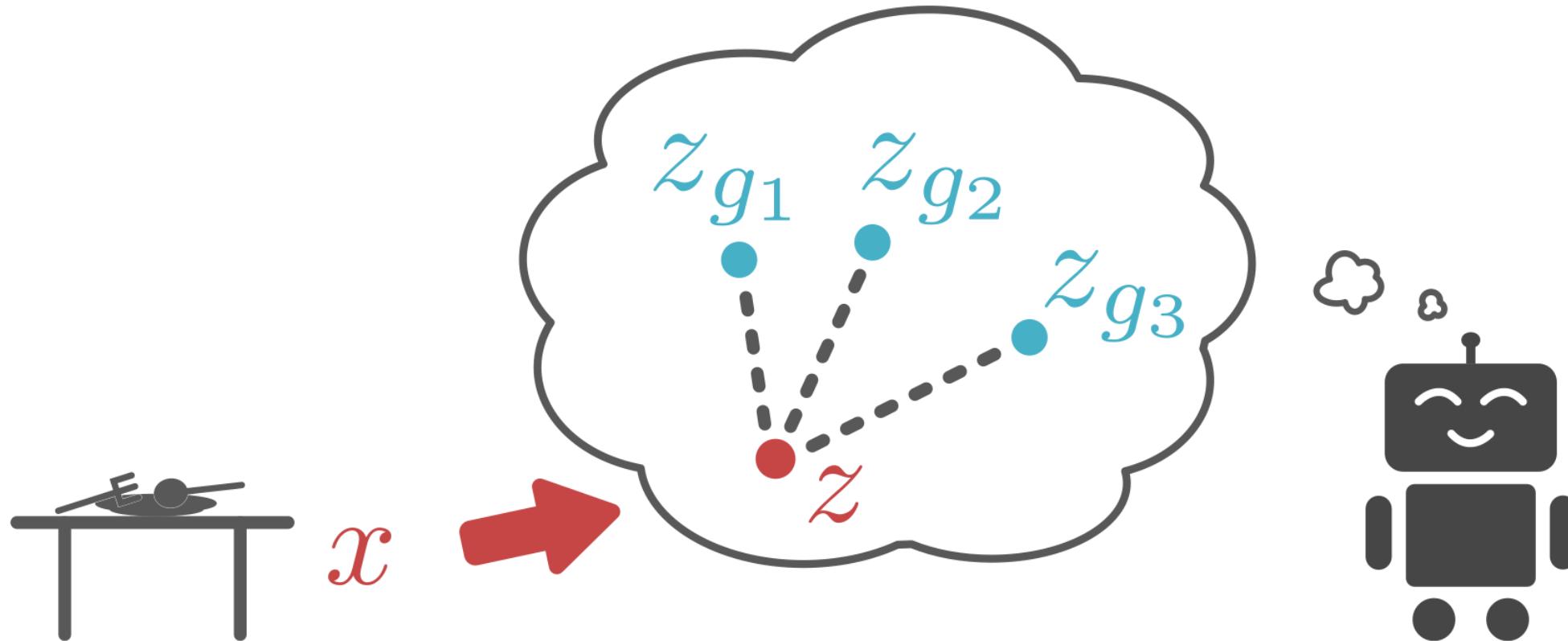
Learn without any rewards at all



VAE (Kingma & Welling '13)
(but there are many other choices)

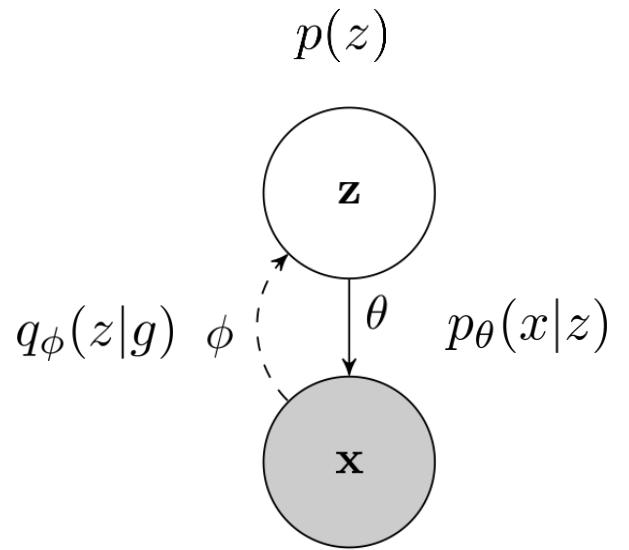
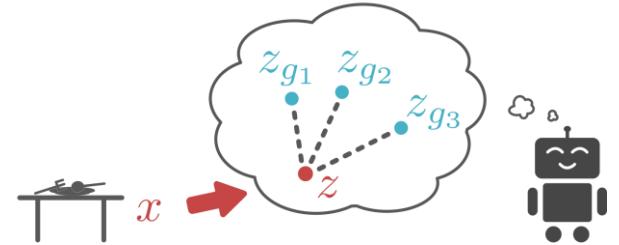


Learn without any rewards at all

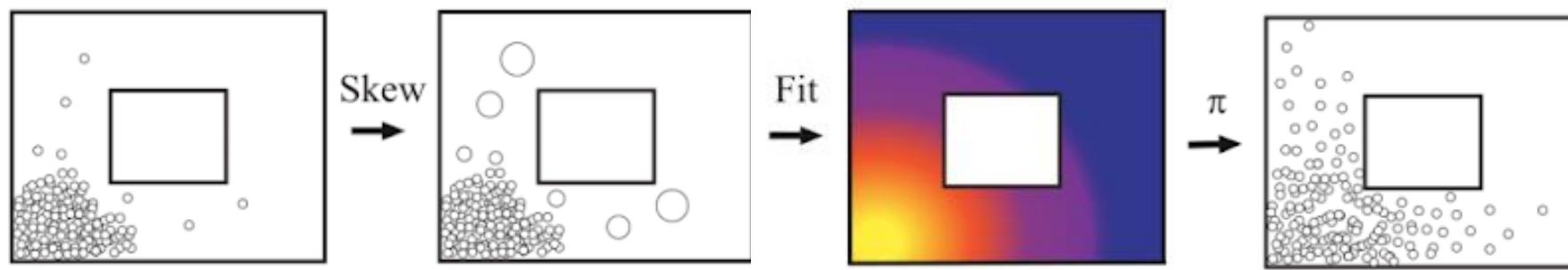
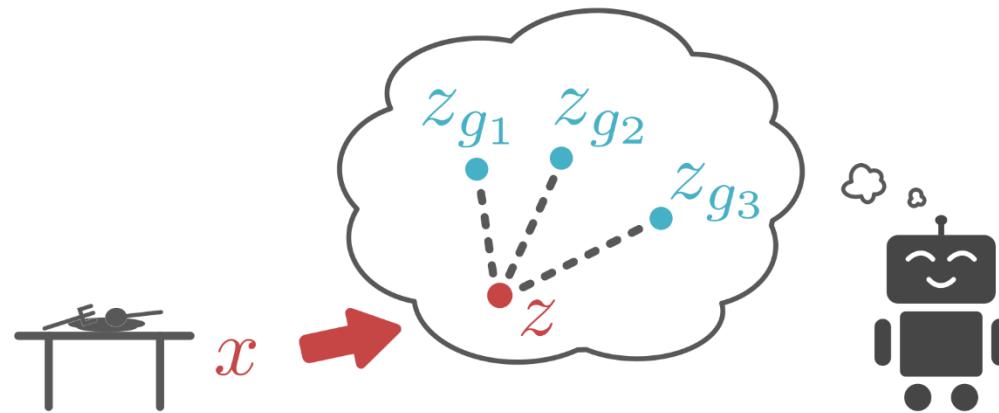


Learn without any rewards at all

1. Propose goal: $z_g \sim p(z)$, $x_g \sim p_\theta(x_g|z_g)$
2. Attempt to reach goal using $\pi(a|x, x_g)$, reach \bar{x}
3. Use data to update π
4. Use data to update $p_\theta(x_g|z_g)$, $q_\phi(z_g|x_g)$



How do we get diverse goals?



How do we get diverse goals?

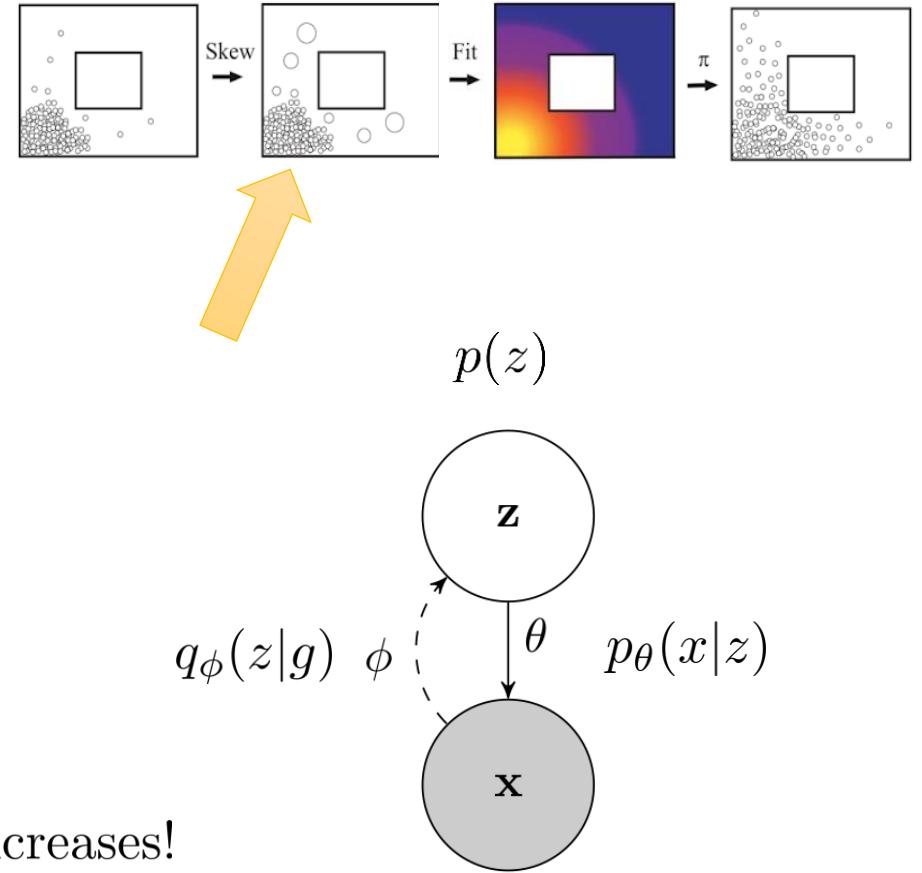
1. Propose goal: $z_g \sim p(z)$, $x_g \sim p_\theta(x_g|z_g)$
2. Attempt to reach goal using $\pi(a|x, x_g)$, reach \bar{x}
3. Use data to update π
4. Use data to update $p_\theta(x_g|z_g)$, $q_\phi(z_g|x_g)$

standard MLE: $\theta, \phi \leftarrow \arg \max_{\theta, \phi} E[\log p(\bar{x})]$

weighted MLE: $\theta, \phi \leftarrow \arg \max_{\theta, \phi} E[w(\bar{x}) \log p(\bar{x})]$

$$w(\bar{x}) = p_\theta(\bar{x})^\alpha$$

key result: for any $\alpha \in [-1, 0)$, entropy $\mathcal{H}(p_\theta(x))$ increases!



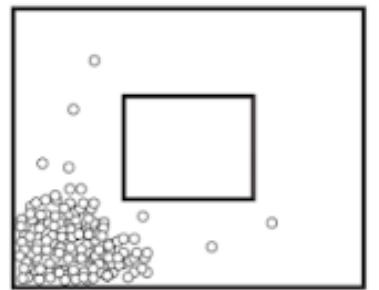
How do we get diverse goals?

what is the objective?

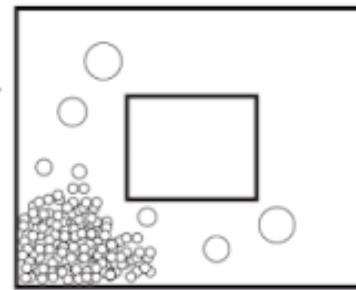
$$\max \mathcal{H}(p(G)) - \mathcal{H}(p(G|S))$$

goals get higher
entropy due to Skew-Fit

$$w(\bar{x}) = p_\theta(\bar{x})^\alpha$$
$$\alpha \in [-1, 0)$$



Skew



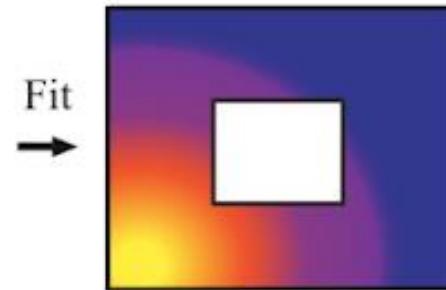
what does RL do?

$\pi(a|S, G)$ trained to reach goal G

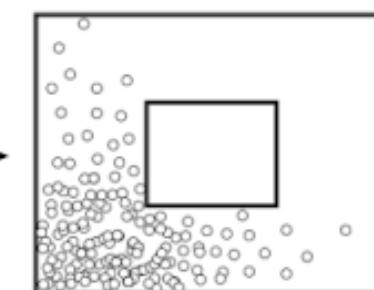
as π gets better, final state S gets close to G

that means $p(G|S)$ becomes more deterministic!

goal final state



π



How do we get diverse goals?

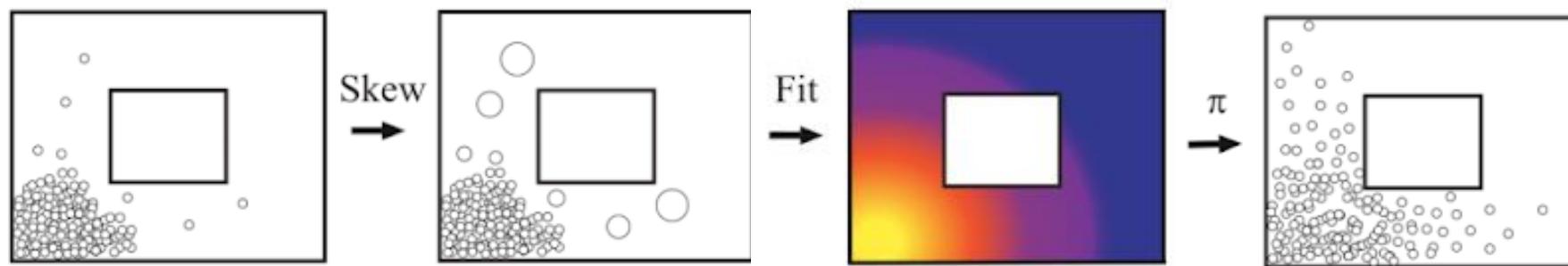
what is the objective?

$$\max \mathcal{H}(p(G)) - \mathcal{H}(p(G|S)) = \max \mathcal{I}(S; G)$$

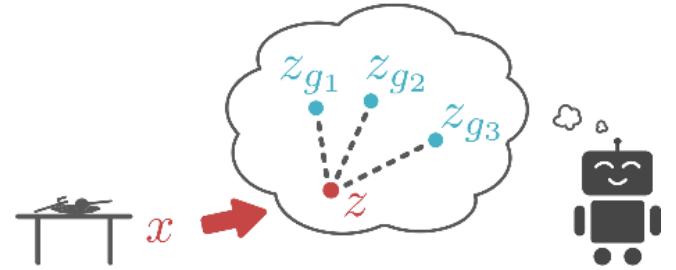
maximizing mutual information between S and G leads to

good exploration (state coverage) – $\mathcal{H}(p(G))$

effective goal reaching – $\mathcal{H}(p(G|S))$



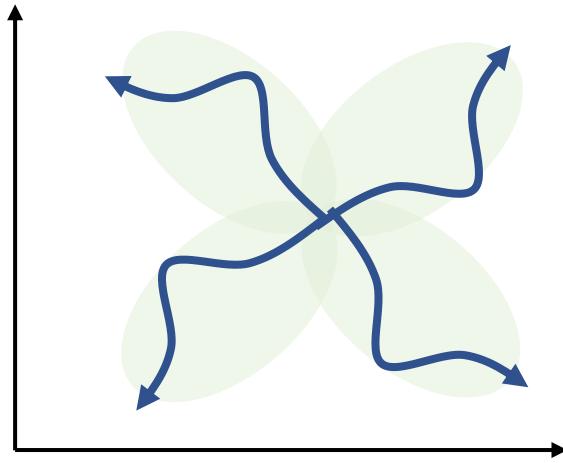
Reinforcement learning with *imagined* goals



In this lecture...

- Definitions & concepts from information theory
- Learning without a reward function by reaching goals
- A *state distribution-matching* formulation of reinforcement learning
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills*

Aside: exploration with intrinsic motivation



common method for exploration:

incentivize policy $\pi(\mathbf{a}|\mathbf{s})$ to explore diverse states

...before seeing any reward

reward visiting **novel** states

if a state is visited *often*, it is not *novel*

⇒ add an exploration bonus to reward: $\tilde{r}(\mathbf{s}) = r(\mathbf{s}) - \log p_\pi(\mathbf{s})$

↑
state density under $\pi(\mathbf{a}|\mathbf{s})$

- ➡ 1. update $\pi(\mathbf{a}|\mathbf{s})$ to maximize $E_\pi[\tilde{r}(\mathbf{s})]$
- 2. update $p_\pi(\mathbf{s})$ to fit state marginal

Can we use this for state marginal matching?

the state marginal matching problem: learn $\pi(\mathbf{a}|\mathbf{s})$ so as to minimize $D_{\text{KL}}(p_{\pi}(\mathbf{s}) \| p^*(\mathbf{s}))$

idea: can we use intrinsic motivation?

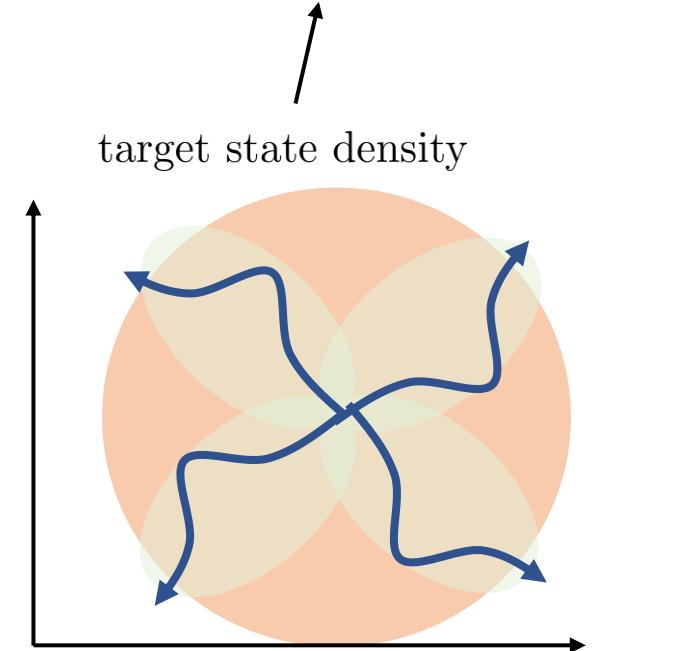
$$\tilde{r}(\mathbf{s}) = \log p^*(\mathbf{s}) - \log p_{\pi}(\mathbf{s})$$

this does **not** perform marginal matching!

- 1. learn $\pi^k(\mathbf{a}|\mathbf{s})$ to maximize $E_{\pi}[\tilde{r}^k(\mathbf{s})]$
- 2. ~~update $p_{\pi^k}(\mathbf{s})$ to fit state marginal~~
2. update $p_{\pi^k}(\mathbf{s})$ to fit *all states seen so far*
- 3. return $\pi^*(\mathbf{a}|\mathbf{s}) = \sum_k \pi^k(\mathbf{a}|\mathbf{s})$

this **does** perform marginal matching!

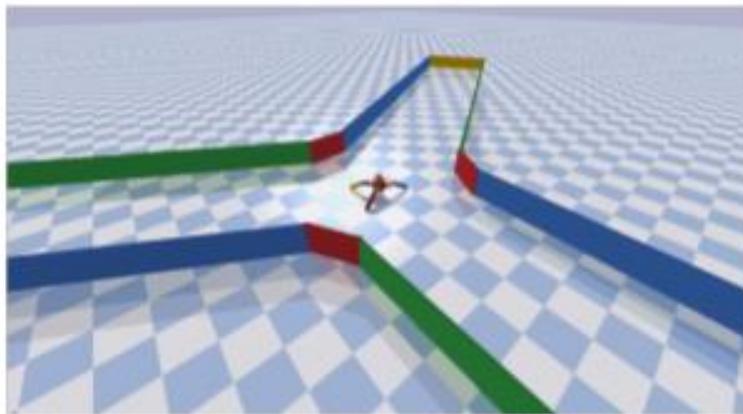
$p_{\pi}(\mathbf{s}) = p^*(\mathbf{s})$ is Nash equilibrium of two player game between π^k and p_{π^k}



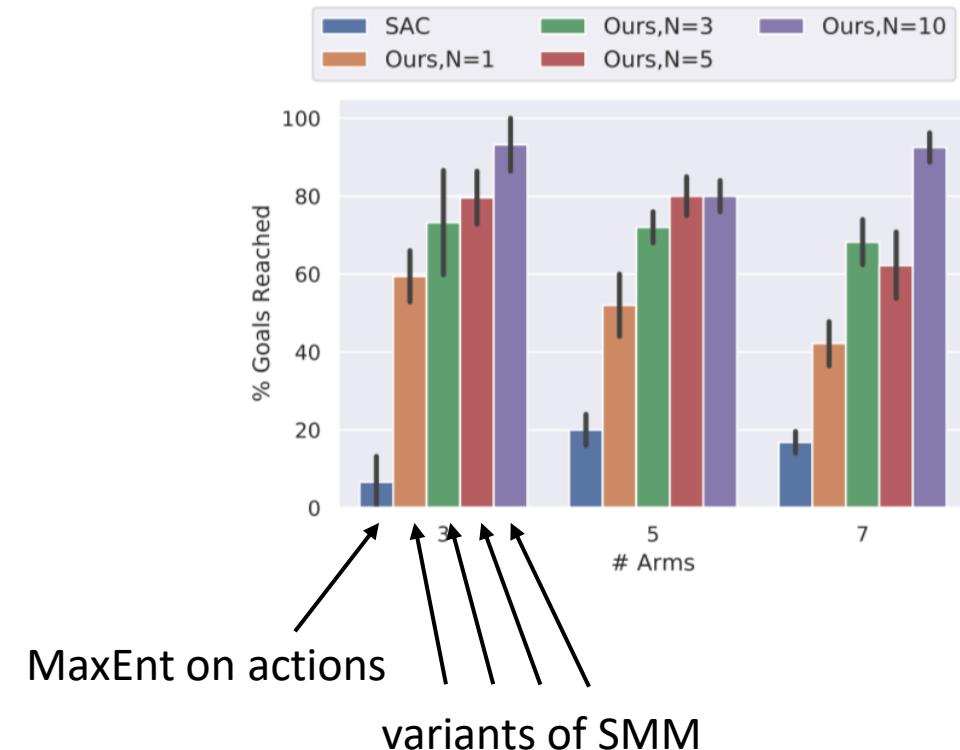
special case: $\log p^*(\mathbf{s}) = C \Rightarrow$ uniform target
 $D_{\text{KL}}(p_{\pi}(\mathbf{s}) \| U(\mathbf{s})) = \mathcal{H}(p_{\pi}(\mathbf{s}))$

State marginal matching for exploration

the state marginal matching problem: learn $\pi(\mathbf{a}|\mathbf{s})$ so as to minimize $D_{\text{KL}}(p_{\pi}(\mathbf{s}) \| p^*(\mathbf{s}))$



much better coverage!



In this lecture...

- Definitions & concepts from information theory
- Learning without a reward function by reaching goals
- A *state distribution-matching* formulation of reinforcement learning
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills*

Is state entropy *really* a good objective?

$$\text{Skew-Fit: } \max \mathcal{H}(p(G)) - \mathcal{H}(p(G|S)) = \max \mathcal{I}(S; G)$$
$$\text{SMM (special case where } p^*(s) = C\text{): } \max \mathcal{H}(p_\pi(S))$$

more or less the same thing

When is this a good idea?

“Eysenbach’s Theorem” (not really what it’s called)
(follows trivially from classic maximum entropy modeling)

at test time, an *adversary* will choose the *worst* goal G

which goal distribution should you use for *training*?

answer: choose $p(G) = \arg \max_p \mathcal{H}(p(G))$

See also: Hazan, Kakade, Singh, Van Soest. **Provably Efficient Maximum Entropy Exploration**

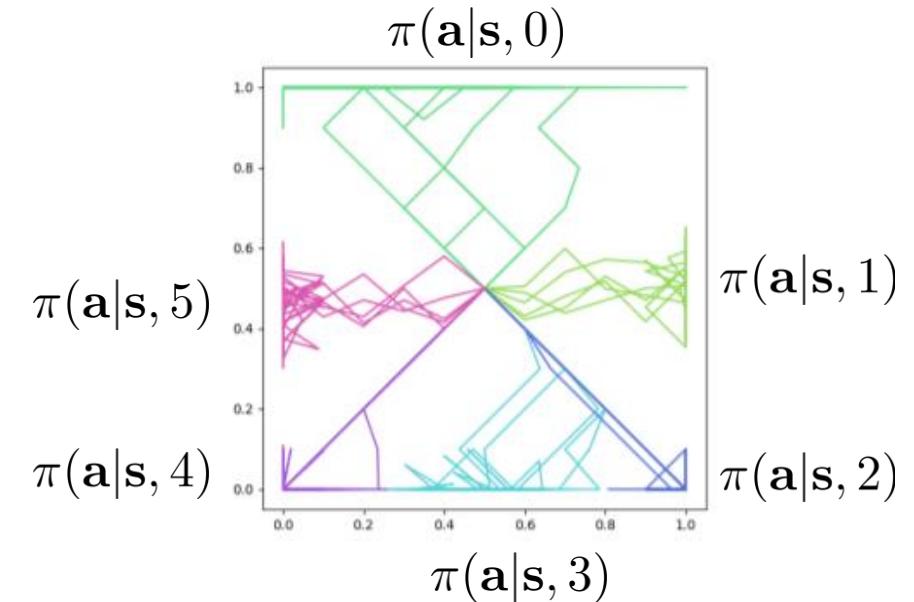
In this lecture...

- Definitions & concepts from information theory
- A *distribution-matching* formulation of reinforcement learning
- Learning without a reward function by reaching goals
- A *state distribution-matching* formulation of reinforcement learning
- Is coverage of valid states a *good* exploration objective?
- Beyond state covering: covering the *space of skills*

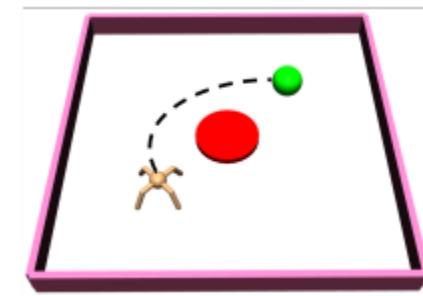
Learning diverse skills

$$\pi(a|s, z)$$

↑
task index



Reaching diverse **goals** is not the same as performing diverse **tasks**
not all behaviors can be captured by **goal-reaching**



Intuition: different **skills** should visit different **state-space regions**

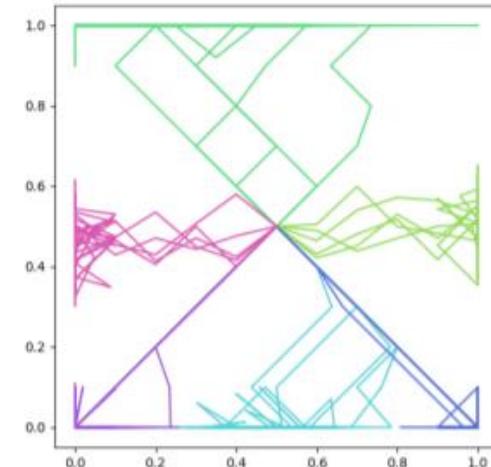
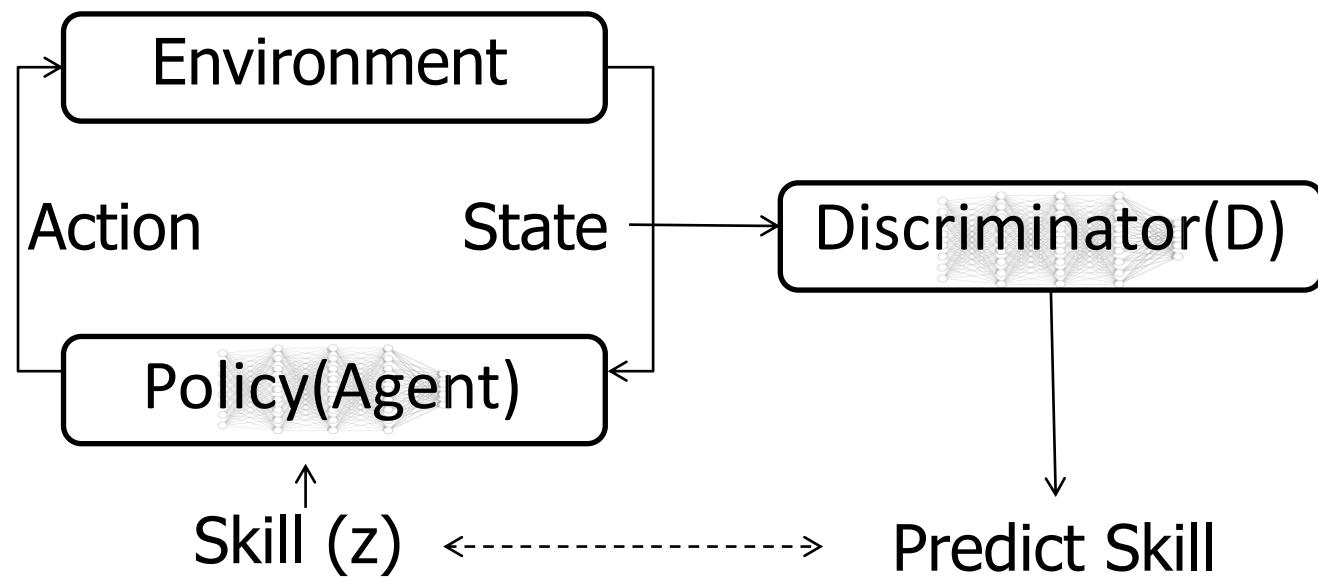
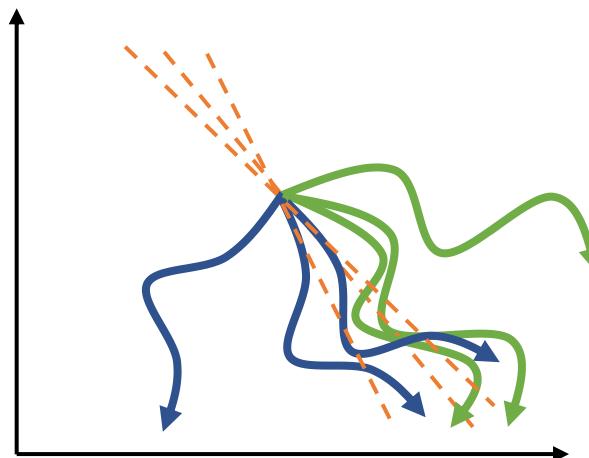
Diversity-promoting reward function

$$\pi(\mathbf{a}|\mathbf{s}, z) = \arg \max_{\pi} \sum_z E_{\mathbf{s} \sim \pi(\mathbf{s}|z)} [r(\mathbf{s}, z)]$$

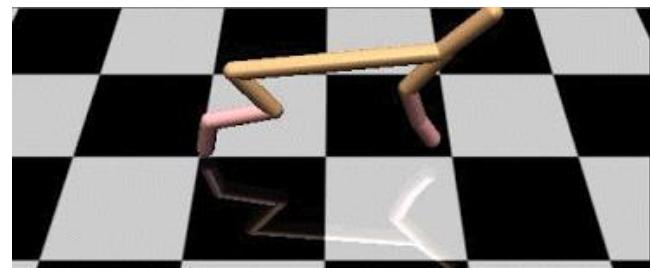
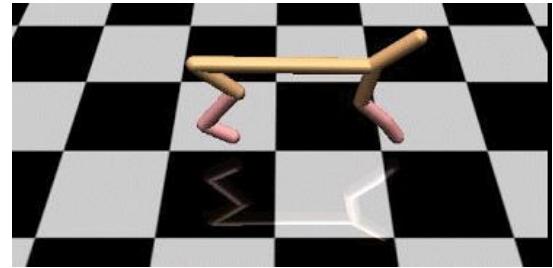


reward states that are unlikely for other $z' \neq z$

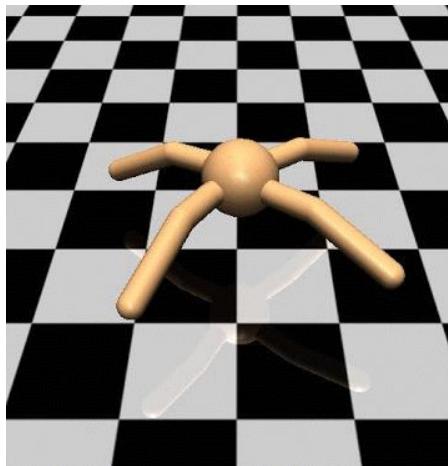
$$r(\mathbf{s}, z) = \log p(z|\mathbf{s})$$



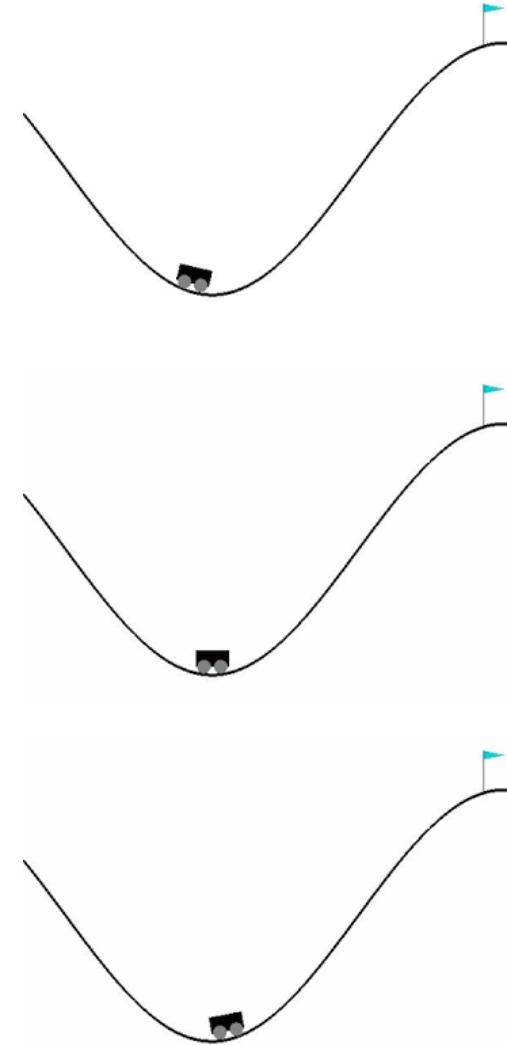
Examples of learned tasks



Cheetah



Ant



Mountain car

A connection to mutual information

$$\pi(\mathbf{a}|\mathbf{s}, z) = \arg \max_{\pi} \sum_z E_{\mathbf{s} \sim \pi(\mathbf{s}|z)}[r(\mathbf{s}, z)]$$

$$r(\mathbf{s}, z) = \log p(z|\mathbf{s})$$

$$I(z, \mathbf{s}) = H(z) - H(z|\mathbf{s})$$

maximized by using uniform prior $p(z)$

minimized by maximizing $\log p(z|\mathbf{s})$

Offline Reinforcement Learning

CS 285

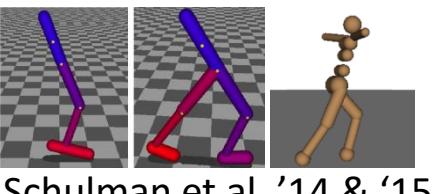
Instructor: Sergey Levine
UC Berkeley



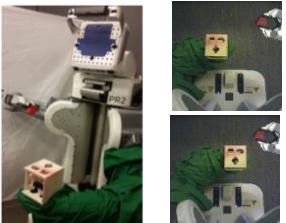
The generalization gap



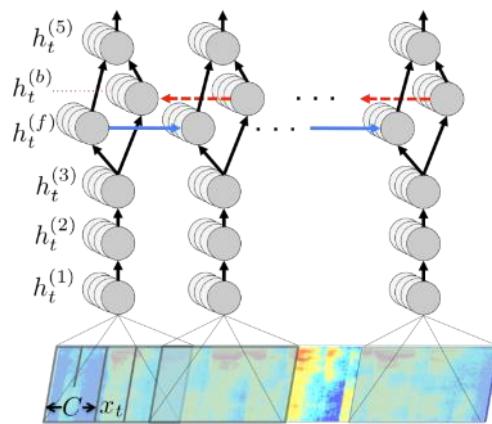
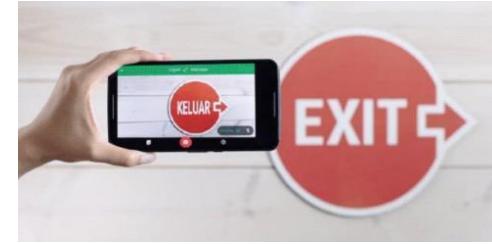
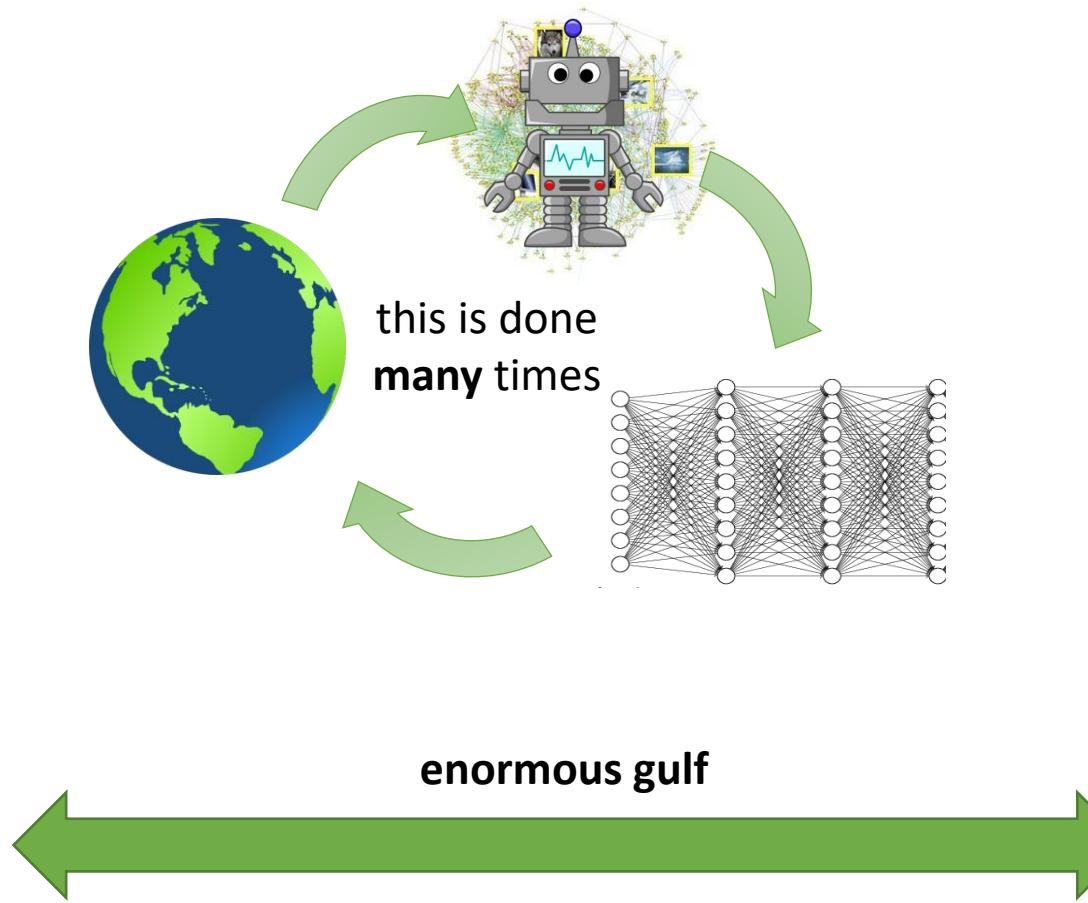
Mnih et al. '13



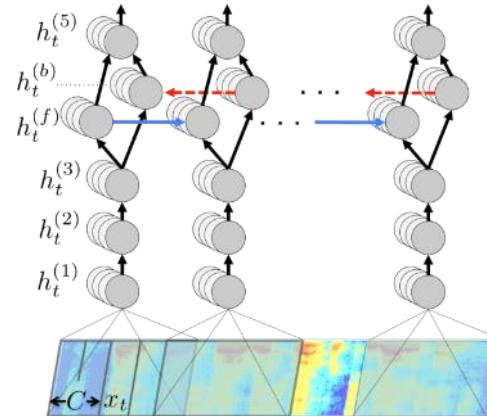
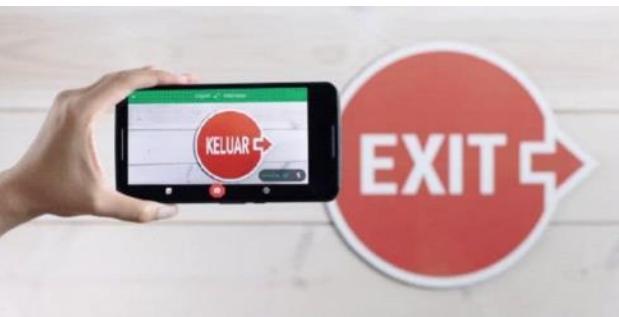
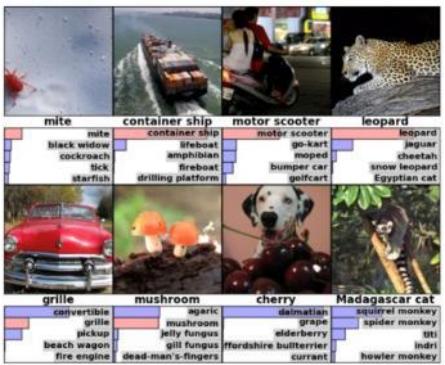
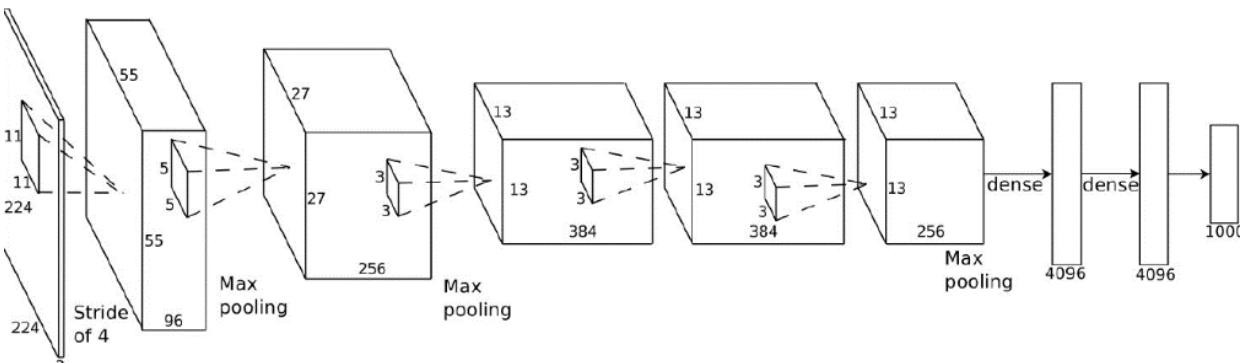
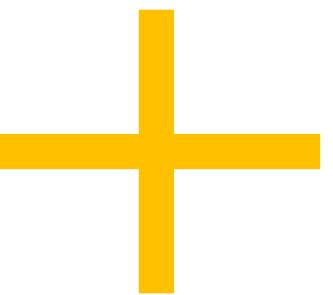
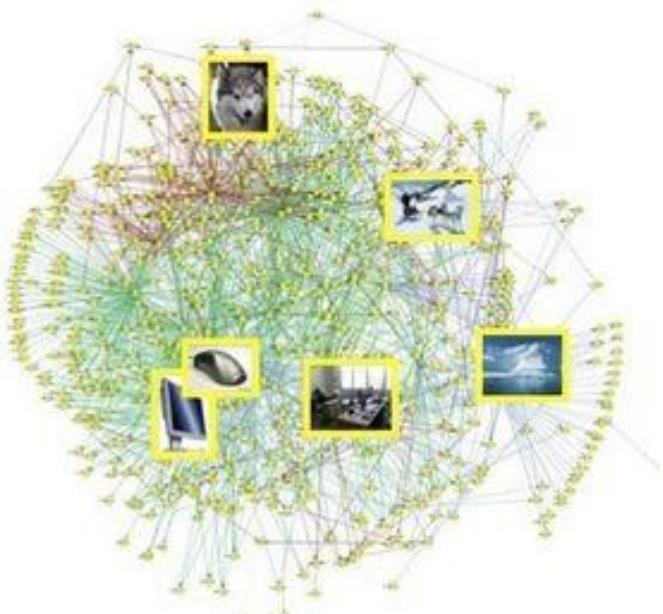
Schulman et al. '14 & '15



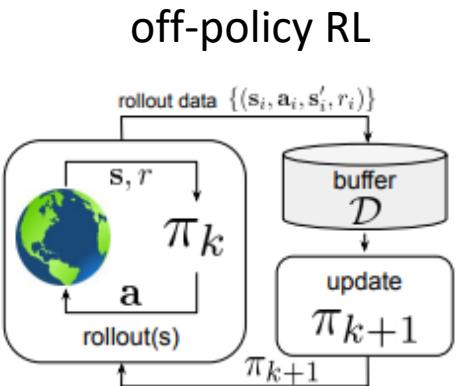
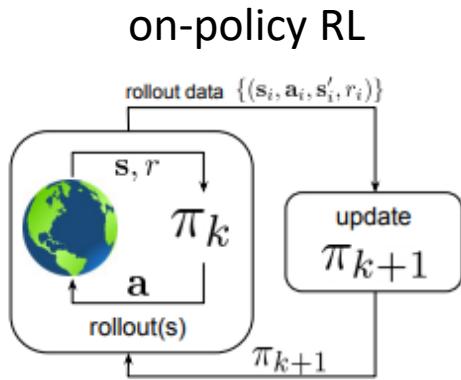
Levine*, Finn*, et al. '16



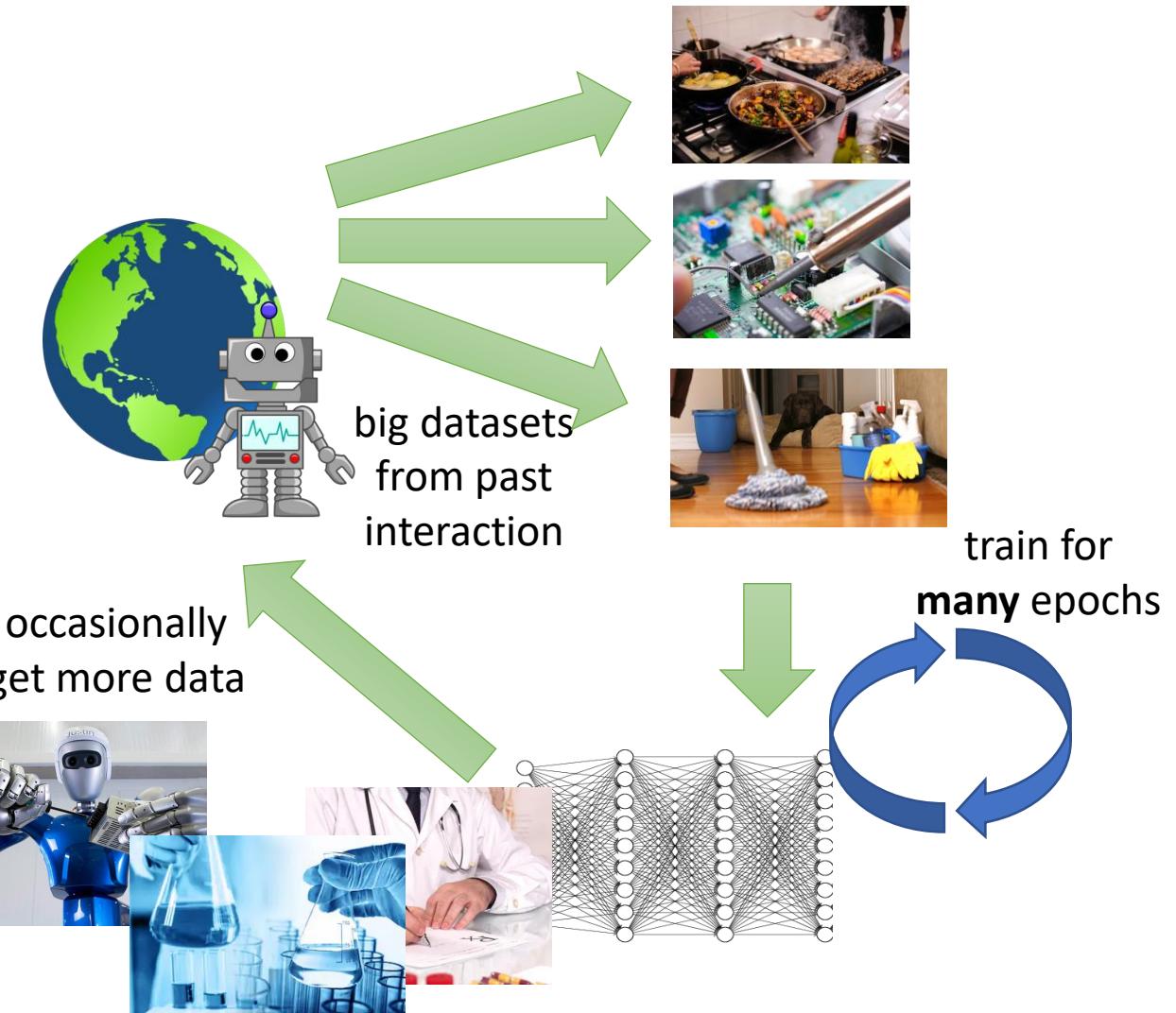
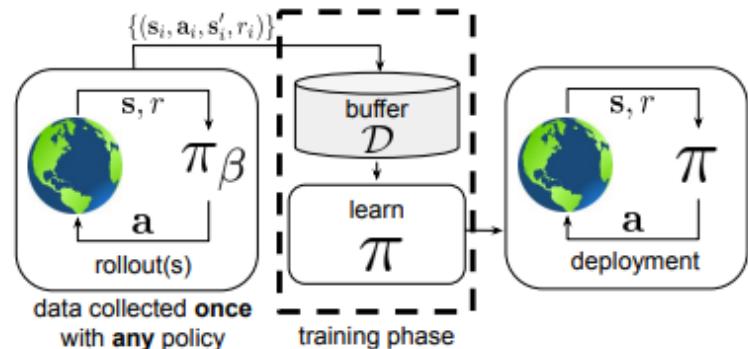
What makes modern machine learning work?



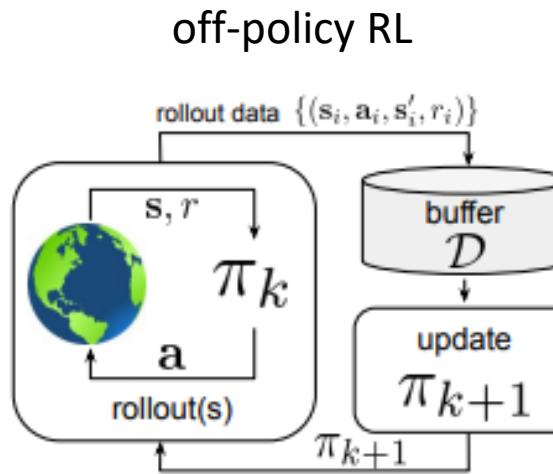
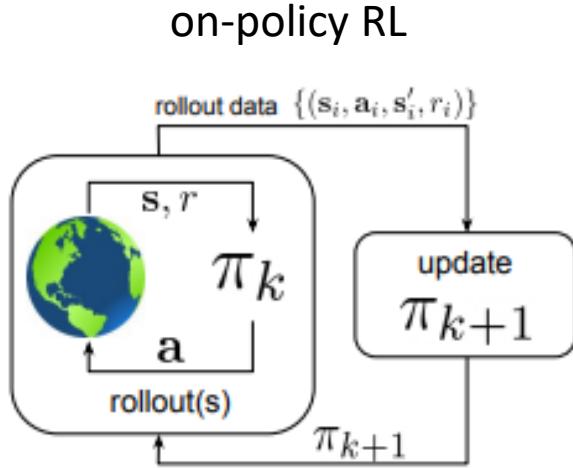
Can we develop data-driven RL methods?



offline reinforcement learning



What does offline RL mean?



Formally:

$$\mathcal{D} = \{(s_i, a_i, s'_i, r_i)\}$$

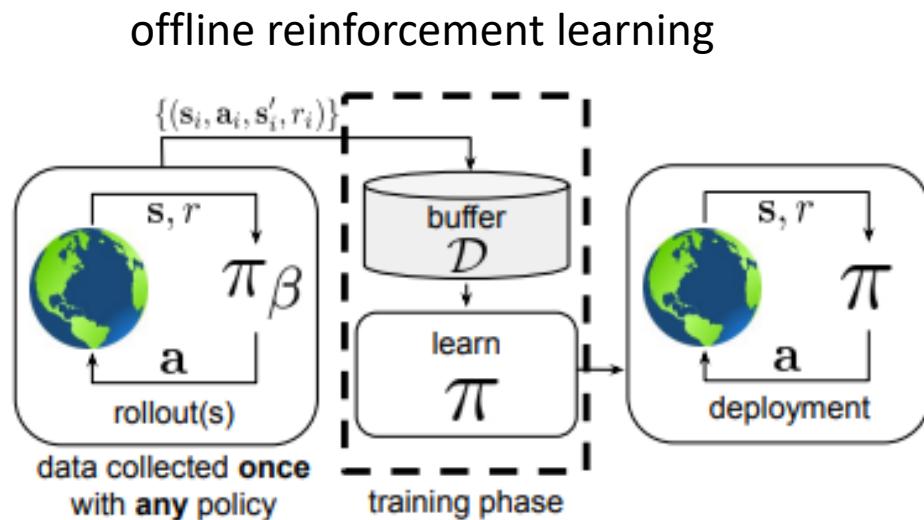
$$s \sim d^{\pi_\beta}(s)$$

$$a \sim \pi_\beta(a|s)$$

$$s' \sim p(s'|s, a)$$

$$r \leftarrow r(s, a)$$

generally **not** known



RL objective: $\max_{\pi} \sum_{t=0}^T E_{s_t \sim d^\pi(s), a_t \sim \pi(a|s)} [\gamma^t r(s_t, a_t)]$

Types of offline RL problems

$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$$

off-policy evaluation (OPE):

given \mathcal{D} , estimate $J(\pi) = E_{\pi} \left[\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right]$

$$\mathbf{s} \sim d^{\pi_{\beta}}(\mathbf{s})$$

$$\mathbf{a} \sim \pi_{\beta}(\mathbf{a}|\mathbf{s})$$

$$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$$

$$r \leftarrow r(\mathbf{s}, \mathbf{a})$$

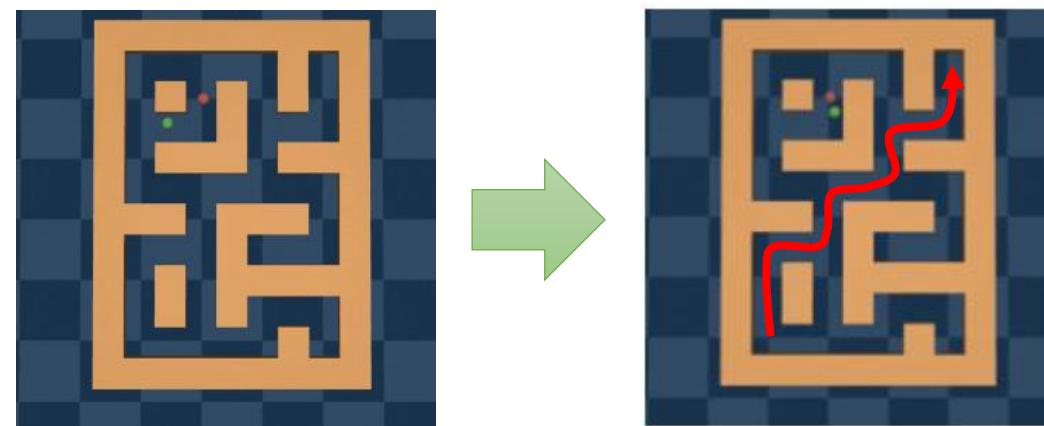
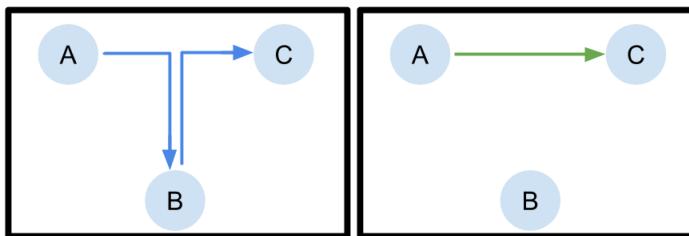
offline reinforcement learning: (a.k.a. batch RL, sometimes fully off-policy RL)

given \mathcal{D} , learn the best possible policy π_{θ}

not necessarily obvious what this means

How is this even possible?

1. Find the “good stuff” in a dataset full of good and bad behaviors
2. Generalization: good behavior in one place may suggest good behavior in another place
3. “Stitching”: parts of good behaviors can be recombined



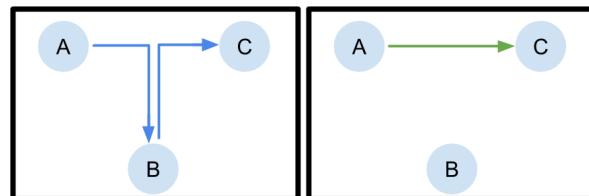
What do we expect offline RL methods to do?

Bad intuition: it's like imitation learning

Though it can be shown to be **provably** better than imitation learning even with optimal data, under some structural assumptions!

See: Kumar, Hong, Singh, Levine. Should I Run Offline Reinforcement Learning or Behavioral Cloning?

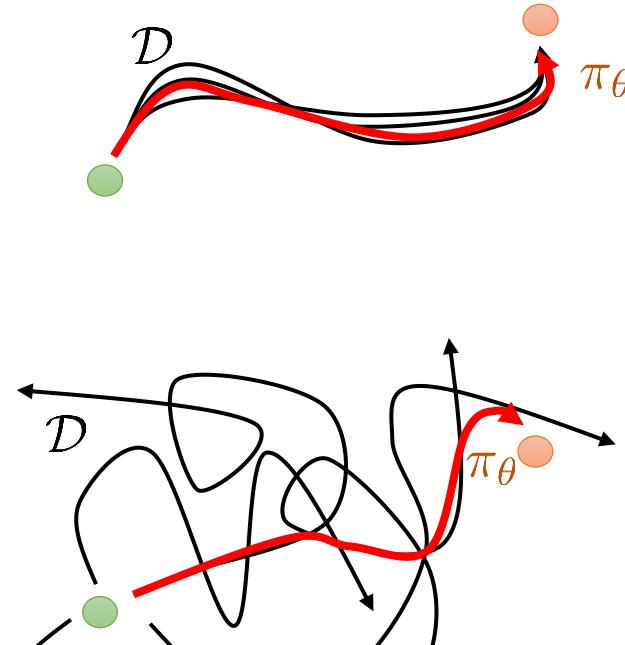
Better intuition: get order from chaos



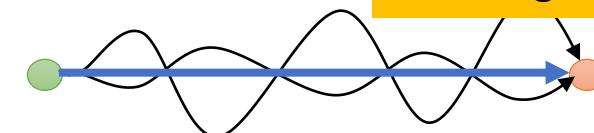
“Macro-scale” stitching

But this is just the clearest example!

“Micro-scale” stitching:



If we have algorithms that properly perform dynamic programming, we can take this idea much further and get near-optimal policies from highly suboptimal data



A vivid example

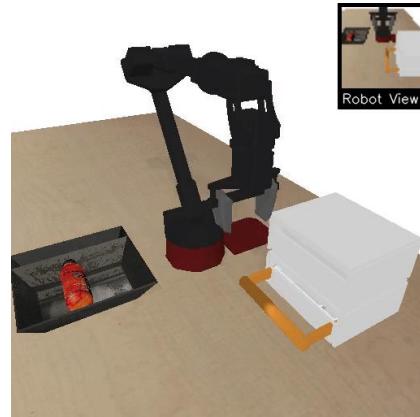
RL policies typically don't generalize to initial conditions that were not seen during training



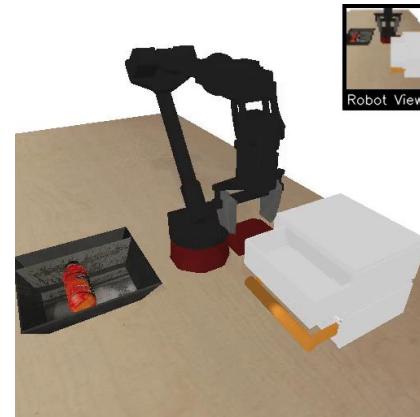
training task



closed drawer



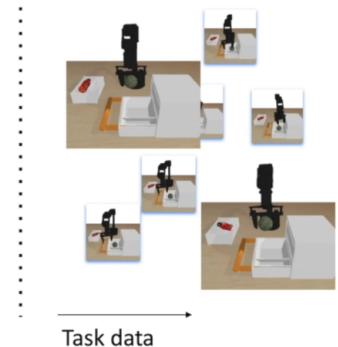
blocked by drawer



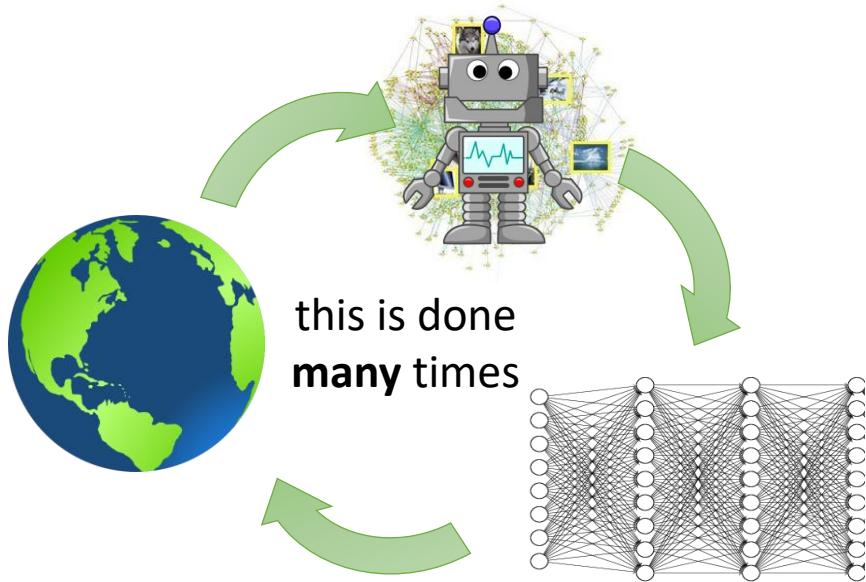
blocked by object



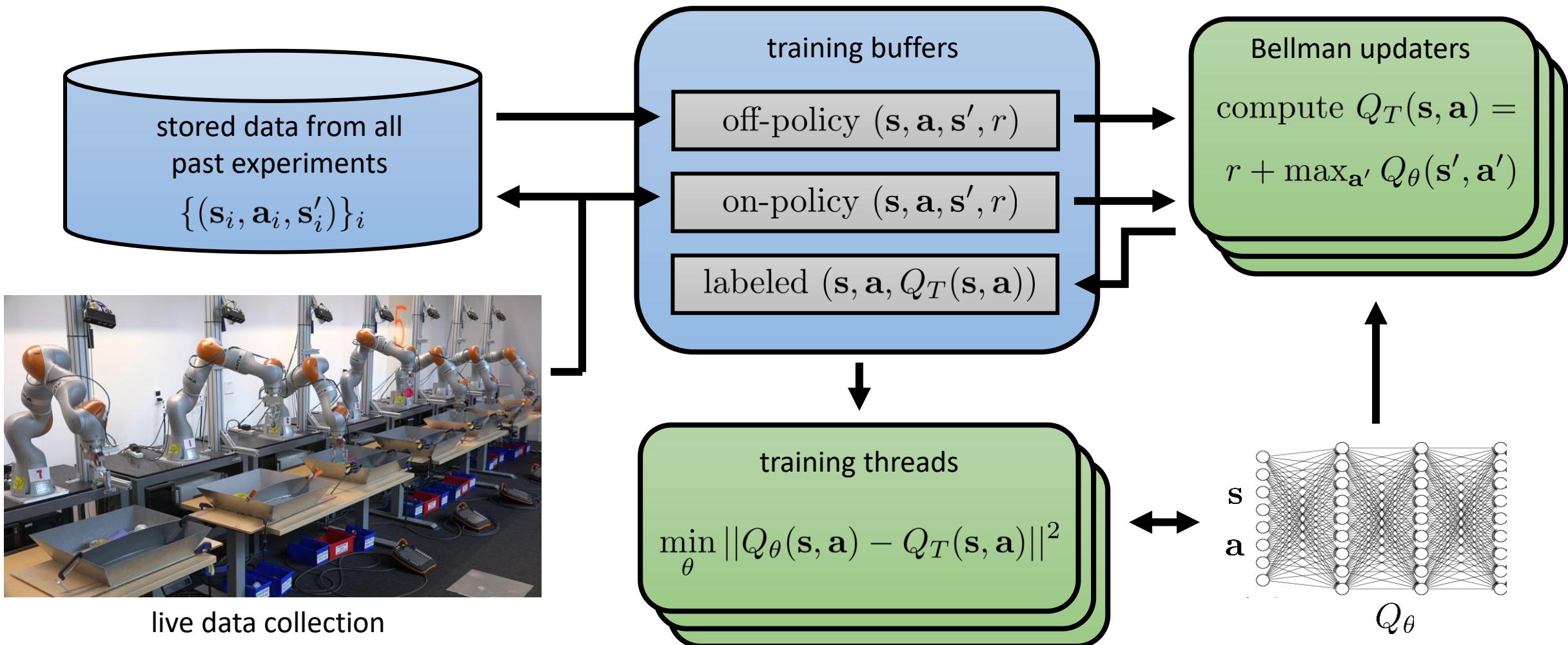
Can we use previously collected, unlabeled datasets to extend learned skills?



Why should we care?



Does it work?



Does it work?



2x



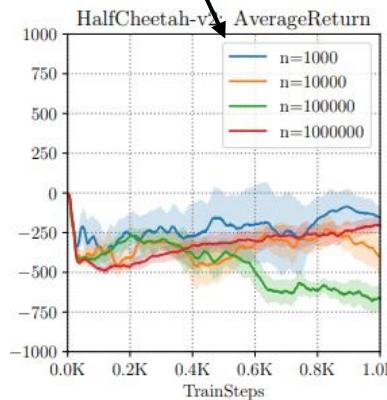
4x speed



Method	Dataset	Success	Failure
Offline QT-Opt	580k offline	87%	13%
Finetuned QT-Opt	580k offline + 28k online	96%	4%

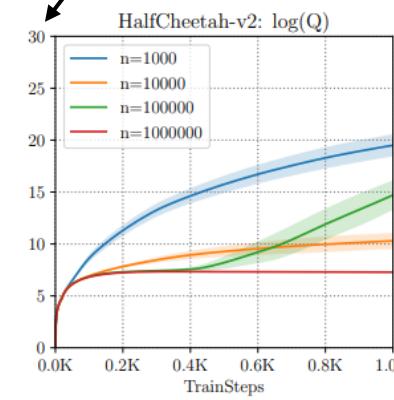
Why is offline RL hard?

amount of data



how well it does

log scale (massive overestimation)

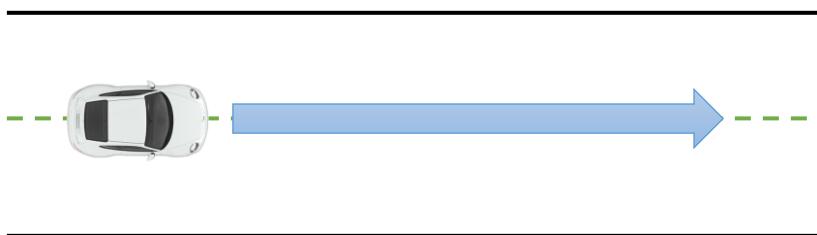


how well it *thinks* it does (Q-values)

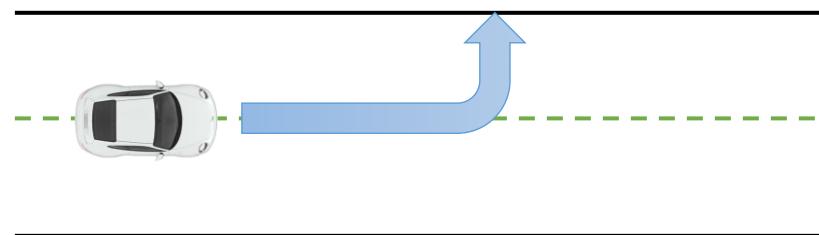
Why is offline RL hard?

Fundamental problem: counterfactual queries

Training data



What the policy wants to do



Is this good? Bad?

How do we know if
we didn't see it in
the data?

Online RL algorithms don't have to handle this, because they can simply **try** this action and see what happens

Offline RL methods must somehow account for these unseen (“out-of-distribution”) actions, ideally in a safe way
...while still making use of generalization to come up with behaviors that are better than the best thing seen in the data!

Distribution shift in a nutshell

Example empirical risk minimization (ERM) problem:

usually we are not worried – neural nets generalize well!

$$\theta \leftarrow \arg \min_{\theta} E_{\mathbf{x} \sim p(\mathbf{x}), y \sim p(y|\mathbf{x})} [(f_{\theta}(\mathbf{x}) - y)^2]$$

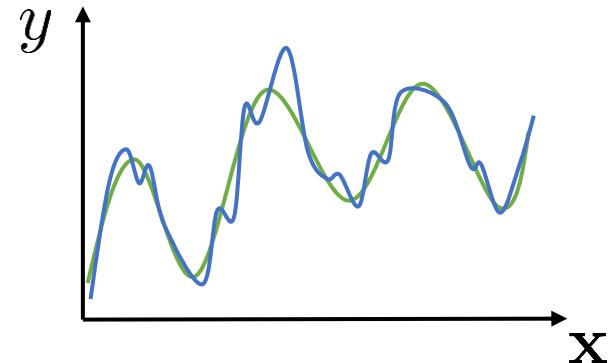
what if we pick $\mathbf{x}^* \leftarrow \arg \max_{\mathbf{x}} f_{\theta}(\mathbf{x})$?

given some \mathbf{x}^* , is $f_{\theta}(\mathbf{x}^*)$ correct?

$E_{\mathbf{x} \sim p(\mathbf{x}), y \sim p(y|\mathbf{x})} [(f_{\theta}(\mathbf{x}) - y)^2]$ is low

$E_{\mathbf{x} \sim \bar{p}(\mathbf{x}), y \sim p(y|\mathbf{x})} [(f_{\theta}(\mathbf{x}) - y)^2]$ is not, for general $\bar{p}(\mathbf{x}) \neq p(\mathbf{x})$

what if $\mathbf{x}^* \sim p(\mathbf{x})$? not necessarily...



Where do we suffer from distribution shift?

~~$Q(s, a) \leftarrow r(s, a) + \max_{a'} Q(s', a')$~~

$$Q(s, a) \leftarrow \underbrace{r(s, a) + E_{a' \sim \pi_{\text{new}}}[Q(s', a')]}_{y(s, a)}$$

what is the objective?

$$\min_Q E_{(s,a) \sim \pi_\beta(s,a)} [(Q(s, a) - y(s, a))^2]$$

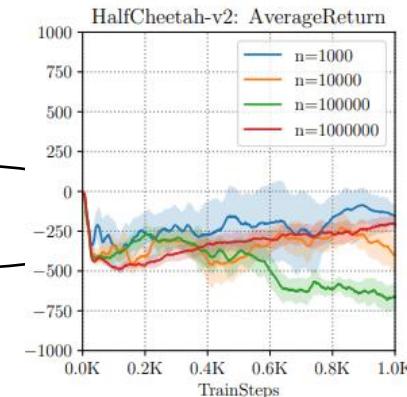
↑
behavior policy ↑
target value

expect good accuracy when $\pi_\beta(a|s) = \pi_{\text{new}}(a|s)$

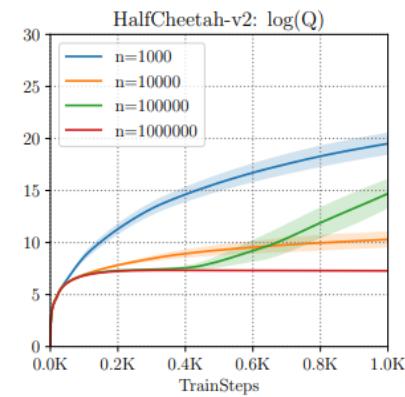
even worse: $\pi_{\text{new}} = \arg \max_\pi E_{a \sim \pi(a|s)}[Q(s, a)]$

(what if we pick $x^* \leftarrow \arg \max_x f_\theta(x)$?)

how often does *that* happen?



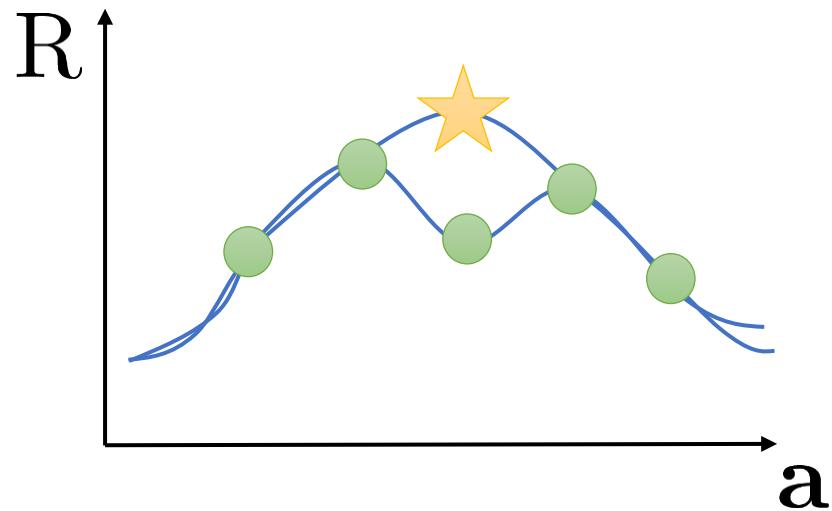
how well it does



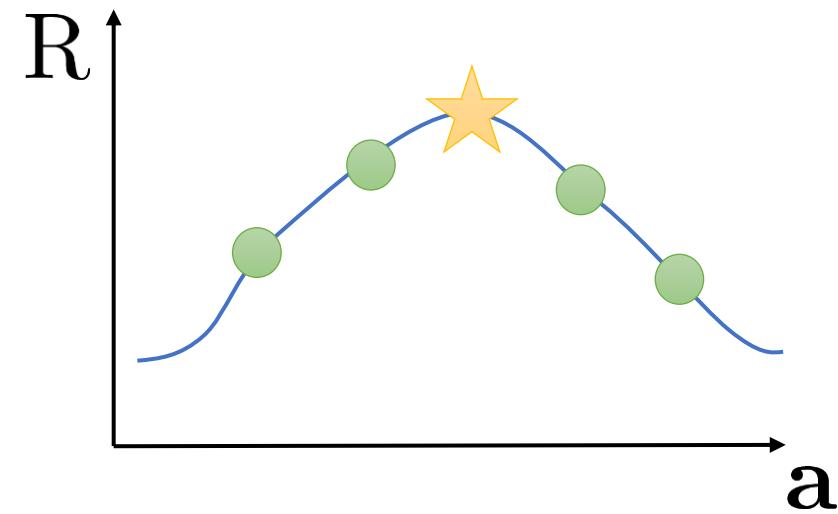
how well it *thinks* it does (Q-values)

Issues with generalization are not corrected

online RL setting



offline RL setting



Existing challenges with sampling error and function approximation error in standard RL become **much more severe** in offline RL

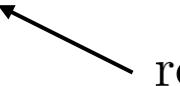
Batch RL via Importance Sampling

Offline RL with policy gradients

RL objective: $\max_{\pi} \sum_{t=0}^T E_{\mathbf{s}_t \sim d^\pi(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})} [\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)]$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)} \left[\sum_{t=0}^T \nabla_{\theta} \gamma^t \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \gamma^t \log \pi_{\theta}(\mathbf{a}_{t,i} | \mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

 requires sampling from π_{θ} ! what if we only have samples from π_{β} ?

importance sampling:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \underbrace{\frac{\pi_{\theta}(\tau_i)}{\pi_{\beta}(\tau_i)}}_{\text{importance weight}} \sum_{t=0}^T \nabla_{\theta} \gamma^t \log \pi_{\theta}(\mathbf{a}_{t,i} | \mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

importance weight

Offline RL with policy gradients

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \frac{\pi_{\theta}(\tau_i)}{\pi_{\beta}(\tau_i)} \sum_{t=0}^T \nabla_{\theta} \gamma^t \log \pi_{\theta}(\mathbf{a}_{t,i} | \mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

$$\frac{\pi_{\theta}(\tau)}{\pi_{\beta}(\tau)} = \frac{p(\mathbf{s}_1) \prod_t p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)}{p(\mathbf{s}_1) \prod_t p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \pi_{\beta}(\mathbf{a}_t | \mathbf{s}_t)}$$

can we fix this?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \underbrace{\left(\prod_{t'=0}^{t-1} \frac{\pi_{\theta}(\mathbf{a}_{t',i} | \mathbf{s}_{t',i})}{\pi_{\beta}(\mathbf{a}_{t',i} | \mathbf{s}_{t',i})} \right)}_{\text{accounts for difference in probability of landing in } \mathbf{s}_{t,i}} \nabla_{\theta} \gamma^t \log \pi_{\theta}(\mathbf{a}_{t,i} | \mathbf{s}_{t,i}) \underbrace{\left(\prod_{t'=t}^T \frac{\pi_{\theta}(\mathbf{a}_{t',i} | \mathbf{s}_{t',i})}{\pi_{\beta}(\mathbf{a}_{t',i} | \mathbf{s}_{t',i})} \right) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})}_{\text{accounts for having the incorrect } \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})}$$

accounts for difference in probability of landing in $\mathbf{s}_{t,i}$
 we have $\mathbf{s}_t \sim d^{\pi_{\beta}}(\mathbf{s}_t)$, but want $\mathbf{s}_t \sim d^{\pi_{\theta}}(\mathbf{s}_t)$

why is this a reasonable approximation?

$$E_{\pi_{\theta}} \left[\sum_{t'=t}^T \gamma^{t'-t} r_{t'} \right] \approx \sum_{t'=t}^T \gamma^{t'-t} r_{t',i}$$

this is exponential in T
 weights likely to be degenerate as T becomes large

Estimating the returns

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \gamma^t \log \pi_{\theta}(\mathbf{a}_{t,i} | \mathbf{s}_{t,i}) \left(\prod_{t'=t}^T \frac{\pi_{\theta}(\mathbf{a}_{t',i} | \mathbf{s}_{t',i})}{\pi_{\beta}(\mathbf{a}_{t',i} | \mathbf{s}_{t',i})} \right) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) \underbrace{\left[E_{\pi_{\theta}} \left[\sum_{t'=t}^T \gamma^{t'-t} r_{t'} \right] \right]}_{\sum_{t'=t}^T \left(\prod_{t''=t}^{t-1} \frac{\pi_{\theta}(\mathbf{a}_{t'',i} | \mathbf{s}_{t'',i})}{\pi_{\beta}(\mathbf{a}_{t'',i} | \mathbf{s}_{t'',i})} \right) \gamma^{t'-t} r_{t',i}}$$

but this is *still* exponential!

To avoid exponentially exploding importance weights, we **must** use value function estimation!

imagine we knew $Q^{\pi_{\theta}}(\mathbf{s}, \mathbf{a})$

We'll see how to do this shortly, but first let's conclude our discussion of importance sampling

The doubly robust estimator

$$V^{\pi_\theta}(\mathbf{s}_0) \approx \sum_{t=0}^T \left(\prod_{t'=0}^{t'} \frac{\pi_\theta((\mathbf{a}_{tt'} | \mathbf{s}_{it} | \mathbf{s}_{t'}), i)}{\pi_\beta((\mathbf{a}_{tt'} | \mathbf{s}_{it} | \mathbf{s}_{t'}), i)} \right) r_t \gamma^{t' - t} r_{t', i}$$

$$= \sum_{t=0}^T \left(\prod_{t'=0}^t \rho_{t'} \right) \gamma^t r_t$$

$$= \rho_0 r_0 + \rho_0 \gamma \rho_1 r_1 + \rho_0 \gamma \rho_1 \gamma \rho_2 r_2 + \dots$$

$$= \rho_0 (r_0 + \gamma (\rho_1 (r_1 + \gamma (\rho_2 (r_2 + \gamma \dots)))))$$

$$= \bar{V}^T \quad \text{where } \bar{V}^{T+1-t} = \rho_t (r_t + \gamma \bar{V}^{T-t})$$

$$\bar{V}_{\text{DR}}^{T+1-t} = \hat{V}(\mathbf{s}_t) + \rho_t (r_t + \gamma \bar{V}_{\text{DR}}^{T-t} - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t))$$

model or function approximator

this is exponential!

doubly robust estimation (bandit case)

$$V_{\text{DR}}(s) = \hat{V}(s) + \rho(s, a)(r_{s,a} - \hat{Q}(s, a))$$

model or function approximator

Marginalized importance sampling

Main idea: instead of using $\prod_t \frac{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\beta(\mathbf{a}_t | \mathbf{s}_t)}$, estimate $w(\mathbf{s}, \mathbf{a}) = \frac{d^{\pi_\theta}(\mathbf{s}, \mathbf{a})}{d^{\pi_\beta}(\mathbf{s}, \mathbf{a})}$

if we can do this, we can estimate $J(\theta) \approx \frac{1}{N} \sum_i w(\mathbf{s}_i, \mathbf{a}_i) r_i$

typically this is done for off-policy evaluation, rather than policy learning

how to determine $w(\mathbf{s}, \mathbf{a})$? typically solve some kind of consistency condition

example (Zhang et al., GenDICE):

$$d^{\pi_\beta}(\mathbf{s}', \mathbf{a}') w(\mathbf{s}', \mathbf{a}') = \underbrace{(1-\gamma)p_0(\mathbf{s}')\pi_\theta(\mathbf{a}'|\mathbf{s}')}_{\text{probability of starting in } (\mathbf{s}', \mathbf{a}')} + \gamma \underbrace{\sum_{\mathbf{s}, \mathbf{a}} \pi_\theta(\mathbf{a}'|\mathbf{s}') p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) d^{\pi_\beta}(\mathbf{s}, \mathbf{a}) w(\mathbf{s}, \mathbf{a})}_{\text{probability of transitioning into } (\mathbf{s}', \mathbf{a}')}$$

solving for $w(\mathbf{s}, \mathbf{a})$ typically involves some fixed point problem

Additional readings: importance sampling

Classic work on importance sampled policy gradients and return estimation:

Precup, D. (2000). Eligibility traces for off-policy policy evaluation.

Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience.

Doubly robust estimators and other improved importance-sampling estimators:

Jiang, N. and Li, L. (2015). Doubly robust off-policy value evaluation for reinforcement learning.

Thomas, P. and Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning.

Analysis and theory:

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. (2015). High-confidence off-policy evaluation.

Marginalized importance sampling:

Hallak, A. and Mannor, S. (2017). Consistent on-line off-policy evaluation.

Liu, Y., Swaminathan, A., Agarwal, A., and Brunskill, E. (2019). Off-policy policy gradient with state distribution correction.

Additional readings in our offline RL survey: Section 3.1, 3.2, 3.3, 3.4: <https://arxiv.org/abs/2005.01643>

Batch RL via Linear Fitted Value Functions

Offline value function estimation

How have people thought about it before?

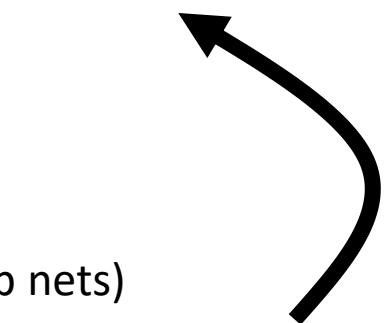
Extend existing ideas for approximate dynamic programming and Q-learning to offline setting

Derive tractable solutions with simple (e.g., linear) function approximators

How are people thinking about it now?

Derive approximate solutions with highly expressive function approximators (e.g., deep nets)

The primary challenge turns out to be **distributional shift**



generally not concerned with
distributional shift before

(maybe it was not such a big
problem with linear models)

We'll discuss some older offline/batch RL methods
next for completeness

Warmup: linear models

Φ – feature matrix, $|S| \times K$

could also think of as a vector-valued function $\Phi(\mathbf{s})$

Can we do (offline) model-based RL in feature space?

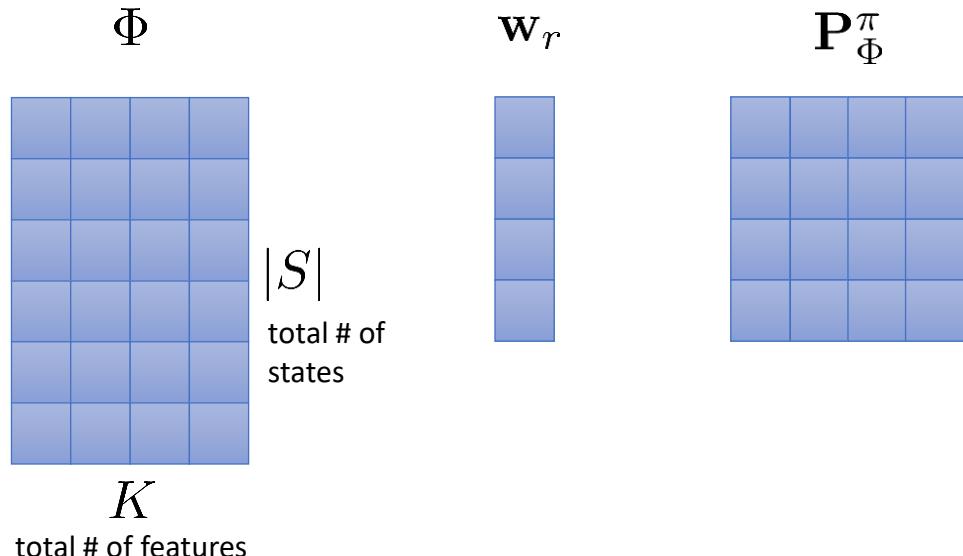
1. Estimate the reward
2. Estimate the transitions
3. Recover the value function
4. Improve the policy

1. Reward model: $\Phi \mathbf{w}_r \approx r$

2. Transition model: $\Phi \mathbf{P}_\Phi \approx \mathbf{P}^\pi \Phi$

estimated feature-space
transition matrix
 $K \times K$

real transition matrix
(on states)
 $|S| \times |S|$



least squares: $\mathbf{w}_r = (\Phi^T \Phi)^{-1} \Phi^T \vec{r}$

least squares: $\mathbf{P}_\Phi = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{P}^\pi \Phi$

all of this is for a *fixed* policy π

Recovering the value function

1. Reward model: $\Phi \mathbf{w}_r \approx r$

$$\text{least squares: } \mathbf{w}_r = (\Phi^T \Phi)^{-1} \Phi \vec{\mathbf{r}}$$

2. Transition model: $\Phi \mathbf{P}_\Phi \approx \mathbf{P}^\pi \Phi$

$$\text{least squares: } \mathbf{P}_\Phi = (\Phi^T \Phi)^{-1} \Phi \mathbf{P}^\pi \Phi$$

3. Estimate $V^\pi \approx V_\Phi^\pi = \Phi \mathbf{w}_V$

can apply the same equation in feature space:

$$\mathbf{w}_V = (\mathbf{I} - \gamma \mathbf{P}_\Phi)^{-1} \mathbf{w}_r$$

← substitute

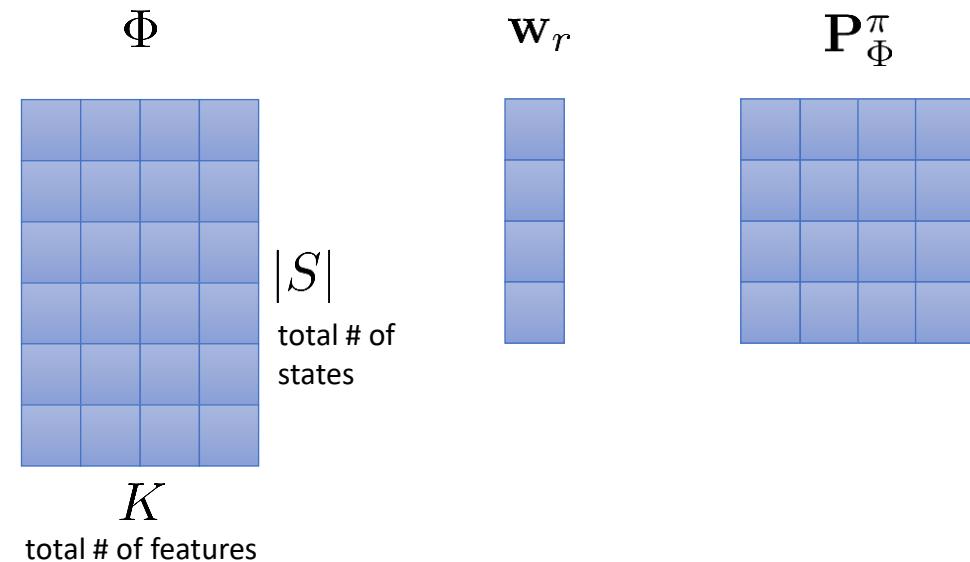
but wait – do we even *need* the model?

$$\mathbf{w}_V = (\mathbf{I} - \gamma(\Phi^T \Phi)^{-1} \Phi^T \mathbf{P}^\pi \Phi)^{-1} (\Phi^T \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$

after a bit of algebra...

$$\mathbf{w}_V = (\Phi^T \Phi - \gamma \Phi^T \mathbf{P}^\pi \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$

this is called least-squares temporal difference (LSTD)



Aside: solving for V^π in terms of \mathbf{P}^π and \mathbf{r} :

$$\begin{aligned} V^\pi &= \mathbf{r} + \gamma \mathbf{P}^\pi V^\pi \\ (\mathbf{I} - \gamma \mathbf{P}^\pi) V^\pi &= \mathbf{r} \\ V^\pi &= (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{r} \end{aligned}$$

Doing it all with samples

$$\mathbf{w}_V = (\Phi^T \Phi - \gamma \Phi^T \mathbf{P}^\pi \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$

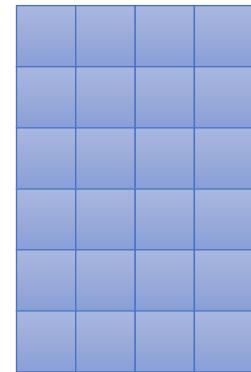
$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$$

replace with Φ'

$$\Phi'_i = \phi(\mathbf{s}'_i)$$

$$\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$$

$$\Phi$$



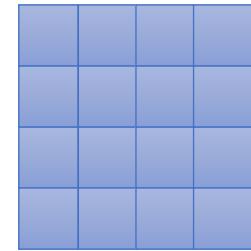
K
total # of features

$$\Phi_i = \phi(\mathbf{s}_i)$$

$$\mathbf{w}_r$$



$$\mathbf{P}_\Phi^\pi$$



$|\mathcal{D}|$
total # of sample

Everything else works **exactly** the same way, only now we have some sampling error

Improving the policy

1. Estimate the reward
2. Estimate the transitions
3. Recover the value function
4. Improve the policy

or just do these together with LSTD!

typical policy improvement step:

~~→ $\pi'(\mathbf{s}) \leftarrow \text{Greedy}(\Phi \mathbf{w}_V)$~~

~~estimate $V^{\pi'}$~~

That's not going to work for offline RL!

$$\mathbf{w}_V = (\Phi^T \Phi - \gamma \Phi^T \mathbf{P}^\pi \Phi)^{-1} \Phi^T \vec{\mathbf{r}}$$
$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$$

↑ ↑
 $\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$

replace with Φ'
 $\Phi'_i = \phi(\mathbf{s}'_i)$

this requires samples from π !

Least-squares policy iteration (LSPI)

Main idea: replace LSTD with LSTDQ – LSTD but for Q-functions

$$\mathbf{w}_Q = (\Phi^T \Phi - \gamma \Phi^T \Phi')^{-1} \Phi^T \vec{\mathbf{r}}$$

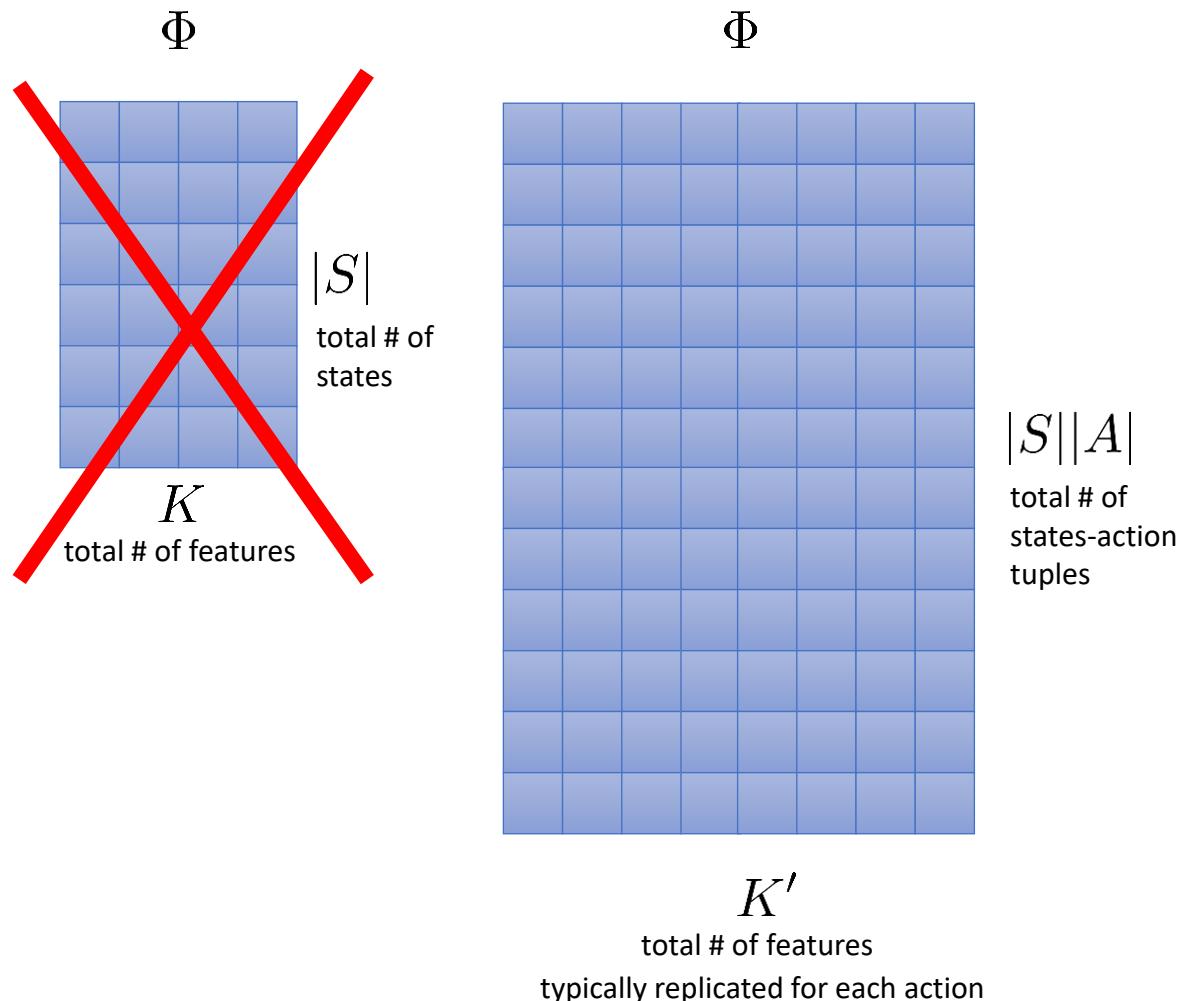
$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$$

$$\Phi'_i = \phi(\mathbf{s}'_i, \pi(\mathbf{s}'_i))$$

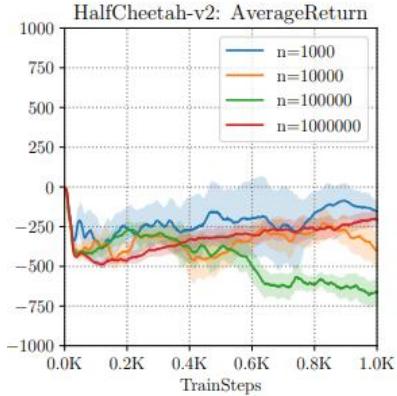
encode the action π would take
not the action in the data

LSPI:

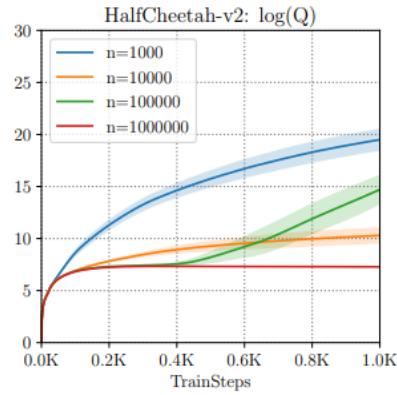
1. compute \mathbf{w}_Q for π_k
2. $\pi_{k+1}(\mathbf{s}) = \arg \max_{\mathbf{a}} \phi(\mathbf{s}, \mathbf{a}) \mathbf{w}_Q$
3. Set $\Phi'_i = \phi(\mathbf{s}'_i, \pi_{k+1}(\mathbf{s}'_i))$



What's the issue?



how well it does



how well it *thinks* it does (Q-values)

$$Q(\mathbf{s}, \mathbf{a}) \leftarrow \underbrace{r(\mathbf{s}, \mathbf{a}) + E_{\mathbf{a}' \sim \pi_{\text{new}}} [Q(\mathbf{s}', \mathbf{a}')]}$$

$y(\mathbf{s}, \mathbf{a})$

expect good accuracy when $\pi_\beta(\mathbf{a}|\mathbf{s}) = \pi_{\text{new}}(\mathbf{a}|\mathbf{s})$

even *worse*: $\pi_{\text{new}} = \arg \max_\pi E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})]$

In general, all approximate dynamic programming (e.g., fitted value/Q iteration) methods will suffer from action distributional shift, and we **must** fix it!

$$\min_Q E_{(\mathbf{s}, \mathbf{a}) \sim \pi_\beta(\mathbf{s}, \mathbf{a})} [(Q(\mathbf{s}, \mathbf{a}) - y(\mathbf{s}, \mathbf{a}))^2]$$

↑
behavior policy ↑
target value

how often does *that* happen?

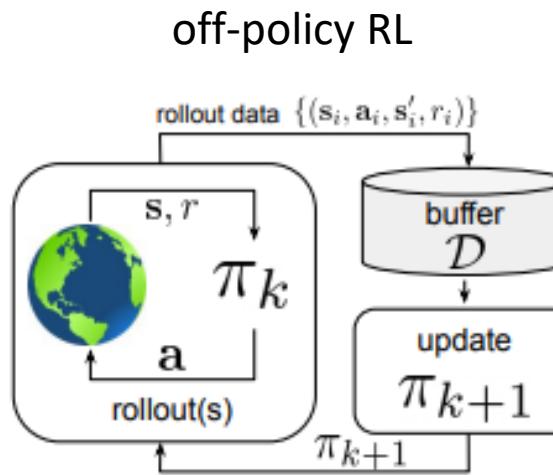
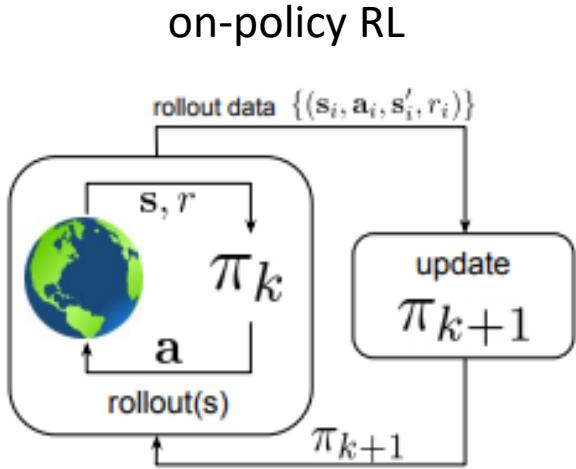
Offline Reinforcement Learning Part 2

CS 285

Instructor: Sergey Levine
UC Berkeley



Offline Reinforcement Learning



Formally:

$$\mathcal{D} = \{(s_i, a_i, s'_i, r_i)\}$$

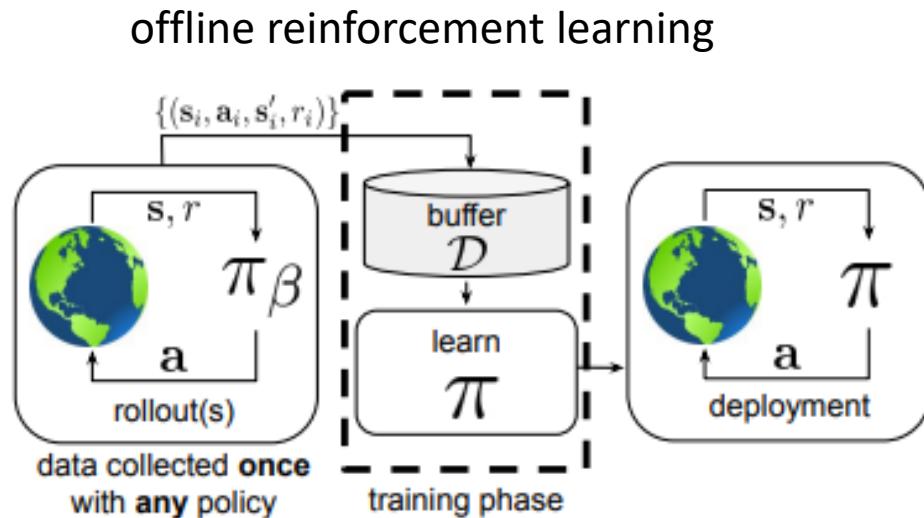
$$s \sim d^{\pi_\beta}(s)$$

$$a \sim \pi_\beta(a|s)$$

$$s' \sim p(s'|s, a)$$

$$r \leftarrow r(s, a)$$

generally **not** known



RL objective: $\max_{\pi} \sum_{t=0}^T E_{s_t \sim d^\pi(s), a_t \sim \pi(a|s)} [\gamma^t r(s_t, a_t)]$

Where do we suffer from distribution shift?

$$\cancel{Q(s, a) \leftarrow r(s, a) + \max_{a'} Q(s', a')}$$

$$Q(s, a) \leftarrow \underbrace{r(s, a) + E_{a' \sim \pi_{\text{new}}}[Q(s', a')]}_{y(s, a)}$$

what is the objective?

$$\min_Q E_{(s,a) \sim \pi_\beta(s,a)} [(Q(s, a) - y(s, a))^2]$$

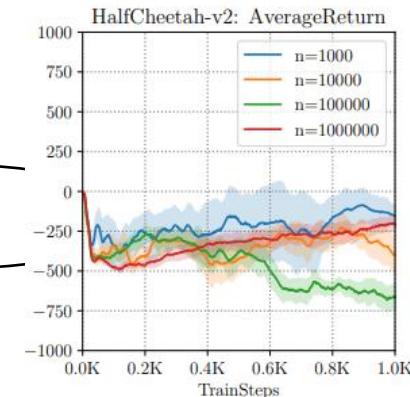
↑
behavior policy ↑
target value

expect good accuracy when $\pi_\beta(a|s) = \pi_{\text{new}}(a|s)$

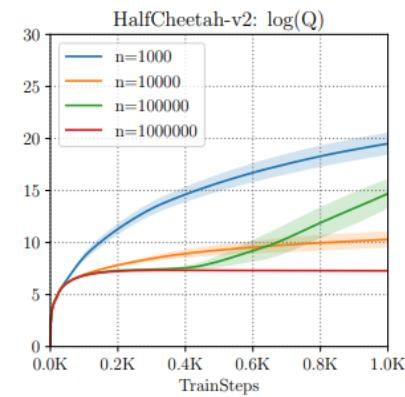
even worse: $\pi_{\text{new}} = \arg \max_\pi E_{a \sim \pi(a|s)}[Q(s, a)]$

(what if we pick $x^* \leftarrow \arg \max_x f_\theta(x)$?)

how often does *that* happen?

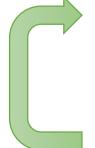


how well it does



how well it *thinks* it does (Q-values)

How do prior methods address this?


$$Q(s, a) \leftarrow r(s, a) + E_{a' \sim \pi_{\text{new}}} [Q(s', a')]$$


$$\pi_{\text{new}}(a|s) = \arg \max_{\pi} E_{a \sim \pi(a|s)} [Q(s, a)] \text{ s.t. } D_{\text{KL}}(\pi \| \pi_{\beta}) \leq \epsilon$$

This solves distribution shift, right?

No more erroneous values?

Issue 1: we usually don't know the behavior policy $\pi_{\beta}(a|s)$

- human-provided data
- data from hand-designed controller
- data from many past RL runs
- all of the above

Issue 2: this is both *too pessimistic* and *not pessimistic enough*

“policy constraint” method

very old idea (but it had no single name?)

Todorov et al. [passive dynamics in linearly-solvable MDPs]

Kappen et al. [KL-divergence control, etc.]

trust regions, covariant policy gradients, natural policy gradients, etc.

used in some form in recent papers:

Fox et al. '15 (“Taming the Noise...”)

Fujimoto et al. '18 (“Off Policy...”)

Jaques et al. '19 (“Way Off Policy...”)

Kumar et al. '19 (“Stabilizing...”)

Wu et al. '19 (“Behavior Regularized...”)

Explicit policy constraint methods

What kinds of constraints can we use?

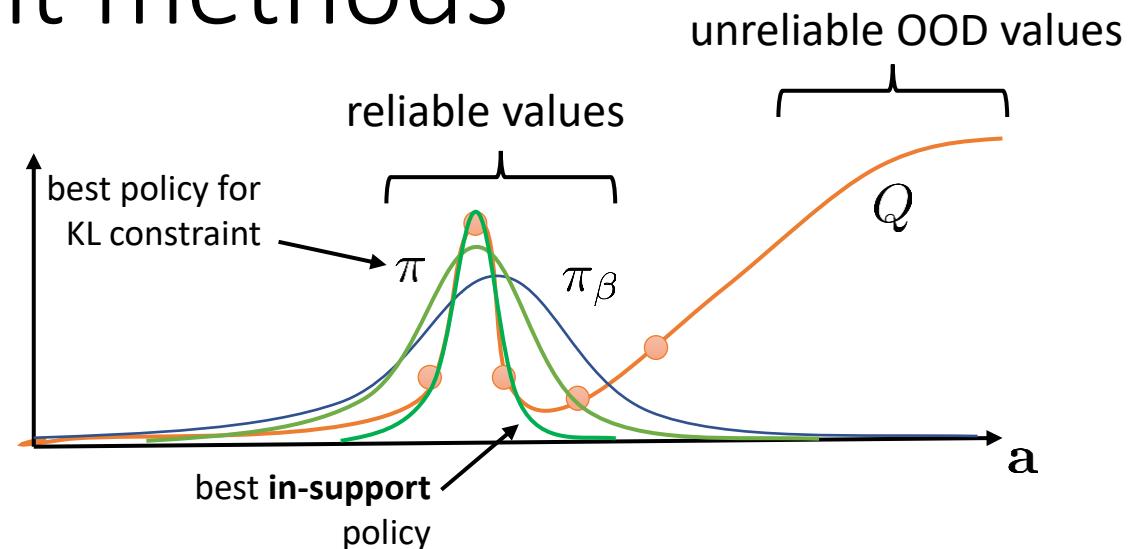
KL-divergence: $D_{\text{KL}}(\pi \parallel \pi_\beta)$

- + easy to implement (more on this later)
- not necessarily what we want

support constraint: $\pi(a|s) \geq 0$ only if $\pi_\beta(a|s) \geq \epsilon$

can approximate with MMD

- significantly more complex to implement
- + much closer to what we really want



For more information, see:

Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** '20

Kumar, Fu, Tucker, Levine. **Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction.** '19

Wu, Tucker, Nachum. **Behavior Regularized Offline Reinforcement Learning.** '19

Explicit policy constraint methods

How do we implement constraints?

1. Modify the actor objective

$$\begin{aligned} \theta &\leftarrow \arg \max_{\theta} E_{s \sim D} [E_{a \sim \pi_{\theta}(a|s)}[Q(s, a)]] \\ \theta &\leftarrow \arg \max_{\theta} E_{s \sim D} [E_{a \sim \pi_{\theta}(a|s)}[Q(s, a) + \lambda \log \pi_{\beta}(a|s)] + \lambda \mathcal{H}(\pi(a|s))] \end{aligned}$$

Lagrange multiplier

easy to compute and differentiate
for Gaussian or categorical policies

2. Modify the reward function

$$\bar{r}(s, a) = r(s, a) - D(\pi, \pi_{\beta})$$

simple modification to directly penalize divergence
also accounts for **future** divergence

See: Wu, Tucker, Nachum. **Behavior Regularized Offline Reinforcement Learning.** '19

generally, the best modern offline RL methods do not do either of these things

Implicit policy constraint methods

$$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \arg \max_{\pi} E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s}, \mathbf{a})] \text{ s.t. } D_{\text{KL}}(\pi \| \pi_{\beta}) \leq \epsilon$$

$$\pi^*(\mathbf{a}|\mathbf{s}) = \frac{1}{Z(\mathbf{s})} \pi_{\beta}(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi}(\mathbf{s}, \mathbf{a})\right) \quad \text{straightforward to show via duality}$$

See also:

Peters et al. (REPS)

Rawlik et al. ("psi-learning")

...many follow-ups

approximate via **weighted** max likelihood!

$$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \arg \max_{\pi} E_{(\mathbf{s}, \mathbf{a}) \sim \pi_{\beta}} \left[\log \pi(\mathbf{a}|\mathbf{s}) \frac{1}{Z(\mathbf{s})} \exp\left(\frac{1}{\lambda} A^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a})\right) \right]$$

$w(\mathbf{s}, \mathbf{a})$

↑
samples from dataset
 $\mathbf{a} \sim \pi_{\beta}(\mathbf{a}|\mathbf{s})$

critic can be used
to give us this

Implicit policy constraint methods

$$\mathcal{L}_C(\phi) = E_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim D} \left[(Q_\phi(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{a}' \sim \pi_\theta(\mathbf{a}'|\mathbf{s}')} [Q_\phi(\mathbf{s}', \mathbf{a}')]))^2 \right]$$

$$\mathcal{L}_A(\theta) = -E_{(\mathbf{s}, \mathbf{a}) \sim \pi_\beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \frac{1}{Z(\mathbf{s})} \exp \left(\frac{1}{\lambda} A^{\pi_{\text{old}}}(\mathbf{s}, \mathbf{a}) \right) \right]$$

- 1. $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_C(\phi)$
- 2. $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_A(\theta)$

- $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + E_{\mathbf{a}' \sim \pi_{\text{new}}} [Q(\mathbf{s}', \mathbf{a}')]$
- $\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \arg \max_{\pi} E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})] \text{ s.t. } D_{\text{KL}}(\pi \| \pi_\beta) \leq \epsilon$

Can we also avoid all OOD actions in the Q update?

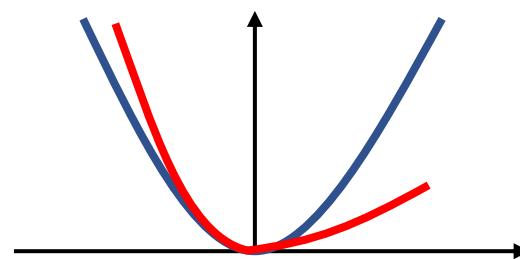
$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \underbrace{E_{\mathbf{a}' \sim \pi_{\text{new}}} [Q(\mathbf{s}', \mathbf{a}')]}_{V(\mathbf{s}') \leftarrow \text{just another neural network}}$$

$$V \leftarrow \arg \min_V \frac{1}{N} \sum_{i=1}^N \ell(V(\mathbf{s}_i), Q(\mathbf{s}_i, \mathbf{a}_i))$$

e.g., MSE loss $(V(\mathbf{s}_i) - Q(\mathbf{s}_i, \mathbf{a}_i))^2$

this action comes from π_β
not from π_{new}

expectile: $\ell_2^\tau(x) = \begin{cases} (1 - \tau)x^2 & \text{if } x > 0 \\ \tau x^2 & \text{else} \end{cases}$

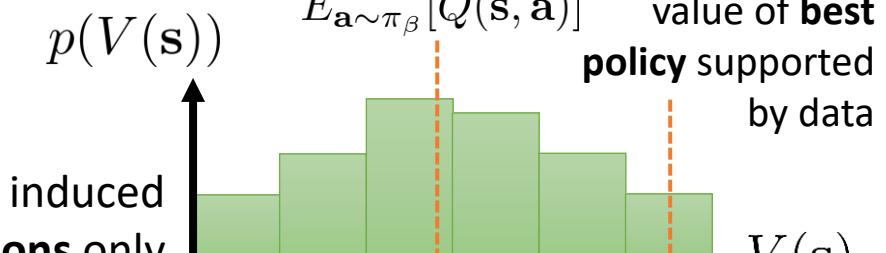


$$V(\mathbf{s}) \leftarrow \max_{\mathbf{a} \in \Omega(\mathbf{s})} Q(\mathbf{s}, \mathbf{a})$$

$$\Omega(\mathbf{s}) = \{\mathbf{a} : \pi_\beta(\mathbf{a}|\mathbf{s}) \geq \epsilon\}$$

if we use ℓ_2^τ for big τ

distribution is induced
by **actions only**

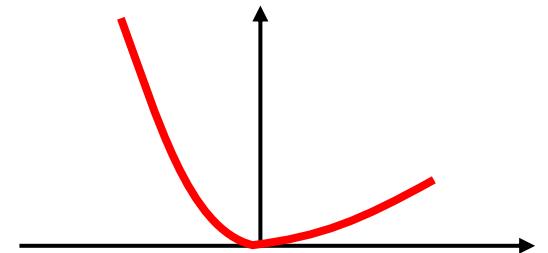


could **another** loss give us this?

Implicit Q-learning (IQL)

Q-learning with *implicit* policy improvement

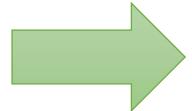
$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + V(\mathbf{s}') \quad V \leftarrow \arg \min_V \frac{1}{N} \sum_{i=1}^N \ell_2^\tau(V(\mathbf{s}_i), Q(\mathbf{s}_i, \mathbf{a}_i))$$



$$V(\mathbf{s}) \leftarrow \max_{\mathbf{a} \in \Omega(\mathbf{s})} Q(\mathbf{s}, \mathbf{a})$$

$$\Omega(\mathbf{s}) = \{\mathbf{a} : \pi_\beta(\mathbf{a}|\mathbf{s}) \geq \epsilon\}$$

if we use ℓ_2^τ for big τ



$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \max_{\mathbf{a}' \in \Omega(\mathbf{s}')} Q(\mathbf{s}', \mathbf{a}')$$

“implicit” policy

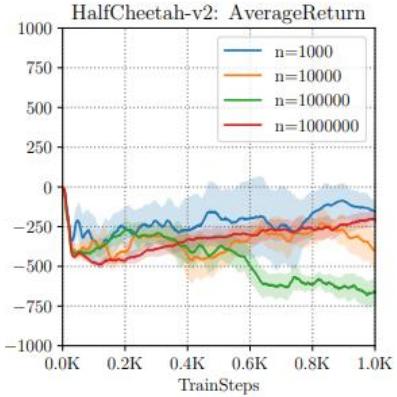
$$\pi_{\text{new}}(\mathbf{a}|\mathbf{s}) = \delta(\mathbf{a} = \arg \max_{\mathbf{a} \in \Omega(\mathbf{s})} Q(\mathbf{s}, \mathbf{a}))$$

Now we can do value function updates without ever risking out-of-distribution actions!

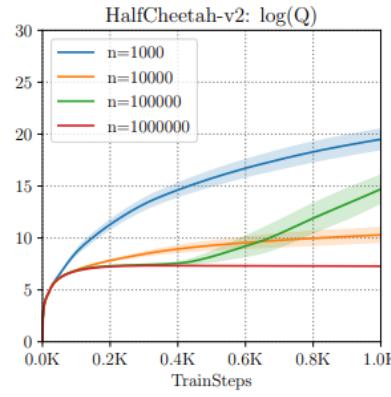
We'll see results soon, but first let's talk about **Option 2...**

Conservative Q-Learning

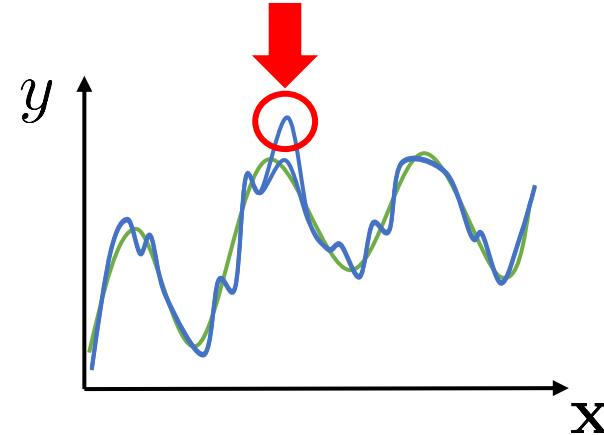
Conservative Q-learning (CQL)



how well it does



how well it *thinks*
it does (Q-values)



$$\hat{Q}^\pi = \arg \min_Q \max_\mu \alpha E_{\mathbf{s} \sim D, \mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})] \quad \text{term to push down big Q-values}$$

$$\text{regular objective } \left\{ +E_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim D} \left[(Q(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + E_\pi [Q(\mathbf{s}', \mathbf{a}')]))^2 \right] \right.$$

can show that $\hat{Q}^\pi \leq Q^\pi$ for large enough α

↑
true Q-function

Conservative Q-learning (CQL)

A *better* bound:

$$\hat{Q}^\pi = \arg \min_Q \max_\mu \alpha E_{\mathbf{s} \sim D, \mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})} [Q(\mathbf{s}, \mathbf{a})] - \alpha E_{(\mathbf{s}, \mathbf{a}) \sim D} [Q(\mathbf{s}, \mathbf{a})] + E_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim D} \left[(Q(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + E_\pi [Q(\mathbf{s}', \mathbf{a}')])))^2 \right]$$

always pushes Q-values down push up on (\mathbf{s}, \mathbf{a}) samples in data

} $\mathcal{L}_{\text{CQL}}(\hat{Q}^\pi)$

no longer guaranteed that $\hat{Q}^\pi(\mathbf{s}, \mathbf{a}) \leq Q^\pi(\mathbf{s}, \mathbf{a})$ for all (\mathbf{s}, \mathbf{a})

but guaranteed that $E_{\pi(\mathbf{a}|\mathbf{s})} [\hat{Q}^\pi(\mathbf{s}, \mathbf{a})] \leq E_{\pi(\mathbf{a}|\mathbf{s})} [Q^\pi(\mathbf{s}, \mathbf{a})]$ for all $\mathbf{s} \in D$

Conservative Q-learning (CQL)

- 
1. Update \hat{Q}^π w.r.t. $\mathcal{L}_{\text{CQL}}(\hat{Q}^\pi)$ using \mathcal{D}
 2. Update policy π

if actions are discrete:

$$\pi(\mathbf{a}|\mathbf{s}) = \begin{cases} 1 & \text{if } \mathbf{a} = \arg \max_{\mathbf{a}} \hat{Q}(\mathbf{s}, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$$

if actions are continuous:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \sum_i E_{\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)} [\hat{Q}(\mathbf{s}_i, \mathbf{a})]$$

Conservative Q-learning (CQL)

$$\hat{Q}^\pi = \arg \min_Q \max_\mu \underbrace{\alpha E_{\mathbf{s} \sim D, \mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s}, \mathbf{a})] - \alpha E_{(\mathbf{s}, \mathbf{a}) \sim D}[Q(\mathbf{s}, \mathbf{a})] - \mathcal{R}(\mu)}_{+ E_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim D} \left[(Q(\mathbf{s}, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + E_\pi[Q(\mathbf{s}', \mathbf{a}')]))^2 \right]} \quad \boxed{\mathcal{L}_{\text{CQL}}(\hat{Q}^\pi)}$$

regularization

common choice: $\mathcal{R} = E_{\mathbf{s} \sim D}[\mathcal{H}(\mu(\cdot|\mathbf{s}))]$ maximum entropy regularization

optimal choice: $\mu(\mathbf{a}|\mathbf{s}) \propto \exp(Q(\mathbf{s}, \mathbf{a}))$

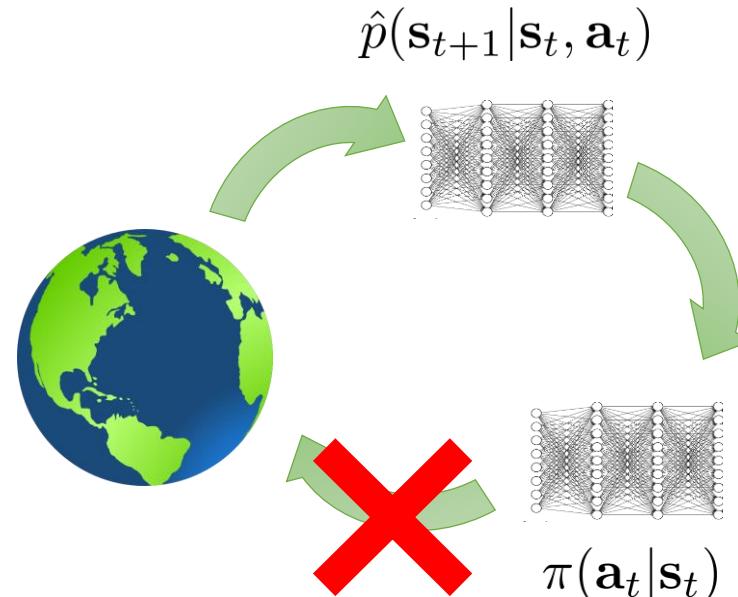
$$E_{\mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s}, \mathbf{a})] = \frac{\log \sum_{\mathbf{a}} \exp(Q(\mathbf{s}, \mathbf{a}))}{\text{_____}}$$

for discrete actions: just calculate directly

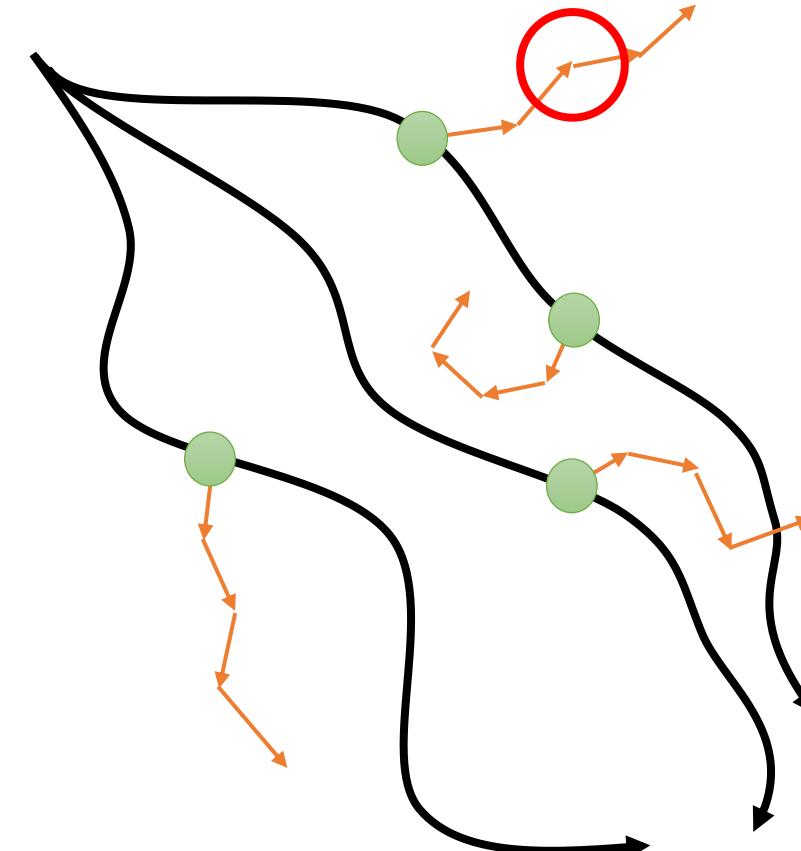
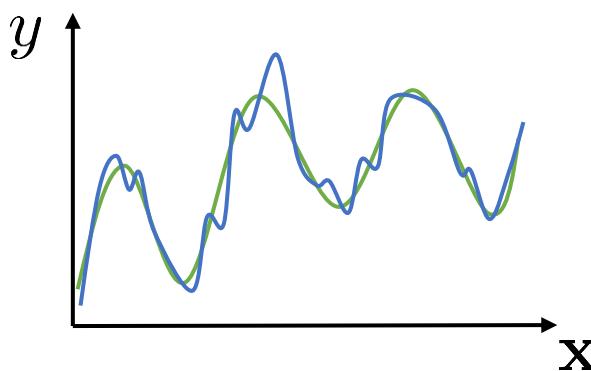
for continuous actions: use importance sampling to estimate $E_{\mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s}, \mathbf{a})]$

Model-Based Offline RL

How does model-based RL work?



what goes wrong when we can't collect more data?



the model answers "what if" questions

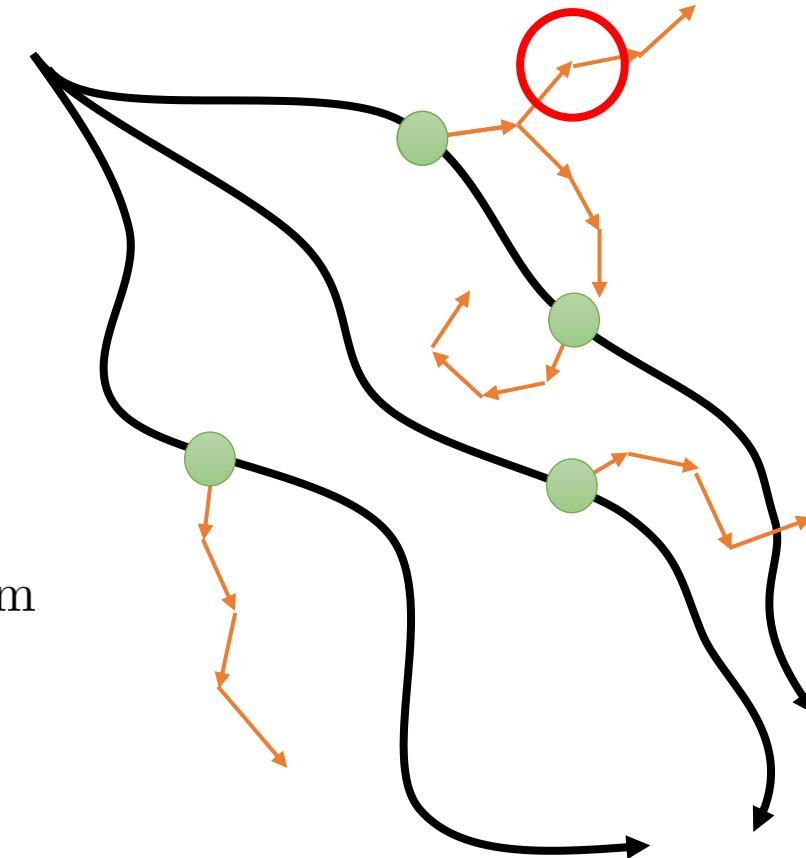
MOPO: Model-Based Offline Policy Optimization

solution: “punish” the policy for exploiting

$$\tilde{r}(s, a) = r(s, a) - \lambda u(s, a)$$

uncertainty penalty

...and then use any existing model-based RL algorithm



MOPO: Theoretical Analysis

$$\tilde{r}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) - \lambda u(\mathbf{s}, \mathbf{a})$$

we can represent the value function

model error is bounded (above) by $u(\mathbf{s}, \mathbf{a})$

Theorem 4.4. Under Assumption 4.2 and 4.3, the learned policy $\hat{\pi}$ in MOPO (Algorithm 1) satisfies

true return of policy trained under model $\longrightarrow \eta_M(\hat{\pi}) \geq \sup_{\pi} \{\eta_M(\pi) - 2\lambda\epsilon_u(\pi)\}$ (11)

In particular, for all $\delta \geq \delta_{\min}$,

$$\epsilon_u(\pi) := \bar{\mathbb{E}}_{(s,a) \sim \rho_T^\pi} [u(s, a)]$$

some implications:

$$\eta_M(\hat{\pi}) \geq \eta_M(\pi^B) - 2\lambda\epsilon_u(\pi^B)$$

➤ improves over behavior policy

$$\eta_M(\hat{\pi}) \geq \eta_M(\pi^*) - 2\lambda\epsilon_u(\pi^*)$$

➤ quantifies “optimality gap” in terms of model error

$$\eta_M(\hat{\pi}) \geq \eta_M(\pi^\delta) - 2\lambda\delta \quad (12)$$

$$\pi^\delta := \arg \max_{\pi: \epsilon_u(\pi) \leq \delta} \eta_M(\pi)$$

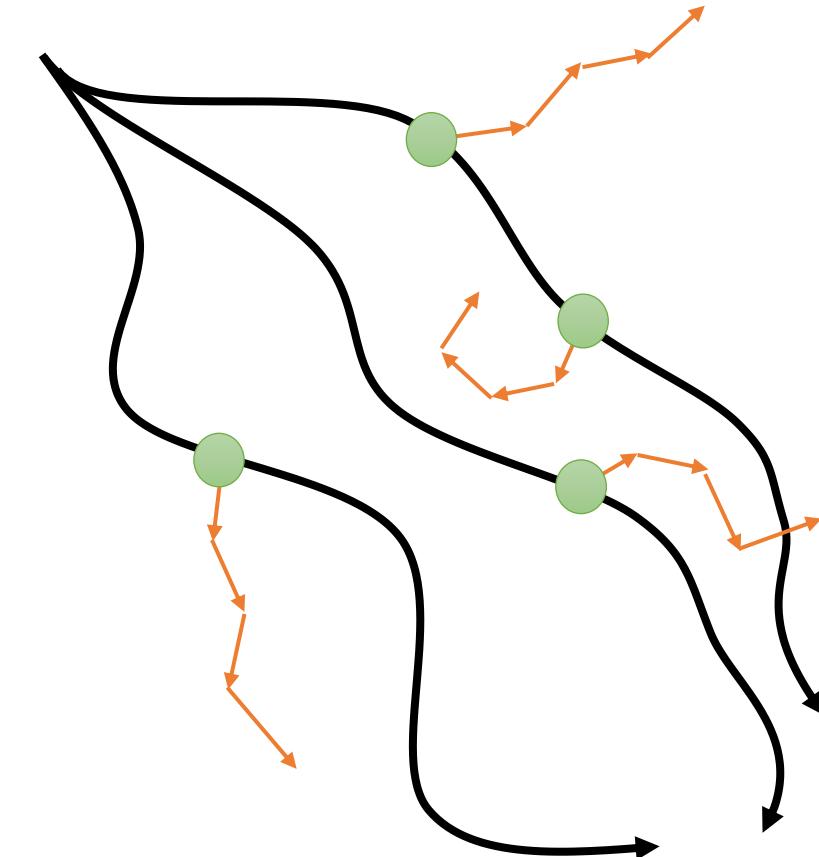
COMBO: Conservative Model-Based RL

Basic idea: just like CQL minimizes Q-value of policy actions, we can minimize Q-value of model state-action tuples

state-action tuples from the model

$$\begin{aligned}\hat{Q}^{k+1} \leftarrow \arg \min_Q & \beta (\mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \rho(\mathbf{s}, \mathbf{a})} [Q(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \mathcal{D}} [Q(\mathbf{s}, \mathbf{a})]) \\ & + \frac{1}{2} \mathbb{E}_{\mathbf{s}, \mathbf{a}, \mathbf{s}' \sim d_f} \left[(Q(\mathbf{s}, \mathbf{a}) - \hat{\mathcal{B}}^\pi \hat{Q}^k(\mathbf{s}, \mathbf{a}))^2 \right]. \quad (4)\end{aligned}$$

Intuition: if the model produces something that looks clearly different from real data, it's easy for the Q-function to make it look bad



Trajectory Transformer

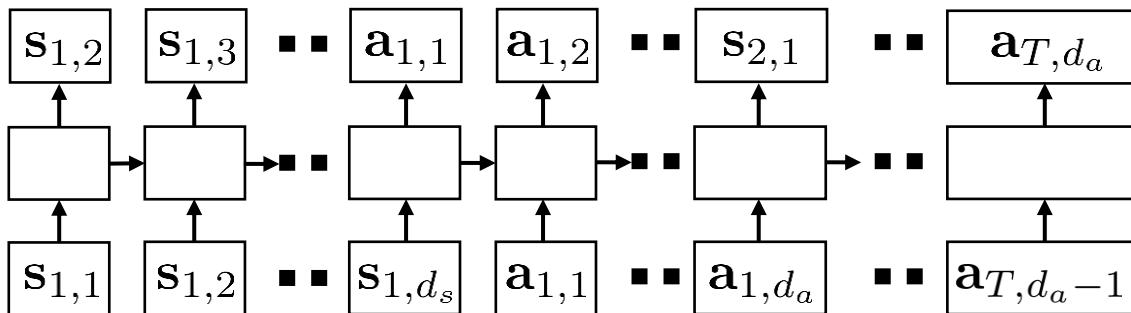
Basic ideas:

1. train a joint state-action model:

$$p_\beta(\tau) = p_\beta(s_1, a_2, \dots, s_T, a_T)$$

2. use a big expressive model (a Transformer)

The model:



Why does this work?

generating high-probability trajectories avoids out-of-distribution states & actions

using a really big model works well in offline mode (lots of compute, captures complex behavior policies)



Trajectory Transformer making accurate predictions to hundreds of steps

How to do control:

beam search, but use $\sum_t r(s_t, a_t)$ instead of probability

1. given current sequence, sample next tokens from model
2. store top K tokens with highest cumulative reward
3. move on to next token

Summary, Applications, Open Questions

Which offline RL algorithm do I use?

If you want to *only* train offline...

Conservative Q-learning	+ just one hyperparameter	+ well understood and widely tested
Implicit Q-learning	+ more flexible (offline + online)	- more hyperparameters

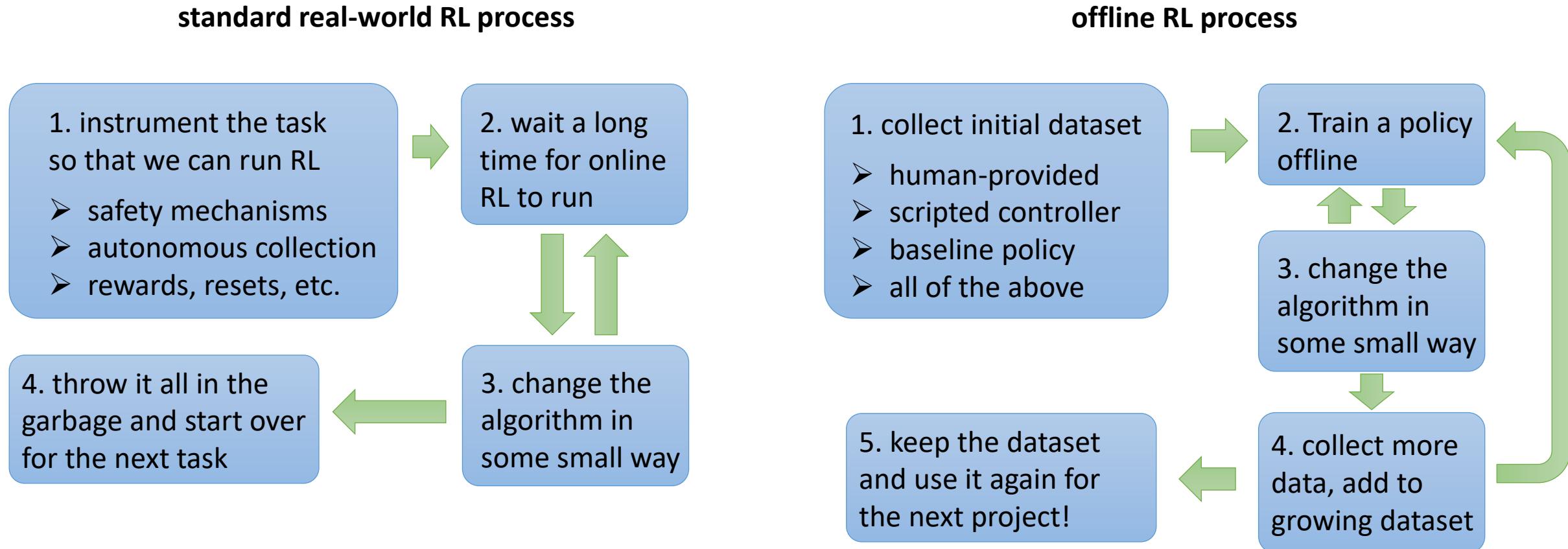
If you want to *only* train offline and finetune online

Advantage-weighted actor-critic (AWAC)	+ widely used and well tested
Implicit Q-learning	+ seems to perform much better!

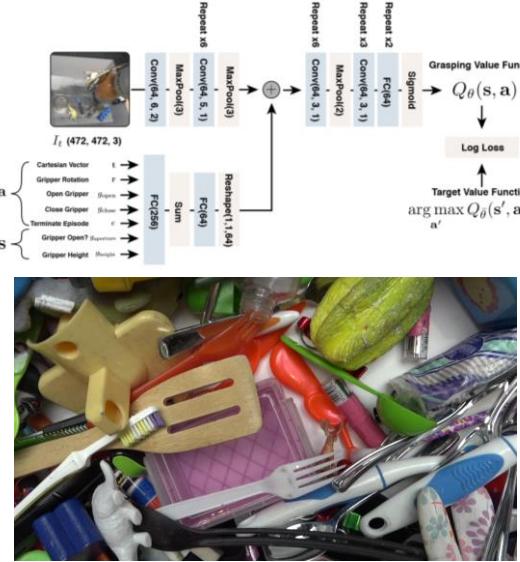
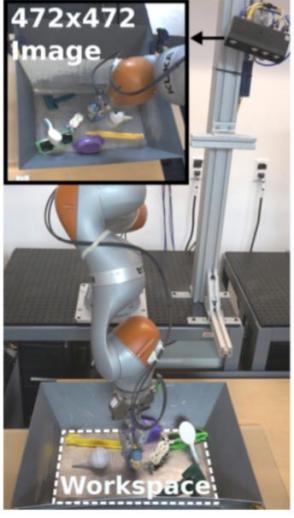
If you have a good way to train models in your domain

COMBO	+ similar properties as CQL, but benefits from models - not always easy to train a good model in your domain!
Trajectory transformer	+ very powerful and effective models - extremely computationally expensive to train and evaluate

The power of offline RL

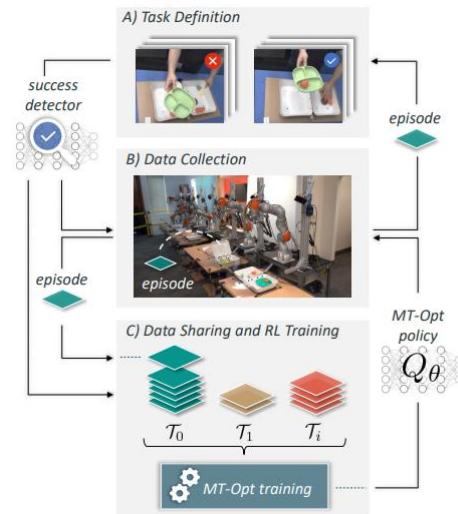
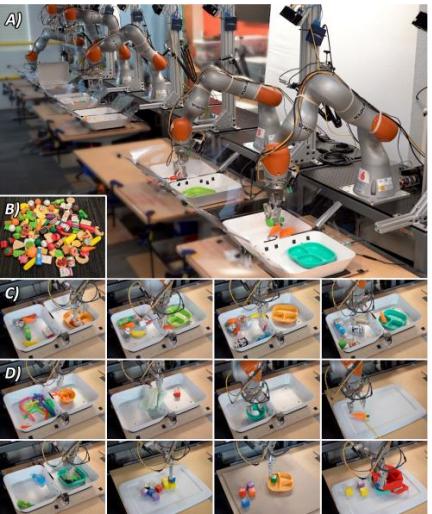


Offline RL in robotic manipulation: MT-Opt, AMs



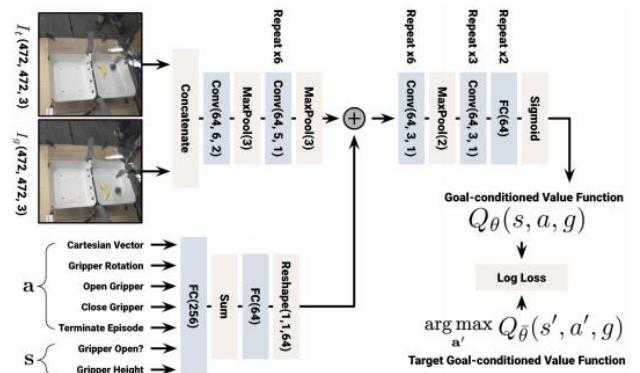
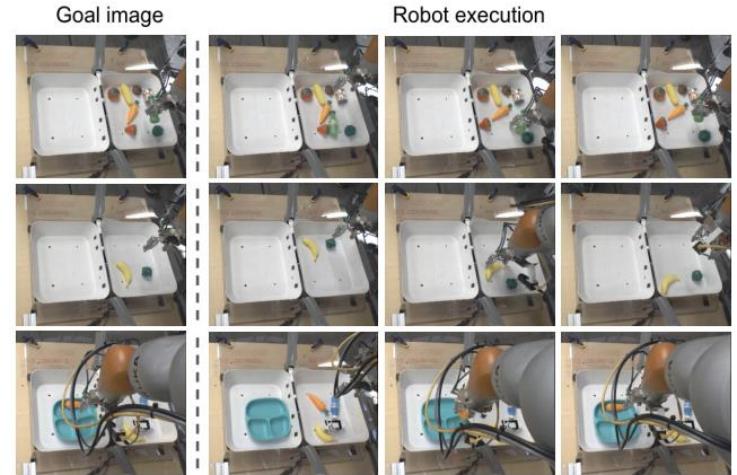
Kalashnikov, Irpan, Pastor, Ibarz, Herzong, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. **QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills**

- 12 different tasks
- Thousands of objects
- Months of data collection



Kalashnikov, Varley, Chebotar, Swanson, Jonschkowski, Finn, Levine, Hausman. **MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale.** 2021.

New hypothesis: could we learn these tasks **without** rewards using goal-conditioned RL?



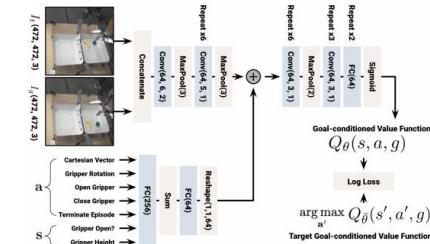
Actionable Models: Offline RL with Goals



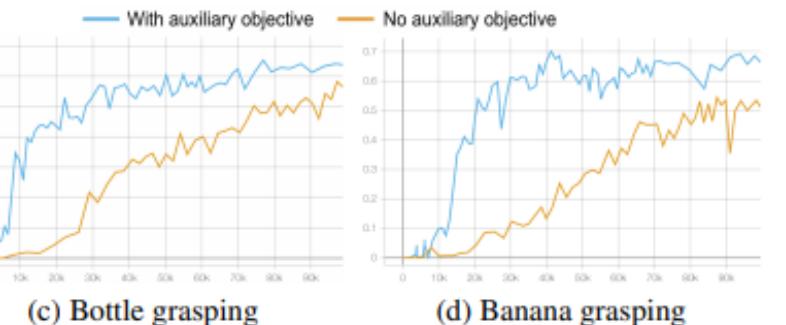
Semantic grasping: grasp bottle

- No reward function at all, task is defined entirely using a **goal image**!
- Uses a conservative offline RL method designed for goal-reaching, based on CQL
- Works very well as an **unsupervised pretraining objective**!

1. Train **goal-conditioned Q**-function with offline RL



2. Finetune with a **task reward** and limited data



(c) Bottle grasping

(d) Banana grasping

More examples

Early 2020: Greg Kahn collects 40 hours of robot navigation data



Kahn, Abbeel, Levine. **BADGR: An Autonomous Self-Supervised Learning-Based Navigation System**. 2020.

Late 2020: Dhruv Shah uses it to build goal-conditioned navigation system

Demo: Contactless Pizza Delivery



Shah, Eysenbach, Kahn, Rhinehart, Levine. **ViNG: Learning Open-World Navigation with Visual Goals**. 2020.

Early 2021: Dhruv Shah uses the **same** dataset to train an exploration system

Satellite view for visualization purposes only

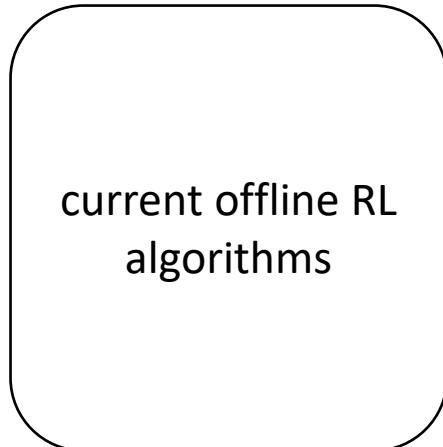
When deployed in a previously *unseen* environment, RECON explores the environment using a *latent goal* model in search of the target image.

Run 1: Exploration

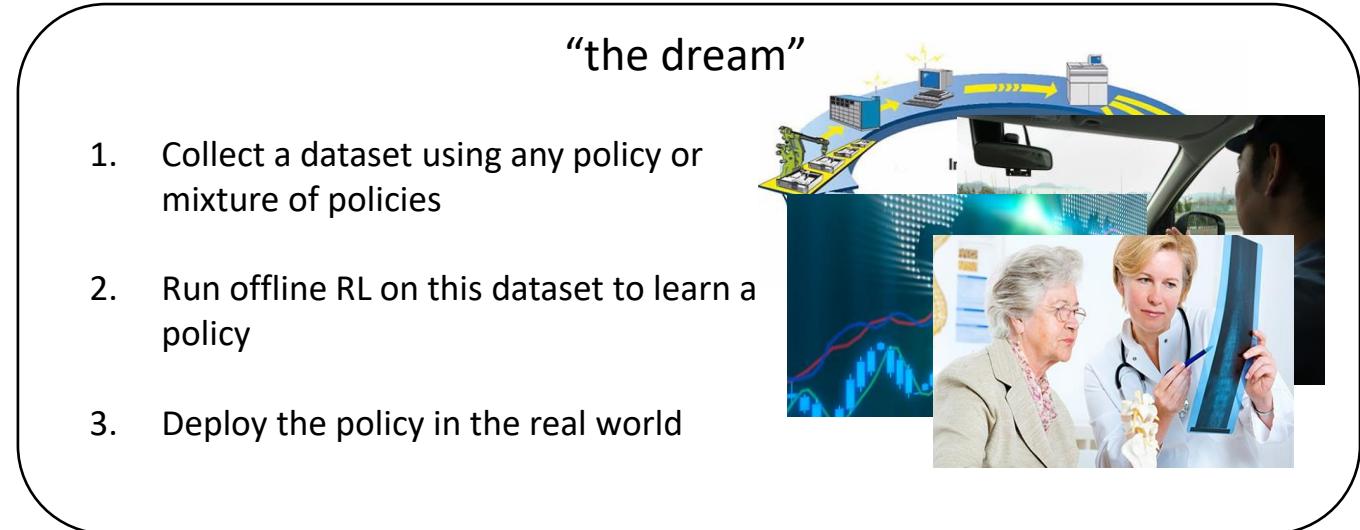


Shah, Eysenbach, Rhinehart, Levine. **RECON: Rapid Exploration for Open-World Navigation with Latent Goal Models**. 2020.

Takeaways, conclusions, future directions



“the gap”



- An offline RL **workflow**
 - Supervised learning workflow: train/test split
 - Offline RL workflow: ???
- Statistical **guarantees**
 - Biggest challenge: distributional shift/counterfactuals
 - Can we make any guarantees?
- Scalable methods, large-scale applications
 - Dialogue systems
 - Data-driven navigation and driving

A starting point: Kumar, Singh, Tian, Finn, Levine. **A Workflow for Offline Model-Free Robotic Reinforcement Learning.** CoRL 2021



Reinforcement Learning Theory Basics

CS 285

Instructor: Sergey Levine
UC Berkeley



What questions do we ask in RL theory?

Lots of different questions! But here are a few common ones:

If I use this algorithm with N samples, k iterations, how good is the result?

Let's say we're doing Q-learning...

$$\|\hat{Q}_k - Q^*\| \leq \epsilon \quad \text{with probability at least } 1 - \delta \quad \text{if } N \geq f(\epsilon, \delta)$$

$$\|Q^{\pi_k} - Q^*\| \leq \epsilon$$

We'll focus on these types of questions today

 not the same thing! Q^{π_k} is the *true* Q -function of policy at iteration k

If I use this exploration algorithm, how high is my regret?

$$\text{Reg}(T) \leq \mathcal{O}\left(\sqrt{T \cdot N \cdot \log \frac{NT}{\delta}}\right) + \delta T$$

But there are many others!

What kinds of assumptions do we make?

Effective analysis is **very** hard in RL without strong assumptions

The trick is to make assumptions that admit interesting conclusions without divorcing us (too much) from reality

Exploration: Performance of RL methods is greatly complicated by exploration – how likely are we to find (potentially sparse) rewards?

Theoretical guarantees typically address worst case performance, and worst case exploration is extremely hard

Goal: show that exploration method (e.g., counts) is $\text{Poly}(|S|, |A|, 1/(1-\gamma))$

Learning: If we somehow “abstract away” exploration, how many samples do we need to effectively learn a model or value function that results in good performance?

“generative model” assumption: assume we can sample from $P(s'|s, a)$ for *any* (s, a)

“oracle exploration”: for every (s, a) , sample $s' \sim P(s'|s, a)$ N times

What's the point?

1. Prove that our RL algorithms will work perfectly every time

Usually not possible with current deep RL methods, which are often not even guaranteed to converge

2. Understand how errors are affected by problem parameters

Do larger discounts work better than smaller ones?

If we want half the error, do we need 2x the samples? 4x? something else?

Usually we use precise theory to get imprecise **qualitative** conclusions about how various factors influence the performance of RL algorithms under **strong assumptions**, and try to make the assumptions reasonable enough that these conclusions are likely to apply to real problems (but they are not guaranteed to apply to real problems)

Don't take someone seriously if they say their RL algorithm has “provable guarantees” – the assumptions are always unrealistic, and theory is at best a rough guide to what might happen

Some basic sample complexity analysis

“oracle exploration”: for every (s, a) , sample $s' \sim P(s'|s, a)$ N times

simple “model based” algorithm:

$$1. \hat{P}(s'|s, a) = \frac{\#(s, a, s')}{N}$$

2. Given π , use \hat{P} to estimate \hat{Q}^π

how close is \hat{Q}^π to Q^π ? \longleftarrow

$$\|Q^\pi(s, a) - \hat{Q}^\pi(s, a)\|_\infty \leq \epsilon$$

with probability at least $1 - \delta$
if $N \geq f(\epsilon, \delta)$

$$\max_{s, a} |Q^\pi(s, a) - \hat{Q}^\pi(s, a)| \leq \epsilon$$

good to use $\|\cdot\|_\infty$ if we want *worst-case* performance

how close is \hat{Q}^* if we learn it using \hat{P} ? \longleftarrow

$$\|Q^*(s, a) - \hat{Q}^*(s, a)\|_\infty \leq \epsilon$$

optimal Q-function learned under \hat{P}

how good is the resulting policy? \longleftarrow

$$\|Q^*(s, a) - Q^{\hat{\pi}}(s, a)\|_\infty \leq \epsilon$$

the arg max policy corresponding to that Q-function

Concentration inequalities

whenever we need to answer questions about how close a learned function is to the true function, in terms of # of samples

Lemma A.1. (Hoeffding's inequality) Suppose X_1, X_2, \dots, X_n are a sequence of independent, identically distributed (i.i.d.) random variables with mean μ . Let $\bar{X}_n = n^{-1} \sum_{i=1}^n X_i$. Suppose that $X_i \in [b_-, b_+]$ with probability 1, then

$$P(\bar{X}_n \geq \mu + \epsilon) \leq e^{-2n\epsilon^2/(b_+ - b_-)^2}.$$

Similarly,

$$P(\bar{X}_n \leq \mu - \epsilon) \leq e^{-2n\epsilon^2/(b_+ - b_-)^2}.$$

interpretation:

if we estimate μ with n samples the probability we're off by more than ϵ is at most $2e^{-2n\epsilon^2/(b_+ - b_-)^2}$

equivalently, if we want this probability to be δ :

$$\delta \leq 2e^{-2n\epsilon^2/(b_+ - b_-)^2} \Rightarrow \log \frac{\delta}{2} \leq -2n\epsilon^2/(b_+ - b_-)^2 \Rightarrow \frac{(b_+ - b_-)^2}{2n} \log \frac{2}{\delta} \geq \epsilon^2 \Rightarrow \frac{b_+ - b_-}{\sqrt{2n}} \sqrt{\log \frac{2}{\delta}} \geq \epsilon$$

or... $n \leq \frac{(b_+ - b_-)^2}{2\epsilon^2} \log \frac{2}{\delta}$ error (ϵ) scales as $\frac{1}{\sqrt{n}}$

Concentration inequalities

$$\hat{P}(s'|s, a) = \frac{\#(s, a, s')}{N} \quad \text{discrete distribution}$$

Proposition A.8. (*Concentration for Discrete Distributions*) Let z be a discrete random variable that takes values in $\{1, \dots, d\}$, distributed according to q . We write q as a vector where $\vec{q} = [\Pr(z = j)]_{j=1}^d$. Assume we have N iid samples, and that our empirical estimate of \vec{q} is $[\hat{q}]_j = \sum_{i=1}^N \mathbf{1}[z_i = j]/N$.

We have that $\forall \epsilon > 0$:

$$\Pr \left(\|\hat{q} - \vec{q}\|_2 \geq 1/\sqrt{N} + \epsilon \right) \leq e^{-N\epsilon^2}.$$

which implies that:

$$\Pr \left(\|\hat{q} - \vec{q}\|_1 \geq \sqrt{d}(1/\sqrt{N} + \epsilon) \right) \leq e^{-N\epsilon^2}.$$

$$\delta \leq e^{-N\epsilon^2} \Rightarrow \epsilon \leq \frac{1}{\sqrt{N}} \sqrt{\log \frac{1}{\delta}} \quad \|\hat{P}(s'|s, a) - P(s'|s, a)\|_1 \leq \sqrt{|S|}(1/\sqrt{N} + \epsilon) \quad \text{with prob } 1 - \delta$$

$$\Rightarrow N \leq \frac{1}{\epsilon^2} \log \frac{1}{\delta} \quad \|\hat{P}(s'|s, a) - P(s'|s, a)\|_1 \leq \sqrt{\frac{|S|}{N}} + \sqrt{\frac{|S| \log 1/\delta}{N}} \leq c \sqrt{\frac{|S| \log 1/\delta}{N}}$$

A few useful lemmas

Next goal: relate error in \hat{P} to error in \hat{Q}^π

Relating P to Q^π :

$$Q^\pi(s, a) = r(s, a) + \gamma E_{s' \sim P(s'|s, a)}[V^\pi(s')]$$

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a)V^\pi(s')$$

$$Q^\pi = r + \gamma P^\pi Q^\pi$$

$$Q^\pi - \gamma P^\pi Q^\pi = r$$

$$(I - \gamma P^\pi)Q^\pi = r$$

$$Q^\pi = (I - \gamma P^\pi)^{-1}r$$

Q^π	r	P	V^π
$Q^\pi = r + \gamma PV^\pi$	 $ S A $	 $ S A $	 $ S A $
Π			 $ S $
$V^\pi = \Pi Q^\pi$	 $ S $	$Q^\pi = r + \gamma P^\pi Q^\pi$	
	$ S A $	$P^\pi = P\Pi$	

A few useful lemmas

$$Q^\pi = (I - \gamma P^\pi)^{-1} r$$

$$\hat{Q}^\pi = (I - \gamma \hat{P}^\pi)^{-1} r$$

true value

Simulation lemma:

$$Q^\pi - \hat{Q}^\pi = \gamma \underbrace{(I - \gamma \hat{P}^\pi)^{-1}}_{\text{evaluation operator}} \underbrace{(P - \hat{P}) V^\pi}_{\text{difference in probabilities}}$$

$$\begin{aligned} Q^\pi - \hat{Q}^\pi &= Q^\pi - (I - \gamma \hat{P}^\pi)^{-1} r \\ &= (I - \gamma \hat{P}^\pi)^{-1} (I - \gamma \hat{P}^\pi) Q^\pi - (I - \gamma \hat{P}^\pi)^{-1} r \\ &= (I - \gamma \hat{P}^\pi)^{-1} (I - \gamma \hat{P}^\pi) Q^\pi - (I - \gamma \hat{P}^\pi)^{-1} (I - \gamma P^\pi) Q^\pi \\ &= (I - \gamma \hat{P}^\pi)^{-1} ((\cancel{I} - \gamma \hat{P}^\pi) - (\cancel{I} - \gamma P^\pi)) Q^\pi \\ &= \gamma (I - \gamma \hat{P}^\pi)^{-1} (P^\pi - \hat{P}^\pi) Q^\pi \\ &= \gamma (I - \gamma \hat{P}^\pi)^{-1} (P\Pi - \hat{P}\Pi) Q^\pi \\ &= \gamma (I - \gamma \hat{P}^\pi)^{-1} (P - \hat{P}) \Pi Q^\pi \\ &= \gamma (I - \gamma \hat{P}^\pi)^{-1} (P - \hat{P}) V^\pi \end{aligned}$$

A few useful lemmas

Another useful lemma: given P^π and *any* vector $v \in \mathbb{R}^{|S||A|}$, we have:

$$\|(I - \gamma P^\pi)^{-1}v\|_\infty \leq \|v\|_\infty / (1 - \gamma)$$

“Q-function” corresponding to “reward” v is at most $1/(1 - \gamma)$ times larger

let $w = (I - \gamma P^\pi)^{-1}v$

$$\sum_{t=0}^{\infty} \gamma^t c = \frac{c}{1 - \gamma}$$

$$\|v\|_\infty = \|(I - \gamma P^\pi)w\|_\infty \geq \|w\|_\infty - \gamma \|P^\pi w\|_\infty \geq \|w\|_\infty - \gamma \|w\|_\infty = (1 - \gamma) \|w\|_\infty$$



triangle inequality



$$\|P^\pi\|_\infty \leq 1$$

$$\|a - b\| \geq \|a\| - \|b\|$$

$$\|v\|_\infty / (1 - \gamma) \geq \|w\|_\infty$$

Putting them together...

$$\|(I - \gamma P^\pi)^{-1}v\|_\infty \leq \|v\|_\infty / (1 - \gamma)$$

$$Q^\pi - \hat{Q}^\pi = \gamma(I - \gamma \hat{P}^\pi)^{-1}(P - \hat{P})V^\pi$$

can we bound $\|V^\pi\|_\infty$?

assume $R_{\max} = 1$

$$\sum_{t=0}^{\infty} \gamma^t r_t \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{1}{1-\gamma} R_{\max}$$

$$\|\hat{P}(s'|s, a) - P(s'|s, a)\|_1 \leq c \sqrt{\frac{|S| \log 1/\delta}{N}}$$

$$\|Q^\pi - \hat{Q}^\pi\|_\infty = \|\gamma(I - \gamma \hat{P}^\pi)^{-1}(P - \hat{P})V^\pi\|_\infty$$

$$\leq \frac{\gamma}{1-\gamma} \|(P - \hat{P})V^\pi\|_\infty$$

$$\leq \frac{\gamma}{1-\gamma} \left(\max_{s,a} \|P(\cdot|s, a) - \hat{P}(\cdot|s, a)\|_1 \right) \|V^\pi\|_\infty$$

$$\leq \frac{\gamma}{(1-\gamma)^2} \left(\max_{s,a} \|P(\cdot|s, a) - \hat{P}(\cdot|s, a)\|_1 \right)$$

$$\leq \frac{\gamma}{(1-\gamma)^2} c_2 \sqrt{\frac{|S| \log 1/\delta}{N}}$$

↑
technically need to use the union
bound here to account for probabilities

What does this mean?

$$\|Q^\pi - \hat{Q}^\pi\|_\infty \leq \frac{\gamma}{(1-\gamma)^2} c_2 \sqrt{\frac{|S| \log 1/\delta}{N}}$$

error grows **quadratically** in the horizon
each backup “accumulates” error

more samples = lower error

Some simple implications...

$$||Q^\pi - \hat{Q}^\pi||_\infty \leq \epsilon$$

$$\epsilon = \frac{\gamma}{(1-\gamma)^2} c_2 \sqrt{\frac{|S| \log 1/\delta}{N}}$$

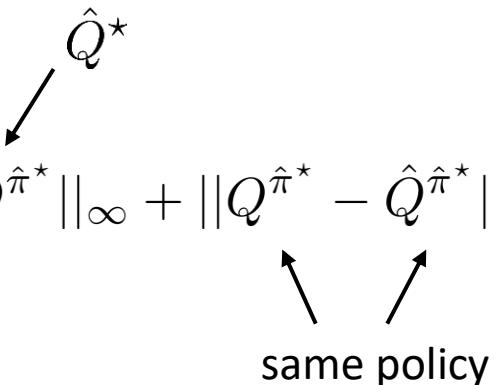
what about $||Q^* - \hat{Q}^*||_\infty$?

$$|\sup_x f(x) - \sup_x g(x)| \leq \sup_x |f(x) - g(x)|$$

$$||Q^* - \hat{Q}^*||_\infty = ||\sup_\pi Q^\pi - \sup_\pi \hat{Q}^\pi||_\infty \leq \sup_\pi ||Q^\pi - \hat{Q}^\pi||_\infty \leq \epsilon$$

what about $||Q^* - Q^{\hat{\pi}^*}||_\infty$?

$$||Q^* - Q^{\hat{\pi}^*}||_\infty = ||Q^* - \hat{Q}^{\hat{\pi}^*} + \hat{Q}^{\hat{\pi}^*} - Q^{\hat{\pi}^*}||_\infty \leq ||Q^* - \hat{Q}^{\hat{\pi}^*}||_\infty + ||Q^{\hat{\pi}^*} - \hat{Q}^{\hat{\pi}^*}||_\infty \leq 2\epsilon$$



What About Model-Free RL?

Analyzing fitted Q-iteration

abstract model of exact Q-iteration:

$$\begin{array}{c} \text{Bellman operator} \\ TQ = r + \gamma P \max_a Q \\ \downarrow \\ Q_{k+1} \leftarrow TQ_k \end{array}$$

abstract model of approximate fitted Q-iteration: $\hat{Q}_{k+1} \leftarrow \arg \min_{\hat{Q}} \|\hat{Q} - \hat{T}\hat{Q}_k\|$

which norm?
no convergence if $\|\cdot\|_2$
we'll assume $\|\cdot\|_\infty$

Question: as $k \rightarrow \infty$, $\hat{Q}_k \rightarrow ?$

$$\lim_{k \rightarrow \infty} \|\hat{Q}_k - Q^*\|_\infty \leq ?$$

where do errors come from?

$T \neq \hat{T}$ “sampling error”

$\hat{Q}_{k+1} \neq \hat{T}\hat{Q}_k$ “approximation error”

$$\hat{r}(s, a) = \frac{1}{N(s, a)} \sum_i \delta((s_i, a_i) = (s, a)) r_i \quad \hat{P}(s'|s, a) = \frac{N(s, a, s')}{N(s, a)}$$

approximate Bellman operator

$$\hat{T}Q = \hat{r} + \gamma \hat{P} \max_a Q$$

Note: these are **not** models, this is the effect of averaging together transitions in the data!

Let's analyze sampling error

$$\hat{Q}_{k+1} \leftarrow \arg \min_{\hat{Q}} \|\hat{Q} - \hat{T}\hat{Q}_k\| \quad T \neq \hat{T}$$

vs. $T\hat{Q}_k$

$$\begin{aligned} |\hat{T}Q(s, a) - TQ(s, a)| &= |\hat{r}(s, a) - r(s, a) + \gamma(E_{\hat{P}(s'|s, a)}[\max_{a'} Q(s', a')] - E_{P(s'|s, a)}[\max_{a'} Q(s', a')])| \\ &\leq \underbrace{|\hat{r}(s, a) - r(s, a)|}_{\text{estimation error of continuous random variable}} + \underbrace{\gamma |(E_{\hat{P}(s'|s, a)}[\max_{a'} Q(s', a')] - E_{P(s'|s, a)}[\max_{a'} Q(s', a')])|}_{\text{just use Hoeffding's inequality directly!}} \end{aligned}$$

$$|\hat{r}(s, a) - r(s, a)| \leq 2R_{\max} \sqrt{\frac{\log 1/\delta}{2N}}$$

$$\begin{aligned} &\sum_{s'} (\hat{P}(s'|s, a) - P(s'|s, a)) \max_{a'} Q(s', a') \\ &\leq \sum_{s'} |\hat{P}(s'|s, a) - P(s'|s, a)| \max_{s', a'} Q(s', a') \\ &= \|\hat{P}(\cdot|s, a) - P(\cdot|s, a)\|_1 \|Q\|_\infty \\ &\leq c \|Q\|_\infty \sqrt{\frac{\log 1/\delta}{N}} \end{aligned}$$

Let's analyze sampling error

$$\hat{Q}_{k+1} \leftarrow \arg \min_{\hat{Q}} \|\hat{Q} - \hat{T}\hat{Q}_k\| \quad T \neq \hat{T}$$

vs. $T\hat{Q}_k$

$$|\hat{T}Q(s, a) - TQ(s, a)| \leq 2R_{\max} \sqrt{\frac{\log 1/\delta}{2N}} + c\|Q\|_\infty \sqrt{\frac{\log 1/\delta}{N}}$$

$$\|\hat{T}Q - TQ\|_\infty \leq 2R_{\max}c_1 \sqrt{\frac{\log |S||A|/\delta}{2N}} + c_2\|Q\|_\infty \sqrt{\frac{\log |S|/\delta}{N}}$$

using union bound

Let's analyze approximation error

approximation error assumption: $\|\hat{Q}_{k+1} - T\hat{Q}_k\|_\infty \leq \epsilon_k$

This is a strong assumption!

we'll analyze the exact backup operator for now,
but we'll come back to approximate backups later!

$$\begin{aligned}\|\hat{Q}_k - Q^*\|_\infty &= \|\hat{Q}_k - T\hat{Q}_{k-1} + T\hat{Q}_{k-1} - Q^*\|_\infty && \text{using fact that } Q^* \text{ is fixed point of } T \\ &= \|(\hat{Q}_k - T\hat{Q}_{k-1}) + (T\hat{Q}_{k-1} - TQ^*)\|_\infty \\ &\leq \|\hat{Q}_k - T\hat{Q}_{k-1}\|_\infty + \|T\hat{Q}_{k-1} - TQ^*\|_\infty \\ &\leq \epsilon_{k-1} + \|T\hat{Q}_{k-1} - TQ^*\|_\infty && \text{using fact that } T \text{ is a } \gamma\text{-contraction} \\ &\leq \epsilon_{k-1} + \gamma \|\hat{Q}_{k-1} - Q^*\|_\infty\end{aligned}$$

Let's analyze approximation error

$$\begin{aligned} \|\hat{Q}_k - Q^*\|_\infty &\leq \epsilon_{k-1} + \gamma \|\hat{Q}_{k-1} - Q^*\|_\infty \\ &\leq \epsilon_{k-1} + \gamma \epsilon_{k-2} + \gamma^2 \|\hat{Q}_{k-2} - Q^*\|_\infty \\ &\leq \epsilon_{k-1} + \gamma \epsilon_{k-2} + \gamma^2 \epsilon_{k-3} + \gamma^3 \|\hat{Q}_{k-2} - Q^*\|_\infty \\ &\leq \sum_{i=0}^{k-1} \gamma^i \epsilon_{k-i-1} + \gamma^k \|\hat{Q}_0 - Q^*\|_\infty \end{aligned}$$

$$\lim_{k \rightarrow \infty} \|\hat{Q}_k - Q^*\|_\infty \leq \sum_{i=0}^{\infty} \gamma^i \max_k \epsilon_k = \frac{1}{1-\gamma} \|\epsilon\|_\infty$$

approximation error scales with “horizon”

Putting it together

$$\|\hat{T}Q - TQ\|_\infty \leq 2R_{\max}c_1 \sqrt{\frac{\log |S||A|/\delta}{2N}} + c_2\|Q\|_\infty \sqrt{\frac{\log |S|/\delta}{N}}$$

“sampling error”

$$\lim_{k \rightarrow \infty} \|\hat{Q}_k - Q^*\|_\infty \leq \frac{1}{1-\gamma} \max_k \epsilon_k = \frac{1}{1-\gamma} \max_k \|\hat{Q}_k - T\hat{Q}_{k-1}\|_\infty$$

“approximation error”

how much \hat{Q}_{k+1} differs from $T\hat{Q}_k$

due to: sampling error ($T \neq \hat{T}$)

approximation error ($\hat{Q}_k \neq \hat{T}\hat{Q}_{k-1}$)

$$\begin{aligned} \|\hat{Q}_k - T\hat{Q}_{k-1}\|_\infty &= \|\hat{Q}_k - \hat{T}\hat{Q}_{k-1} + \hat{T}\hat{Q}_{k-1} - T\hat{Q}_{k-1}\|_\infty \\ &\leq \underbrace{\|\hat{Q}_k - \hat{T}\hat{Q}_{k-1}\|_\infty}_{\text{approximation}} + \underbrace{\|\hat{T}\hat{Q}_{k-1} - T\hat{Q}_{k-1}\|_\infty}_{\text{sampling}} \end{aligned}$$

What does it all mean?

$$||\hat{T}Q - TQ||_\infty \leq 2R_{\max}c_1 \sqrt{\frac{\log |S||A|/\delta}{2N}} + c_2||Q||_\infty \sqrt{\frac{\log |S|/\delta}{N}}$$

“sampling error”

$$\lim_{k \rightarrow \infty} \|\hat{Q}_k - Q^*\|_\infty \leq \frac{1}{1-\gamma} \max_k \epsilon_k = \frac{1}{1-\gamma} \max_k \|\hat{Q}_k - T\hat{Q}_{k-1}\|_\infty$$

“approximation error”

$$||\hat{Q}_k - T\hat{Q}_{k-1}||_\infty \leq ||\hat{Q}_k - \hat{T}\hat{Q}_{k-1}||_\infty + ||\hat{T}\hat{Q}_{k-1} - T\hat{Q}_{k-1}||_\infty$$

error “compounds” with horizon, over iterations and due to sampling

so far we needed strong (infinity norm) assumptions

more advanced results can be derived with p-norms under some distribution:

$$\|\hat{Q}_k - Q^*\|_{p,\mu} = \left(E_{(s,a) \sim \mu(s,a)} [|\hat{Q}_k(s, a) - Q^*(s, a)|^p] \right)^{1/p}$$

Variational Inference and Generative Models

CS 285

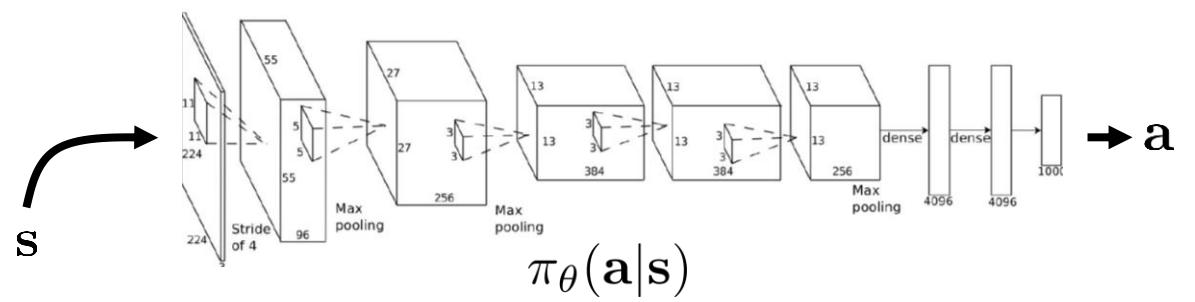
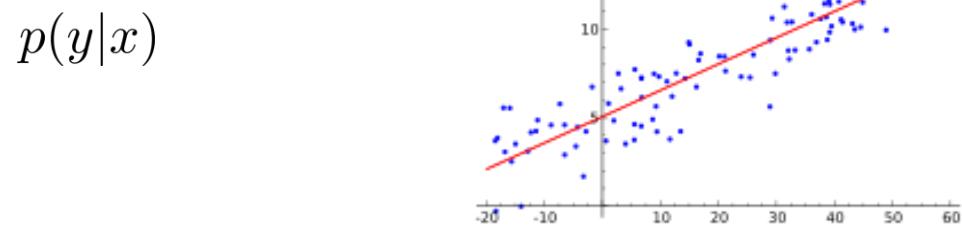
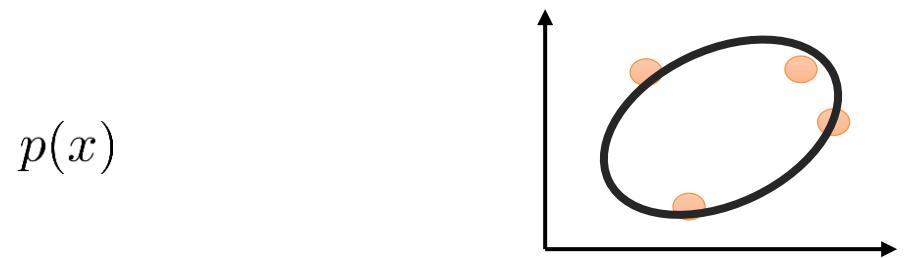
Instructor: Sergey Levine
UC Berkeley



Today's Lecture

1. Probabilistic latent variable models
 2. Variational inference
 3. Amortized variational inference
 4. Generative models: variational autoencoders
- Goals
 - Understand latent variable models in deep learning
 - Understand how to use (amortized) variational inference

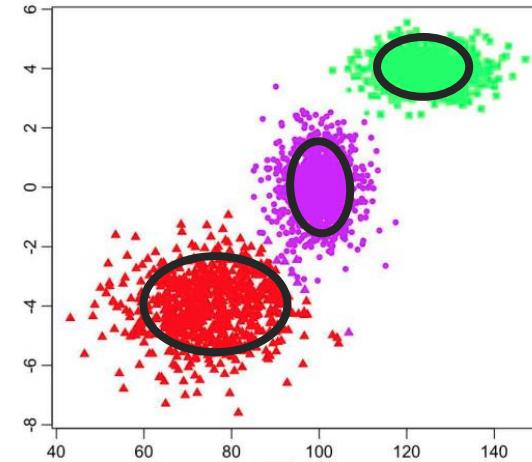
Probabilistic models



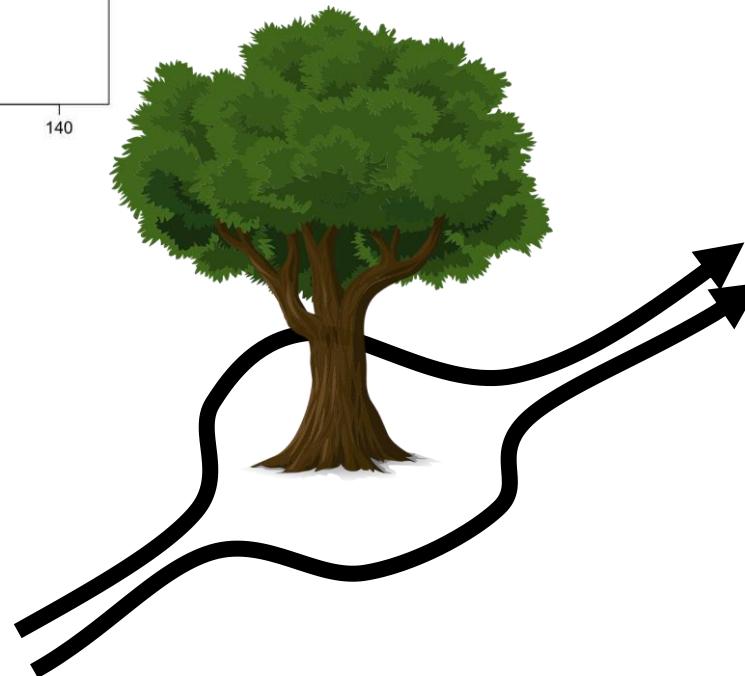
Latent variable models

$$p(x) = \sum_z p(x|z)p(z)$$

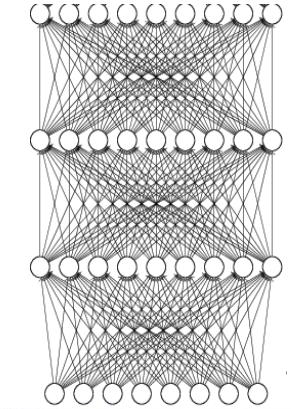
↑
mixture
element



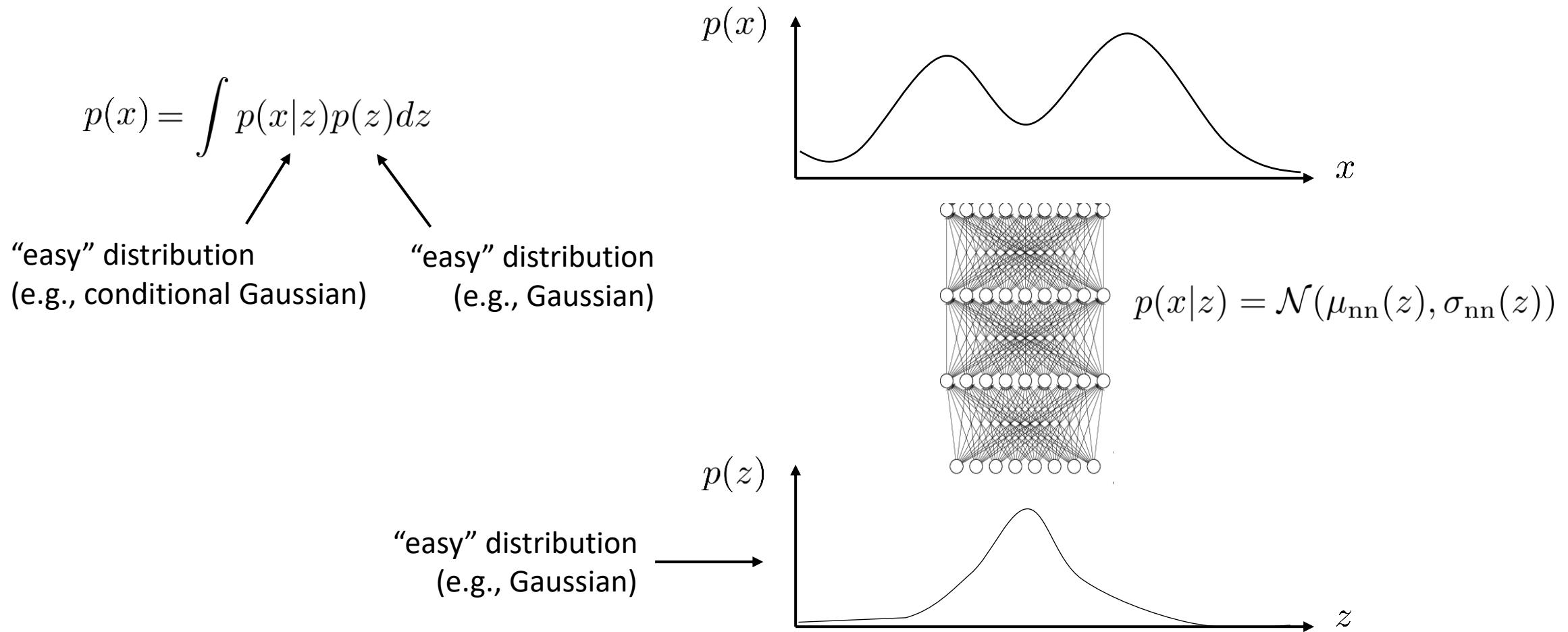
$$p(y|x) = \sum_z p(y|x, z)p(z)$$



Three blue bell-shaped curves representing probability density functions, labeled $w_1, \mu_1, \Sigma_1, \dots, w_N, \mu_N, \sigma_N$.

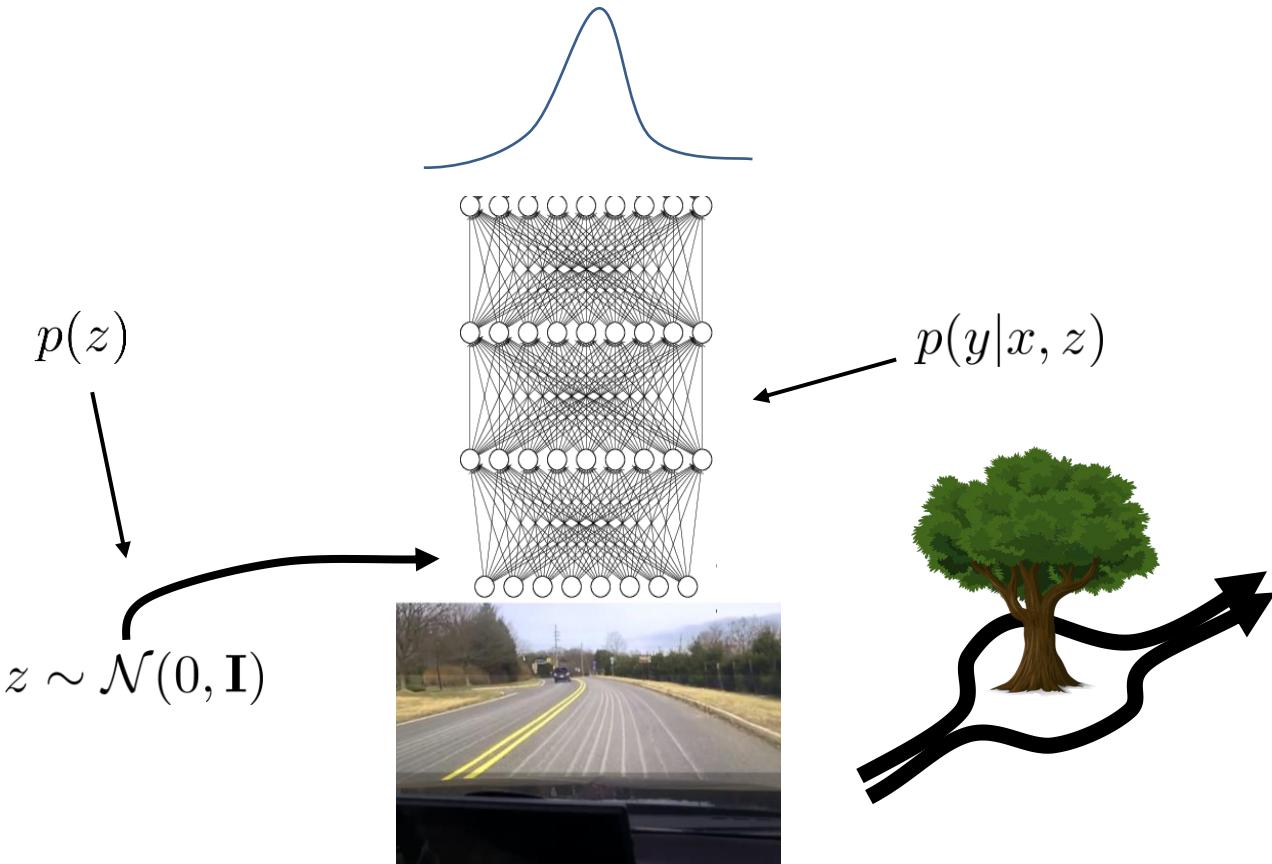


Latent variable models in general

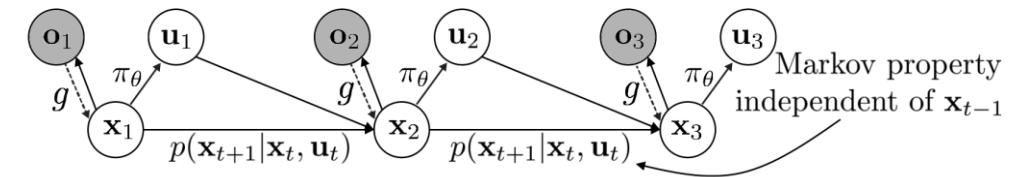


Latent variable models in RL

conditional latent variable
models for multi-modal policies



latent variable models for
model-based RL

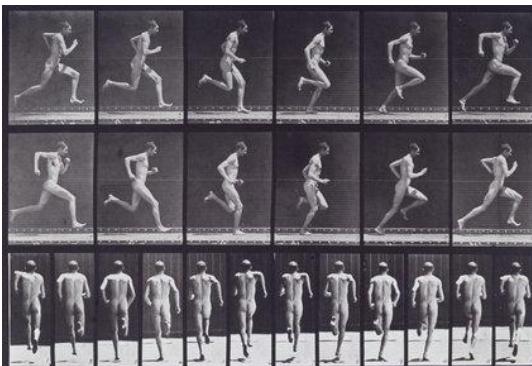


$p(o_t|x_t)$ actually models $p(x_{t+1}|x_t)$ and $p(x_1)$

$p(x_t)$ latent space has *structure*

Other places we'll see latent variable models

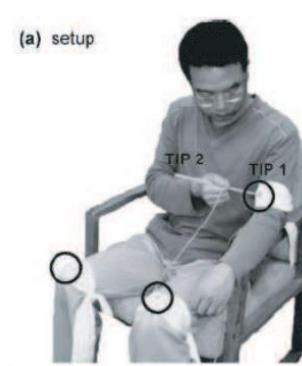
Using RL/control + variational inference to model human behavior



Muybridge (c. 1870)



Mombaur et al. '09



Li & Todorov '06



Ziebart '08

Using generative models and variational inference for exploration



How do we train latent variable models?

the model: $p_\theta(x)$

the data: $\mathcal{D} = \{x_1, x_2, x_3, \dots, x_N\}$

maximum likelihood fit:

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_i \log p_\theta(x_i)$$

$$p(x) = \int p(x|z)p(z)dz$$

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_i \log \left(\int p_\theta(x_i|z)p(z)dz \right)$$



completely intractable

Estimating the log-likelihood

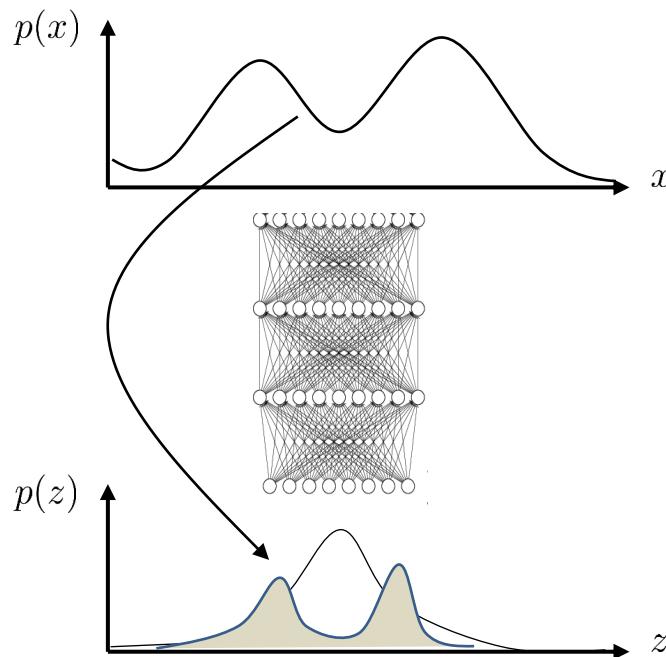
alternative: *expected* log-likelihood:

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_i E_{z \sim p(z|x_i)} [\log p_{\theta}(x_i, z)]$$

but... how do we calculate $p(z|x_i)$?

intuition: “guess” most likely z given x_i ,
and pretend it’s the right one

...but there are many possible values of z
so use the distribution $p(z|x_i)$



Variational Inference

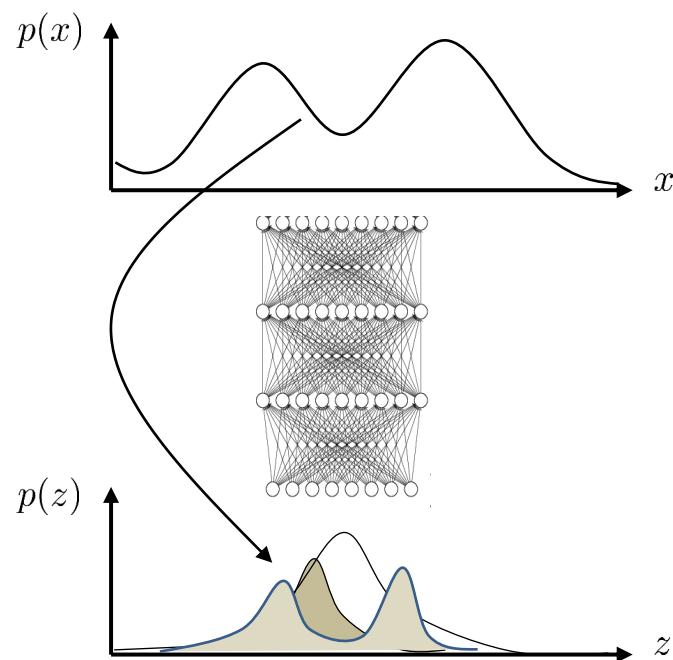
The variational approximation

but... how do we calculate $p(z|x_i)$?

can bound $\log p(x_i)!$

$$\begin{aligned}\log p(x_i) &= \log \int_z p(x_i|z)p(z) \\ &= \log \int_z p(x_i|z)p(z) \frac{q_i(z)}{q_i(z)} \\ &= \log E_{z \sim q_i(z)} \left[\frac{p(x_i|z)p(z)}{q_i(z)} \right]\end{aligned}$$

what if we approximate with $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$



The variational approximation

but... how do we calculate $p(z|x_i)$?

can bound $\log p(x_i)$!

$$\log p(x_i) = \log \int_z p(x_i|z)p(z)$$

$$= \log \int_z p(x_i|z)p(z) \frac{q_i(z)}{q_i(z)}$$

$$= \log E_{z \sim q_i(z)} \left[\frac{p(x_i|z)p(z)}{q_i(z)} \right]$$

$$\geq E_{z \sim q_i(z)} \left[\log \frac{p(x_i|z)p(z)}{q_i(z)} \right] = E_{z \sim q_i(z)} [\log p(x_i|z) + \log p(z)] + H_{\mathcal{A}q_i}(z) [\log q_i(z)]$$

Jensen's inequality

$$\log E[y] \geq E[\log y]$$

maximizing this maximizes $\log p(x_i)$



A brief aside...

Entropy:

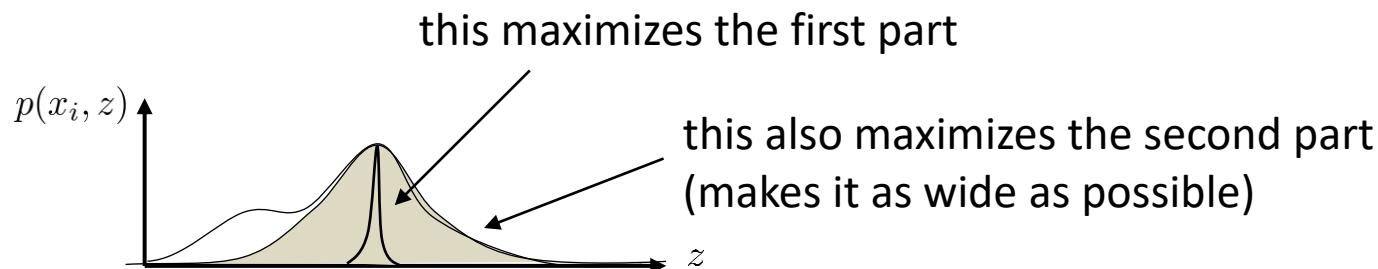
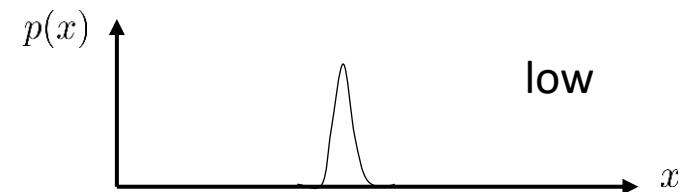
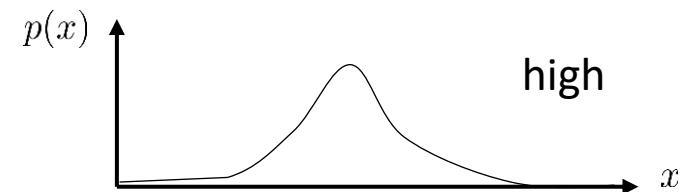
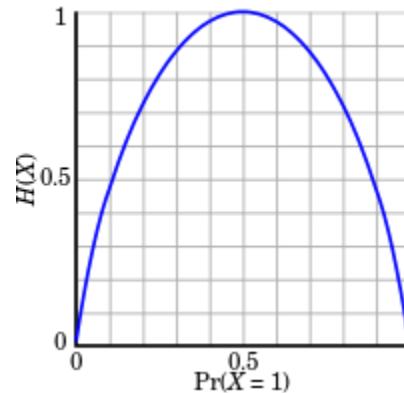
$$\mathcal{H}(p) = -E_{x \sim p(x)}[\log p(x)] = - \int_x p(x) \log p(x) dx$$

Intuition 1: how *random* is the random variable?

Intuition 2: how large is the log probability in expectation *under itself*

what do we expect this to do?

$$E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)$$



A brief aside...

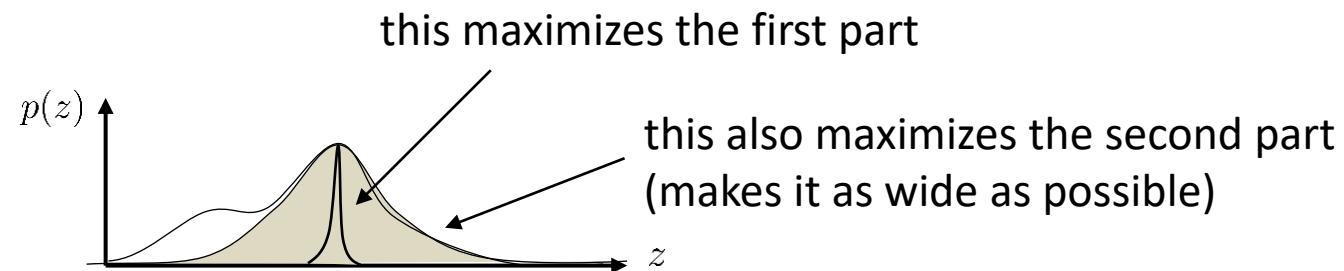
KL-Divergence:

$$D_{\text{KL}}(q \| p) = E_{x \sim q(x)} \left[\log \frac{q(x)}{p(x)} \right] = E_{x \sim q(x)} [\log q(x)] - E_{x \sim q(x)} [\log p(x)] = -E_{x \sim q(x)} [\log p(x)] - \mathcal{H}(q)$$

Intuition 1: how *different* are two distributions?

Intuition 2: how small is the expected log probability of one distribution under another, minus entropy?

why entropy?



The variational approximation

$$\log p(x_i) \geq \overbrace{E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}^{\mathcal{L}_i(p, q_i)}$$

what makes a good $q_i(z)$?

intuition: $q_i(z)$ should approximate $p(z|x_i)$

approximate in what sense?

compare in terms of KL-divergence: $D_{\text{KL}}(q_i(z)\|p(z|x))$

why?

$$\begin{aligned} D_{\text{KL}}(q_i(x_i)\|p(z|x_i)) &= E_{z \sim q_i(z)} \left[\log \frac{q_i(z)}{p(z|x_i)} \right] = E_{z \sim q_i(z)} \left[\log \frac{q_i(z)p(x_i)}{p(x_i, z)} \right] \\ &= -E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] + E_{z \sim q_i(z)}[\log q_i(z)] + E_{z \sim q_i(z)}[\log p(x_i)] \\ &= -E_{z \sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i) + \log p(x_i) \\ &= -\mathcal{L}_i(p, q_i) + \log p(x_i) \end{aligned}$$

$$\log p(x_i) = D_{\text{KL}}(q_i(z)\|p(z|x_i)) + \mathcal{L}_i(p, q_i)$$

$$\log p(x_i) \geq \mathcal{L}_i(p, q_i)$$

The variational approximation

$$\log p(x_i) \geq E_{z \sim q_i(z)} [\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)$$

$$\log p(x_i) = D_{\text{KL}}(q_i(z) \| p(z|x_i)) + \mathcal{L}_i(p, q_i)$$

$$\log p(x_i) \geq \mathcal{L}_i(p, q_i)$$

$$\begin{aligned} D_{\text{KL}}(q_i(z) \| p(z|x_i)) &= E_{z \sim q_i(z)} \left[\log \frac{q_i(z)}{p(z|x_i)} \right] = E_{z \sim q_i(z)} \left[\log \frac{q_i(z)p(x_i)}{p(x_i, z)} \right] \\ &= -E_{z \sim q_i(z)} [\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i) + \log p(x_i) \end{aligned}$$

—————
 $-\mathcal{L}_i(p, q_i)$ independent of q_i !

\Rightarrow maximizing $\mathcal{L}_i(p, q_i)$ w.r.t. q_i minimizes KL-divergence!

How do we use this?

$$\log p(x_i) \geq E_{z \sim q_i(z)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_i)$$

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_i \log p_\theta(x_i)$$

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_i \mathcal{L}_i(p, q_i)$$

for each x_i (or mini-batch):

calculate $\nabla_{\theta} \mathcal{L}_i(p, q_i)$:

sample $z \sim q_i(z)$

$\nabla_{\theta} \mathcal{L}_i(p, q_i) \approx \nabla_{\theta} \log p_\theta(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L}_i(p, q_i)$

update q_i to maximize $\mathcal{L}_i(p, q_i)$

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

gradient ascent on μ_i, σ_i

how?

What's the problem?

for each x_i (or mini-batch):

calculate $\nabla_{\theta} \mathcal{L}_i(p, q_i)$:

sample $z \sim q_i(z)$

$$\nabla_{\theta} \mathcal{L}_i(p, q_i) \approx \nabla_{\theta} \log p_{\theta}(x_i|z)$$

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L}_i(p, q_i)$$

update q_i to maximize $\mathcal{L}_i(p, q_i)$

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

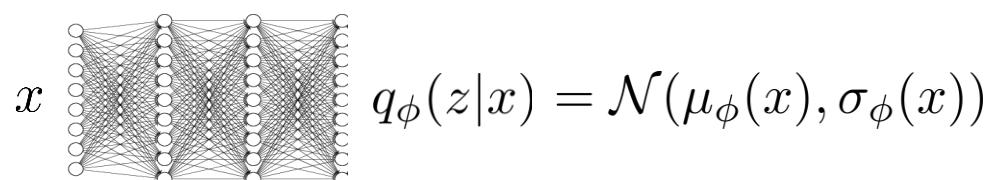
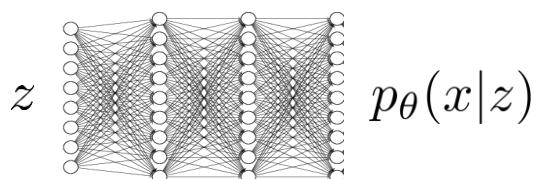
gradient ascent on μ_i, σ_i

How many parameters are there?

$$|\theta| + (|\mu_i| + |\sigma_i|) \times N$$

intuition: $q_i(z)$ should approximate $p(z|x_i)$

what if we learn a *network* $q_i(z) = q(z|x_i) \approx p(z|x_i)$?



Amortized Variational Inference

What's the problem?

for each x_i (or mini-batch):

calculate $\nabla_{\theta} \mathcal{L}_i(p, q_i)$:

sample $z \sim q_i(z)$

$$\nabla_{\theta} \mathcal{L}_i(p, q_i) \approx \nabla_{\theta} \log p_{\theta}(x_i|z)$$

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L}_i(p, q_i)$$

update q_i to maximize $\mathcal{L}_i(p, q_i)$

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$

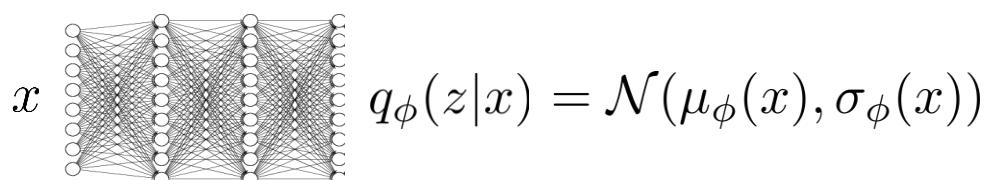
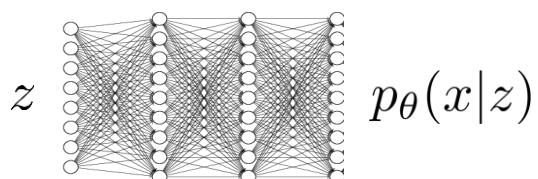
gradient ascent on μ_i, σ_i

How many parameters are there?

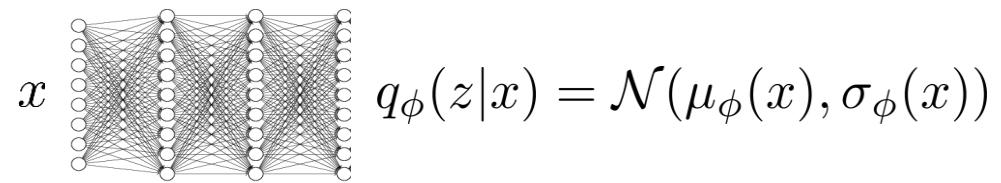
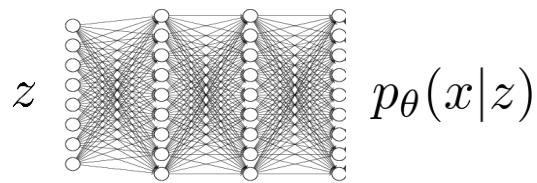
$$|\theta| + (|\mu_i| + |\sigma_i|) \times N$$

intuition: $q_i(z)$ should approximate $p(z|x_i)$

what if we learn a *network* $q_i(z) = q(z|x_i) \approx p(z|x_i)$?



Amortized variational inference



for each x_i (or mini-batch):

calculate $\nabla_\theta \mathcal{L}(p_\theta(x_i|z), q_\phi(z|x_i))$:

sample $z \sim q_\phi(z|x_i)$

$\nabla_\theta \mathcal{L} \approx \nabla_\theta \log p_\theta(x_i|z)$

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$$

$$\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}$$

$$\log p(x_i) \geq \underbrace{E_{z \sim q_\phi(z|x_i)} [\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))}_{\mathcal{L}(p_\theta(x_i|z), q_\phi(z|x_i))}$$

how do we calculate this?

Amortized variational inference

for each x_i (or mini-batch):

calculate $\nabla_{\theta} \mathcal{L}(p_{\theta}(x_i|z), q_{\phi}(z|x_i))$:

sample $z \sim q_{\phi}(z|x_i)$

$\nabla_{\theta} \mathcal{L} \approx \nabla_{\theta} \log p_{\theta}(x_i|z)$

$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L}$

$\phi \leftarrow \phi + \alpha \nabla_{\phi} \mathcal{L}$

$$q_{\phi}(z|x) = \mathcal{N}(\mu_{\phi}(x), \sigma_{\phi}(x))$$

$$\mathcal{L}_i = E_{z \sim q_{\phi}(z|x_i)} [\log p_{\theta}(x_i|z) + \log p(z)] + \mathcal{H}(q_{\phi}(z|x_i))$$


$$J(\phi) = E_{z \sim q_{\phi}(z|x_i)} [r(x_i, z)]$$

look up formula for
entropy of a Gaussian



can just use policy gradient!

What's wrong with this gradient?

$$\nabla J(\phi) \approx \frac{1}{M} \sum_j \nabla_{\phi} \log q_{\phi}(z_j|x_i) r(x_i, z_j)$$

The reparameterization trick

Is there a better way?

$$J(\phi) = E_{z \sim q_\phi(z|x_i)}[r(x_i, z)]$$

$$= E_{\epsilon \sim \mathcal{N}(0,1)}[r(x_i, \mu_\phi(x_i) + \epsilon \sigma_\phi(x_i))]$$

estimating $\nabla_\phi J(\phi)$:

sample $\epsilon_1, \dots, \epsilon_M$ from $\mathcal{N}(0, 1)$ (a single sample works well!)

$$\nabla_\phi J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi r(x_i, \mu_\phi(x_i) + \epsilon_j \sigma_\phi(x_i))$$

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$

$$z = \mu_\phi(x) + \epsilon \sigma_\phi(x)$$



$$\epsilon \sim \mathcal{N}(0, 1)$$

independent of ϕ !

most autodiff software (e.g., TensorFlow)
will compute this for you!

Another way to look at it...

$$\mathcal{L}_i = E_{z \sim q_\phi(z|x_i)} [\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))$$

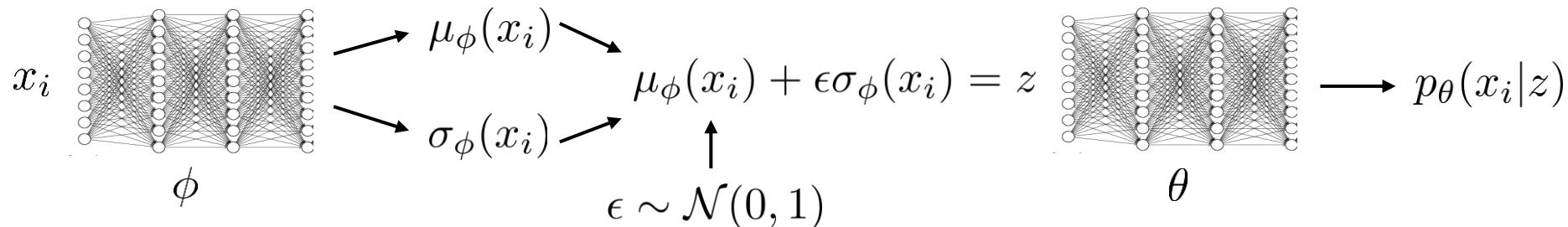
$$= E_{z \sim q_\phi(z|x_i)} [\log p_\theta(x_i|z)] + \underbrace{E_{z \sim q_\phi(z|x_i)} [\log p(z)] + \mathcal{H}(q_\phi(z|x_i))}_{-D_{\text{KL}}(q_\phi(z|x_i) \| p(z))}$$

$-D_{\text{KL}}(q_\phi(z|x_i) \| p(z))$ ← this often has a convenient analytical form (e.g., KL-divergence for Gaussians)

$$= E_{z \sim q_\phi(z|x_i)} [\log p_\theta(x_i|z)] - D_{\text{KL}}(q_\phi(z|x_i) \| p(z))$$

$$= E_{\epsilon \sim \mathcal{N}(0,1)} [\log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i))] - D_{\text{KL}}(q_\phi(z|x_i) \| p(z))$$

$$\approx \log p_\theta(x_i|\mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - D_{\text{KL}}(q_\phi(z|x_i) \| p(z))$$



Reparameterization trick vs. policy gradient

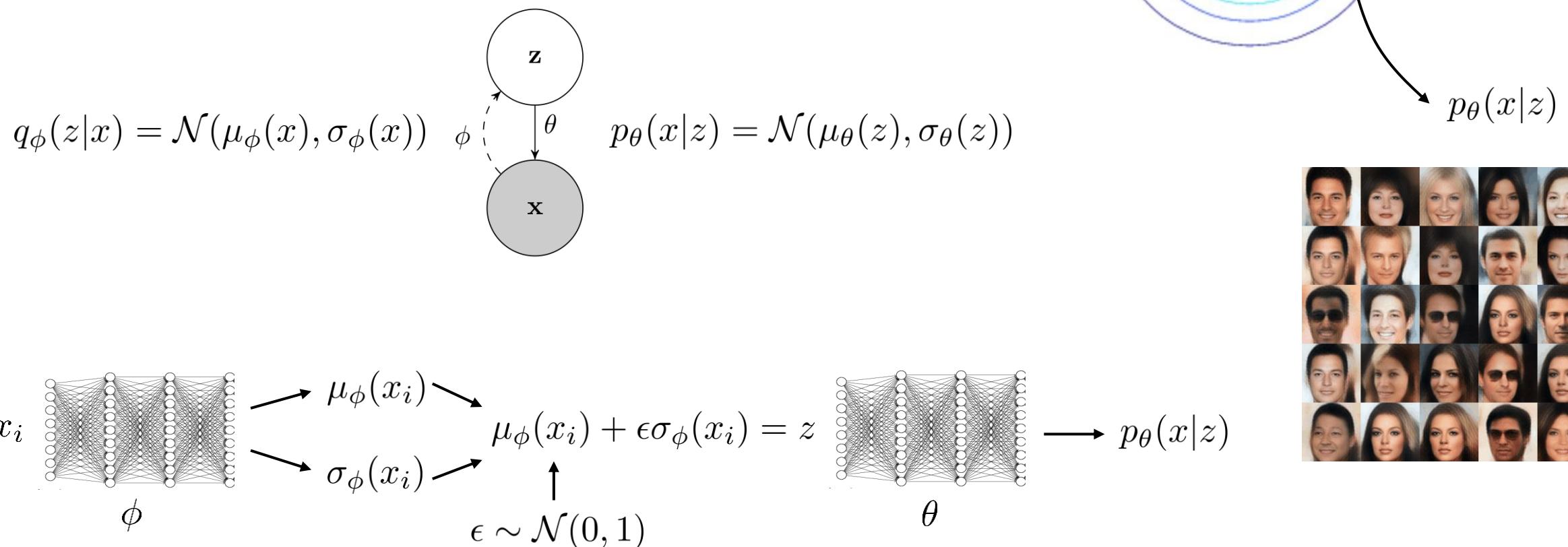
- Policy gradient
 - Can handle both discrete and continuous latent variables
 - High variance, requires multiple samples & small learning rates
- Reparameterization trick
 - Only continuous latent variables
 - Very simple to implement
 - Low variance

$$J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi \log q_\phi(z_j | x_i) r(x_i, z_j)$$

$$\nabla_\phi J(\phi) \approx \frac{1}{M} \sum_j \nabla_\phi r(x_i, \mu_\phi(x_i) + \epsilon_j \sigma_\phi(x_i))$$

Variational Inference in Deep RL

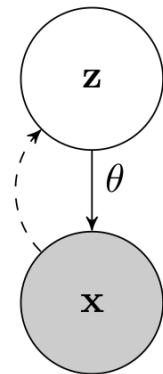
The *variational* autoencoder



$$\max_{\theta, \phi} \frac{1}{N} \sum_i \log p_\theta(x_i | \mu_\phi(x_i) + \epsilon\sigma_\phi(x_i)) - D_{\text{KL}}(q_\phi(z|x_i) \| p(z))$$

Using the variational autoencoder

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$$



$$p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \sigma_\theta(z))$$

$$p(x) = \int p(x|z)p(z)dz$$

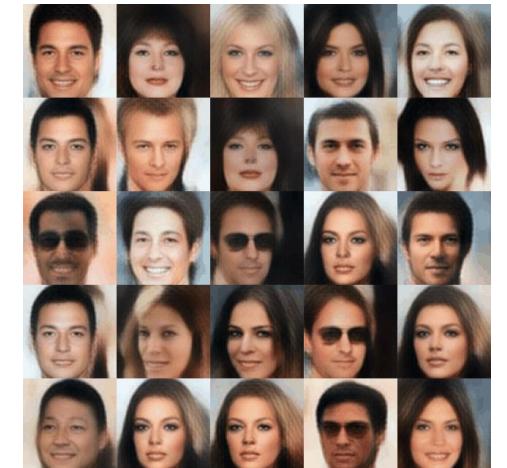
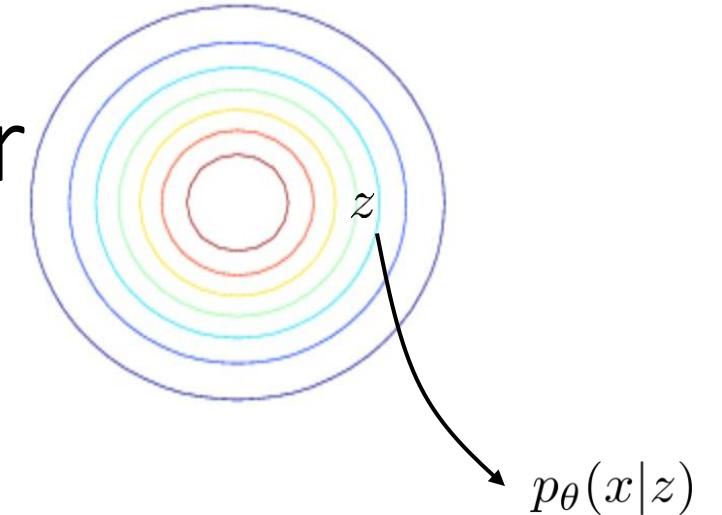
why does this work?

sampling:

$$z \sim p(z)$$

$$x \sim p(x|z)$$

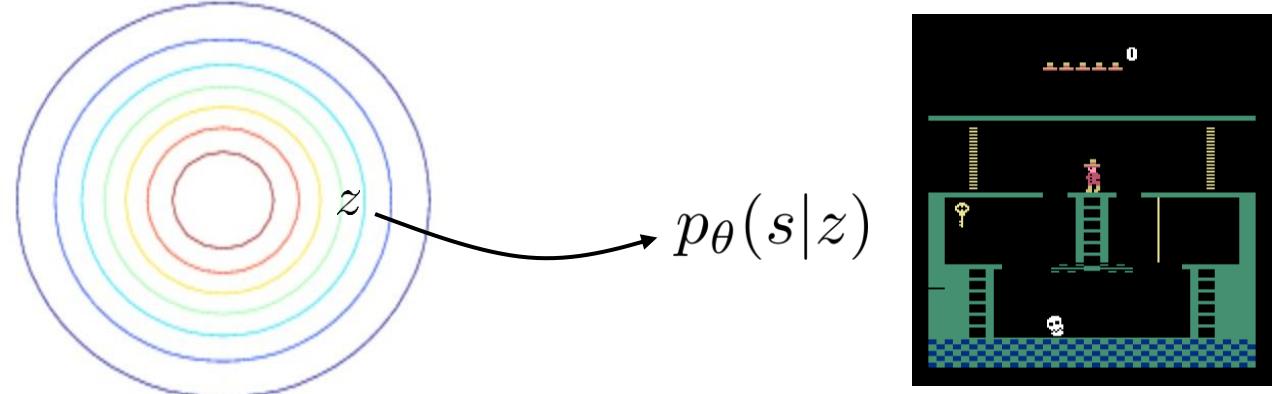
$$\mathcal{L}_i = E_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{\text{KL}}(q_\phi(z|x_i) \| p(z))$$



Example applications

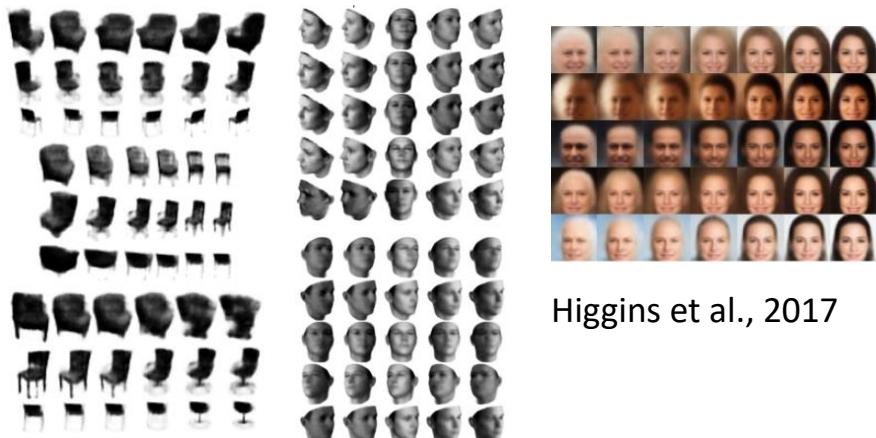
Representation learning

z is a *representation* of s



- 1. Train VAE on states in replay buffer \mathcal{R}
- 2. Run RL, using z as the state instead of s

Why is this a good idea?



Higgins et al., 2017

Sample algorithm

- 1. Collect transition (s, a, s', r) , add to \mathcal{R}
- 2. Update $p_\theta(s|z)$ and $q_\phi(z|s)$ w/ batch from \mathcal{R}
- 3. Update $Q(z, a)$ w/ batch from \mathcal{R}

This also provides a great way to use prior data!

Conditional models

$$\mathcal{L}_i = E_{z \sim q_\phi(z|x_i, y_i)} [\log p_\theta(y_i|x_i, z) + \log p(z|x_i)] + \mathcal{H}(q_\phi(z|x_i, y_i))$$



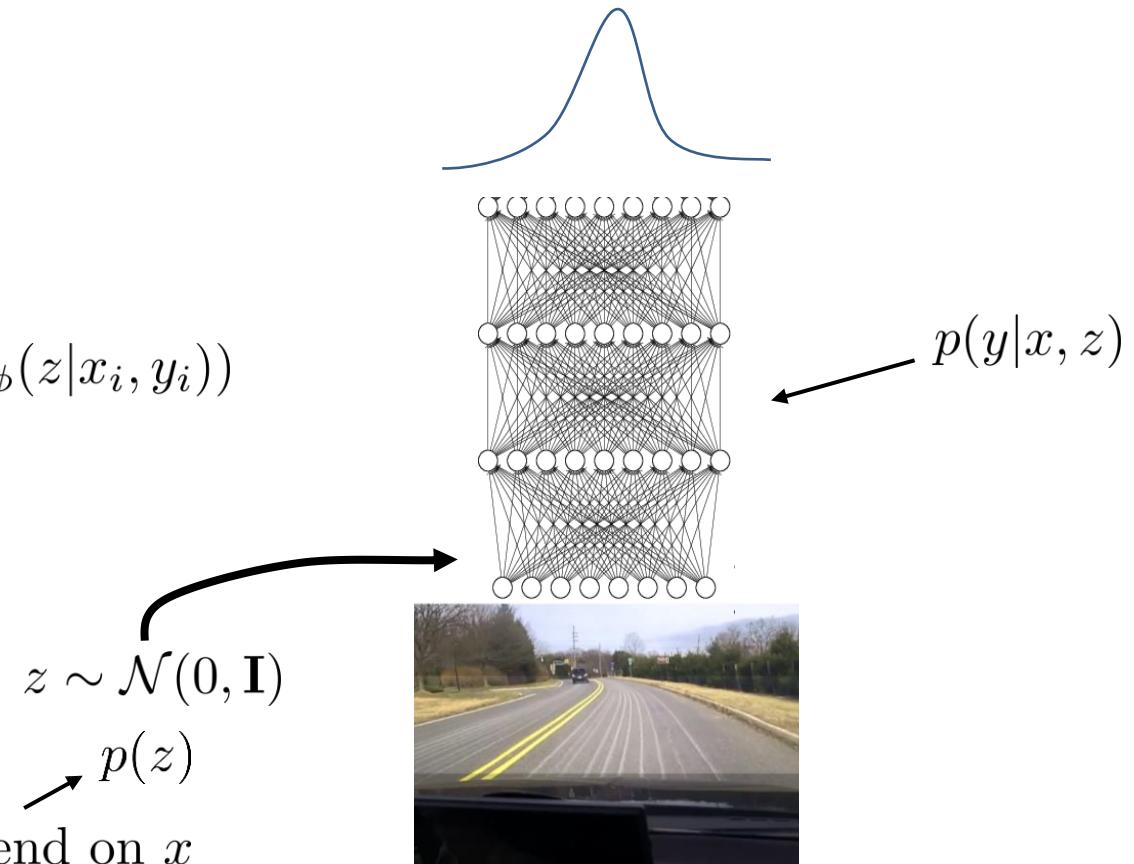
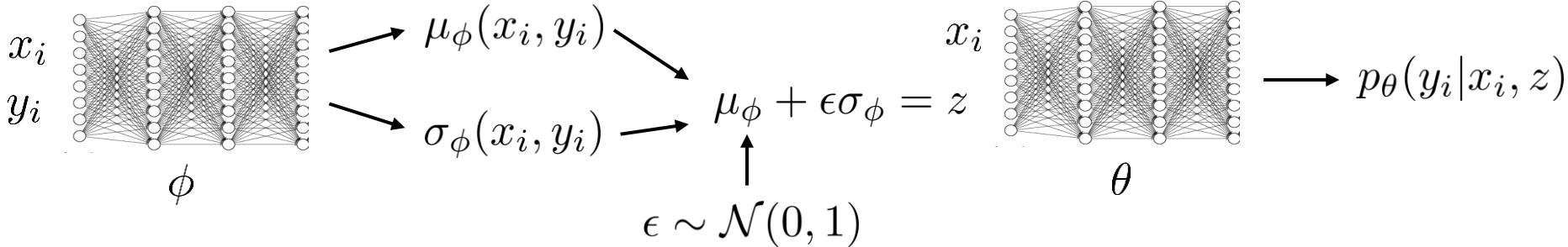
just like before, only now generating y_i
and *everything* is conditioned on x_i

at test time:

$$z \sim p(z|x_i)$$

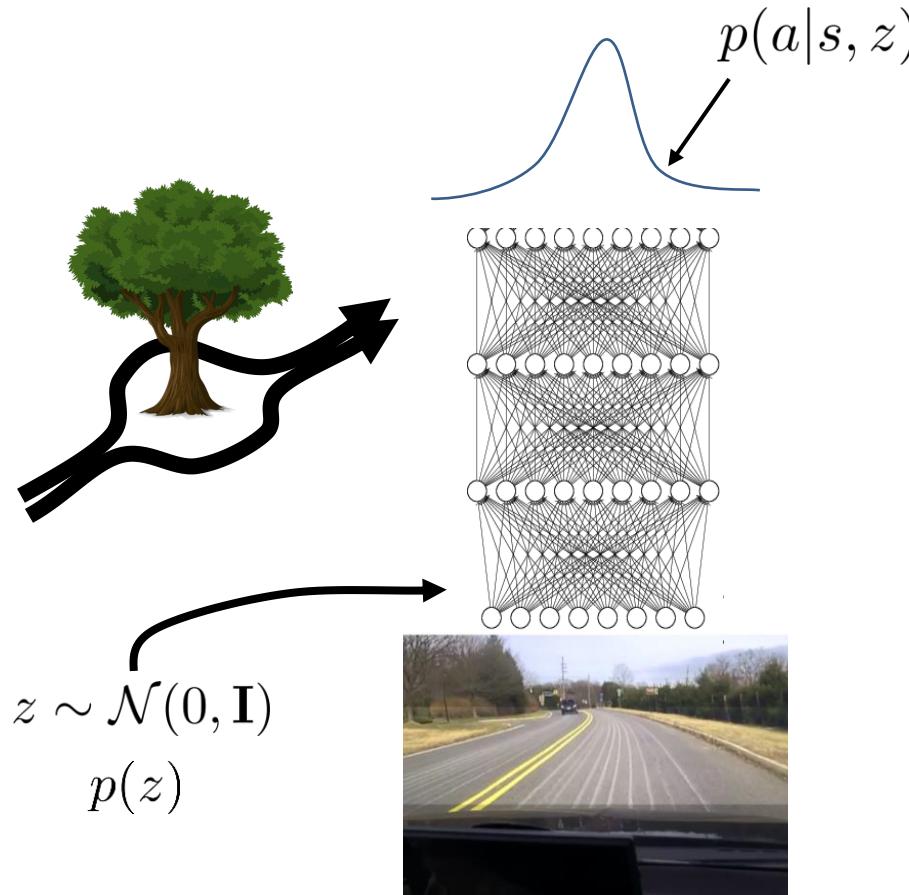
can *optionally* depend on x

$$y \sim p(y|x_i, z)$$



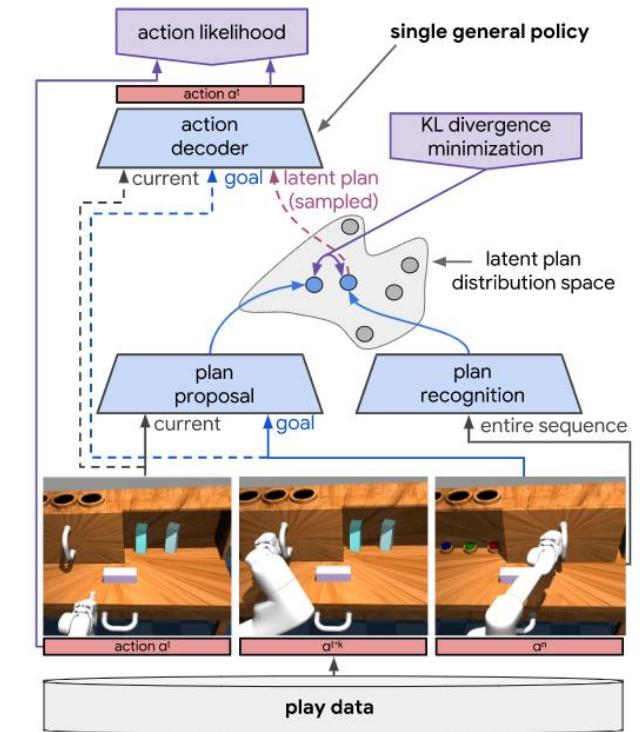
Example applications

Multimodal imitation learning



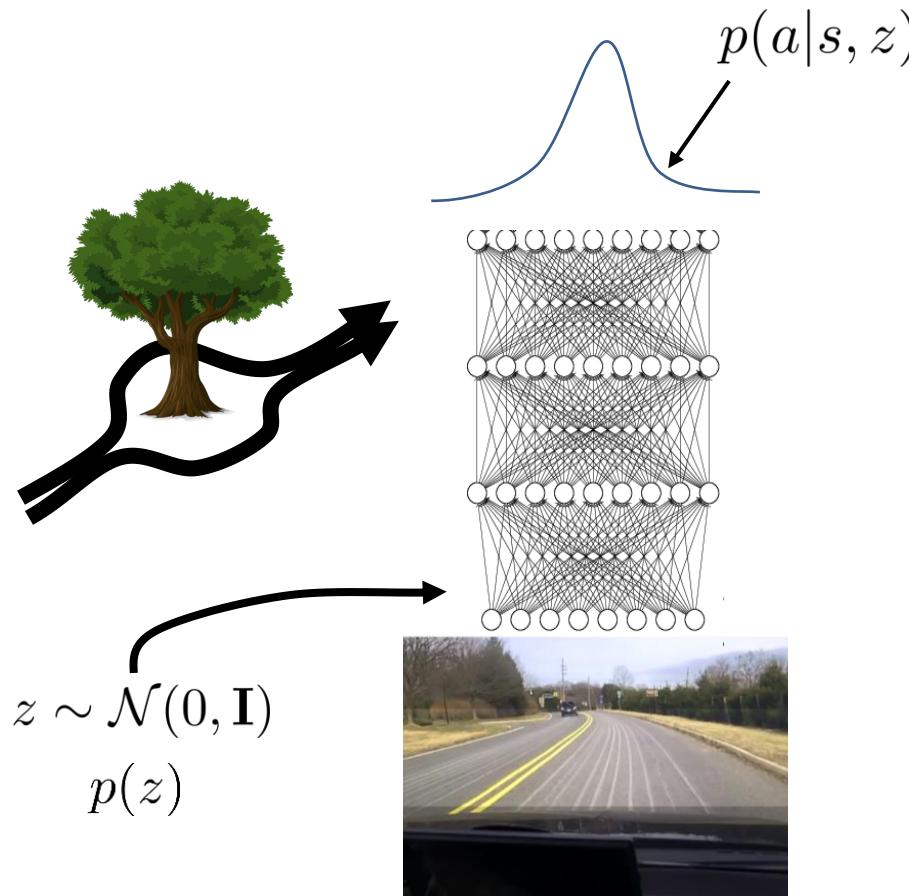
Learning Latent Plans from Play

COREY LYNCH MOHI KHANSARI TED XIAO VIKASH KUMAR JONATHAN TOMPSON SERGEY LEVINE PIERRE SERMANET
Google Brain Google X Google Brain Google Brain Google Brain Google Brain Google Brain



Example applications

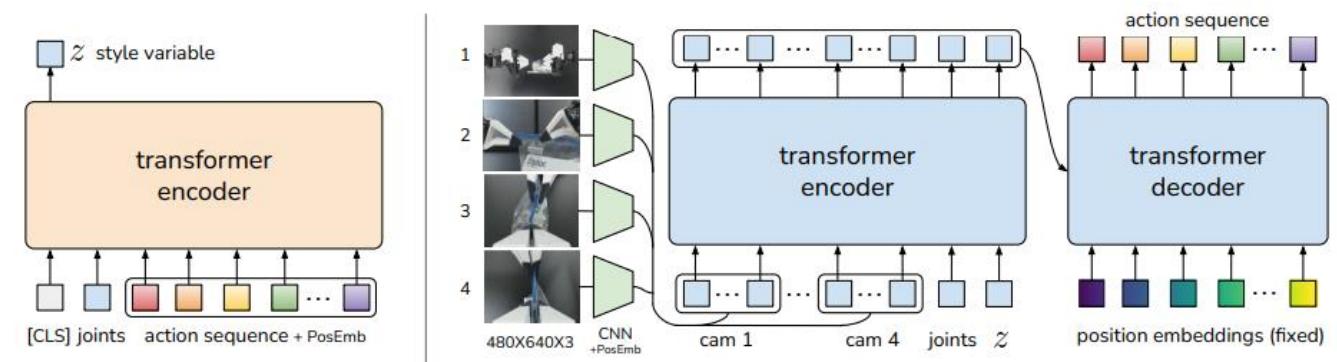
Multimodal imitation learning



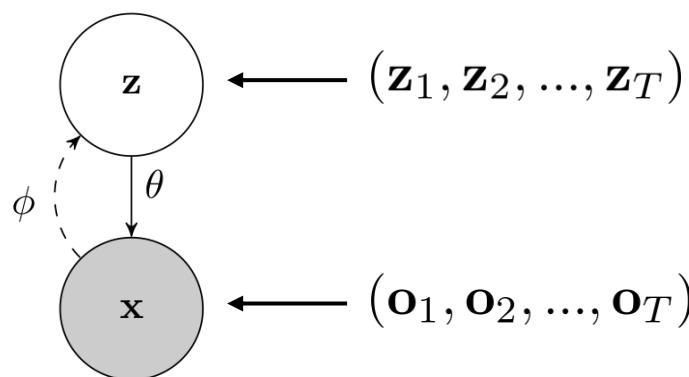
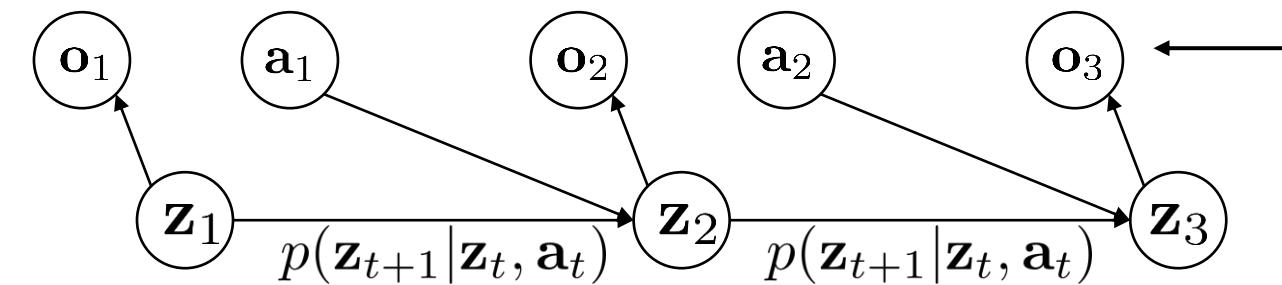
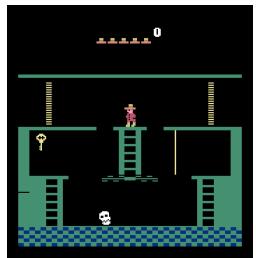
Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

Tony Z. Zhao¹ Vikash Kumar³ Sergey Levine² Chelsea Finn¹

¹ Stanford University ² UC Berkeley ³ Meta



State space models



What is the **prior**?

What is the **decoder**?

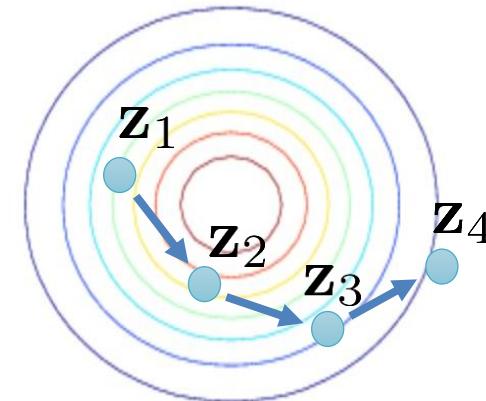
What is the **encoder**?

we are not in partially observed setting

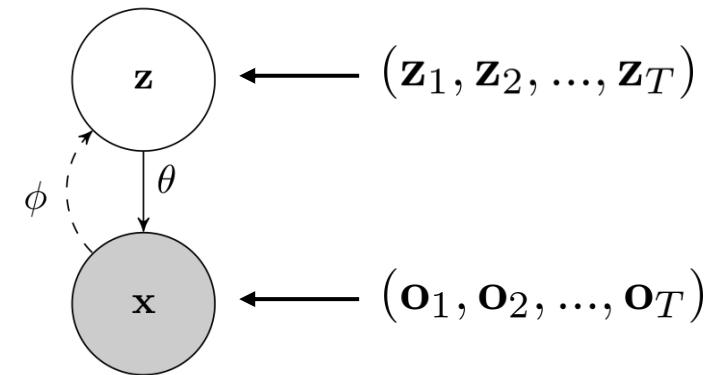
$$\mathcal{N}(0, \mathbf{I}) \xrightarrow{\text{learned}} p(\mathbf{z}) = p(\mathbf{z}_1) \prod_t p(\mathbf{z}_{t+1} | \mathbf{z}_t, \mathbf{a}_t)$$

$$p_\theta(\mathbf{o}|\mathbf{z}) = \prod_t p(\mathbf{o}_t | \mathbf{z}_t)$$

$$q_\phi(\mathbf{z}|\mathbf{o}) = \prod_t q_\phi(\mathbf{z}_t | \mathbf{o}_{1:t})$$

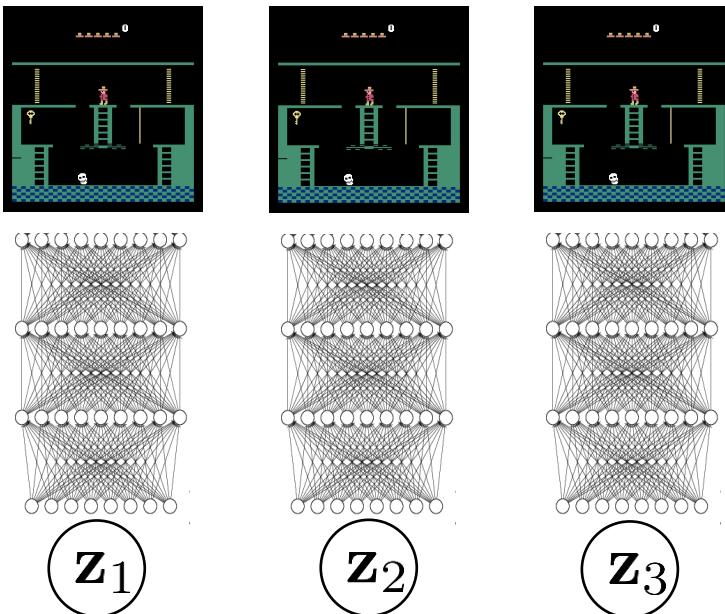


State space models



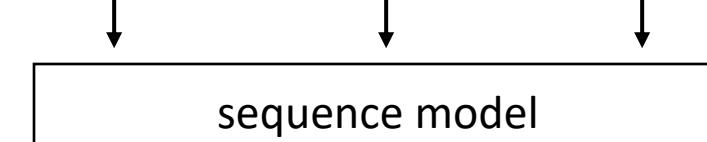
What is the **decoder**?

$$p_{\theta}(\mathbf{o}|z) = \prod_t p(\mathbf{o}_t|z_t)$$



What is the **encoder**?

$$q_{\phi}(z|o) = \prod_t q_{\phi}(z_t|o_{1:t})$$



LSTM
Transformer



Example applications

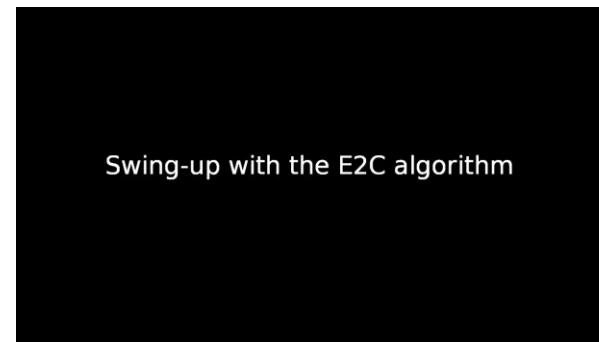
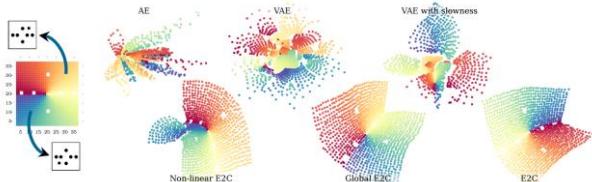
Representation learning and model-based RL

1. Learn state space model and plan in the latent space

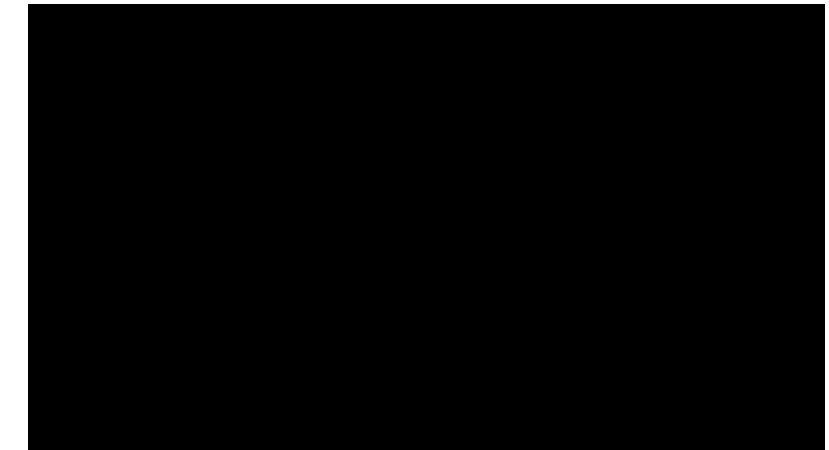
Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images

Manuel Watter* Jost Tobias Springenberg*
Joschka Boedecker
University of Freiburg, Germany
{watterm,springj,jboedeck}@cs.uni-freiburg.de

Martin Riedmiller
Google DeepMind
London, UK
riedmiller@google.com

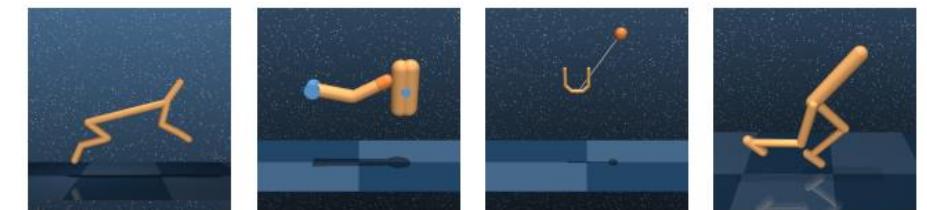


SOLAR: Deep Structured Latent Representations for Model-Based Reinforcement Learning



Learning Latent Dynamics for Planning from Pixels

Danijar Hafner ^{1,2} Timothy Lillicrap ³ Ian Fischer ⁴ Ruben Villegas ^{1,5}
David Ha ¹ Honglak Lee ¹ James Davidson ¹



(c) Cheetah

(d) Finger

(e) Cup

(f) Walker

Example applications

Representation learning and model-based RL

1. Learn state space model and run RL in the state space

Stochastic Latent Actor-Critic: Deep Reinforcement Learning with a Latent Variable Model

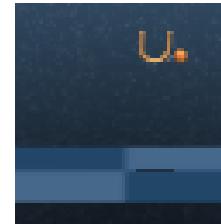
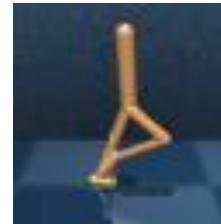
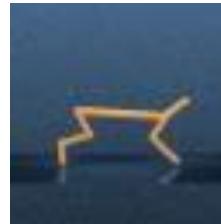
Alex X. Lee^{1,2} Anusha Nagabandi¹ Pieter Abbeel¹ Sergey Levine¹

¹University of California, Berkeley

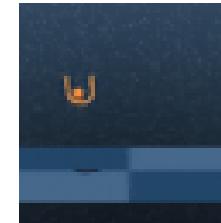
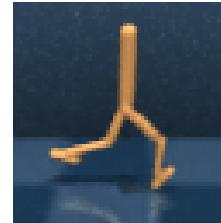
²DeepMind

{alexlee_gk,nagabani2,pabbeel,svlevine}@cs.berkeley.edu

true
rollouts



samples



DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION

Danijar Hafner *

University of Toronto

Google Brain

Timothy Lillicrap

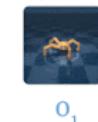
DeepMind

Jimmy Ba

University of Toronto

Mohammad Norouzi

Google Brain



Reframing Control as an Inference Problem

CS 285

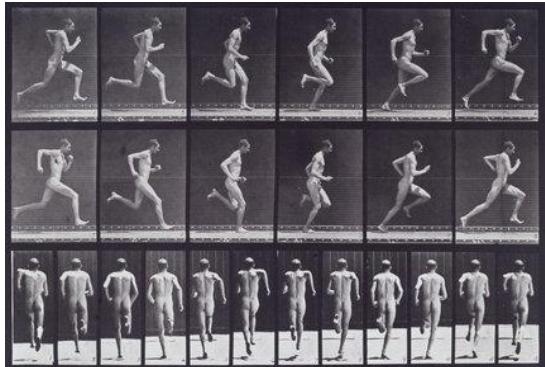
Instructor: Sergey Levine
UC Berkeley



Today's Lecture

1. Does reinforcement learning and optimal control provide a reasonable model of human behavior?
 2. Is there a better explanation?
 3. Can we derive optimal control, reinforcement learning, and planning as *probabilistic inference*?
 4. How does this change our RL algorithms?
 5. (next lecture) We'll see this is crucial for *inverse reinforcement learning*
- Goals:
 - Understand the connection between inference and control
 - Understand how specific RL algorithms can be instantiated in this framework
 - Understand why this might be a good idea

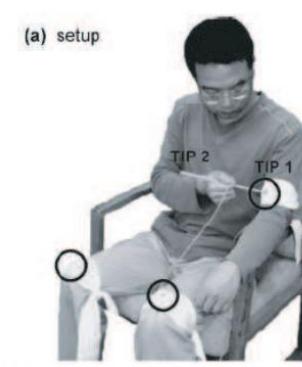
Optimal Control as a Model of Human Behavior



Muybridge (c. 1870)



Mombaur et al. '09



Li & Todorov '06



Ziebart '08

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)$$

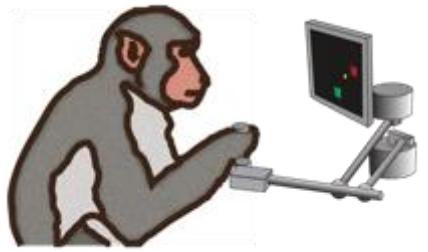
$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

optimize this to explain the data

$$\pi = \arg \max_{\pi} E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

$$\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)$$

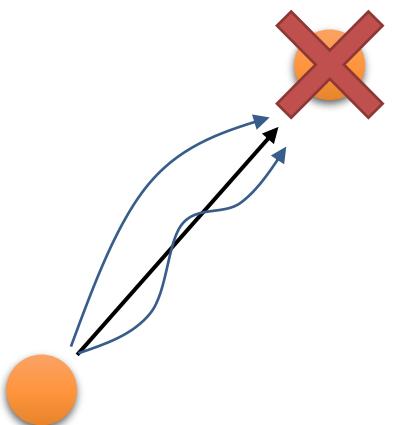
What if the data is not optimal?



some mistakes matter more than others!

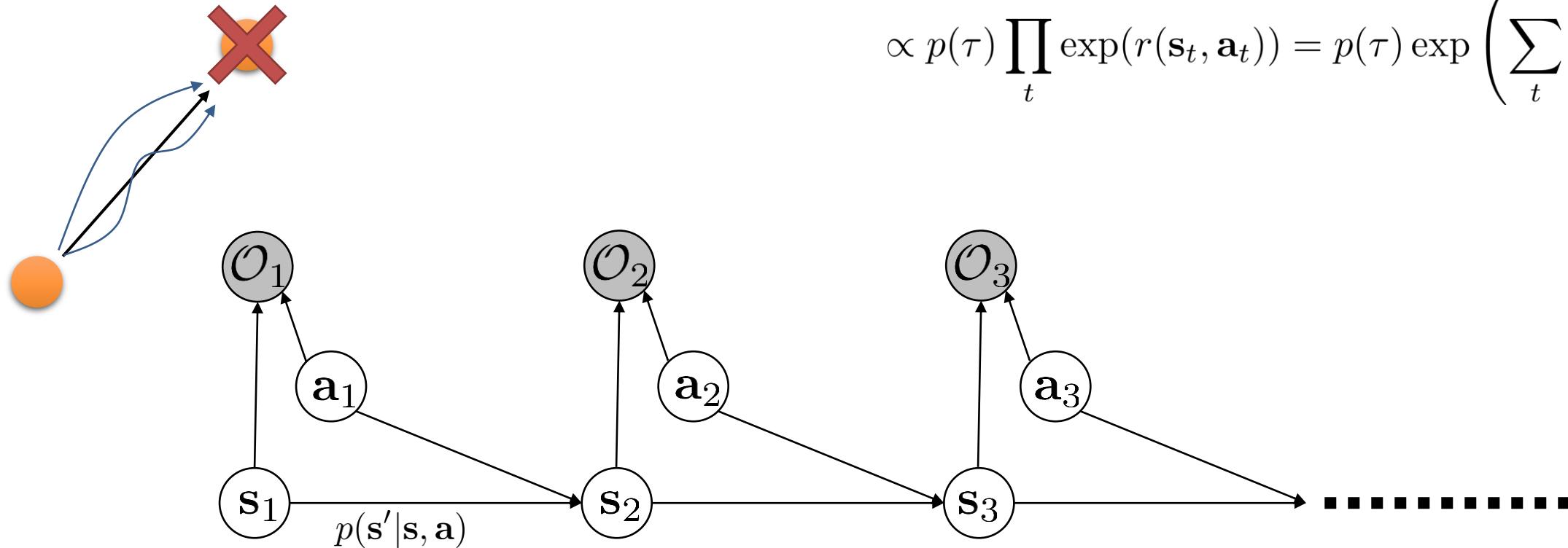
behavior is **stochastic**

but good behavior is still the most likely



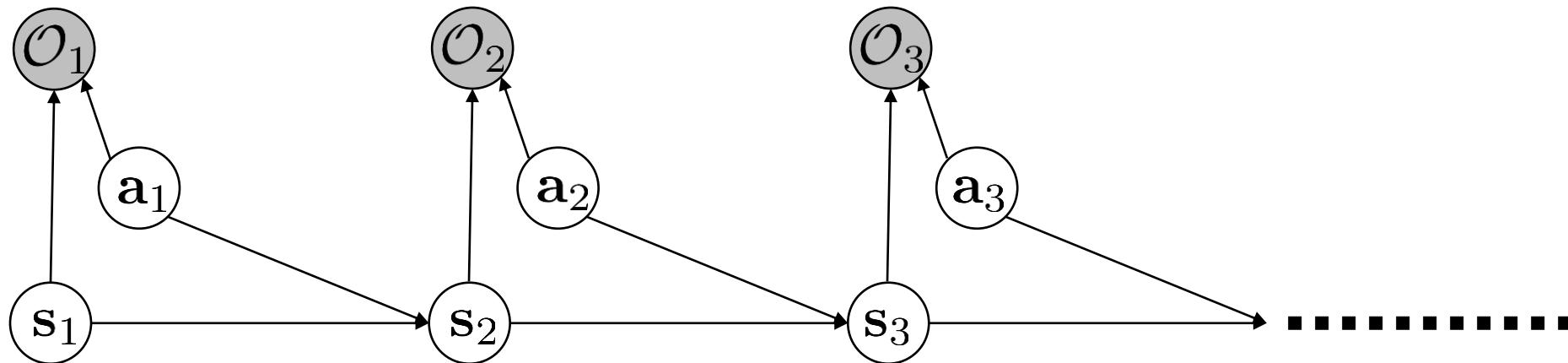
A probabilistic graphical model of decision making

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)$$
$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$



$$p(\underbrace{\mathbf{s}_{1:T}, \mathbf{a}_{1:T}}_{\tau}) = ?? \quad \text{no assumption of optimal behavior!}$$
$$p(\tau | \mathcal{O}_{1:T})$$
$$p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$$
$$p(\tau | \mathcal{O}_{1:T}) = \frac{p(\tau, \mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})}$$
$$\propto p(\tau) \prod_t \exp(r(\mathbf{s}_t, \mathbf{a}_t)) = p(\tau) \exp \left(\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right)$$

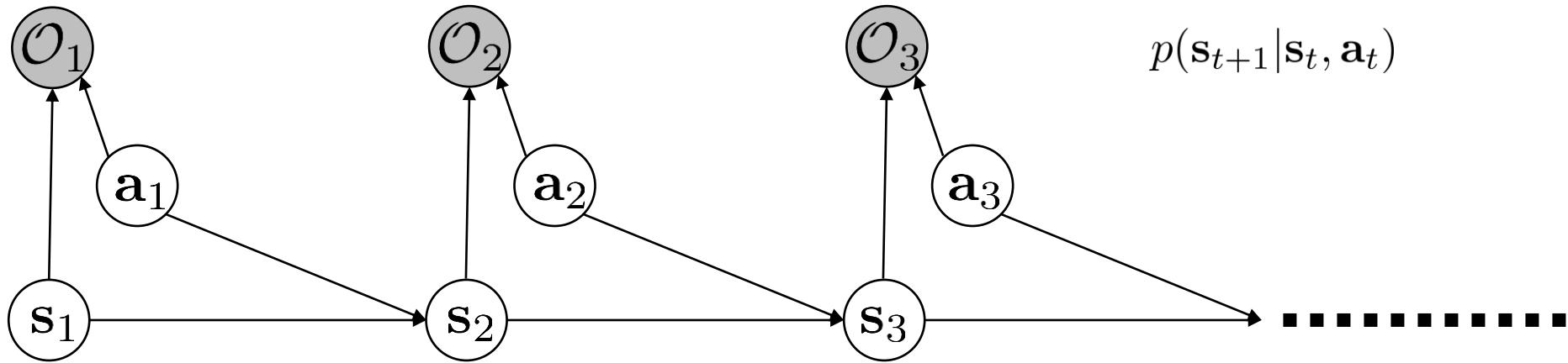
Why is this interesting?



- Can model suboptimal behavior (important for inverse RL)
- Can apply inference algorithms to solve control and planning problems
- Provides an explanation for why stochastic behavior might be preferred (useful for exploration and transfer learning)

Inference = planning

$$p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{s}_t, \mathbf{a}_t))$$



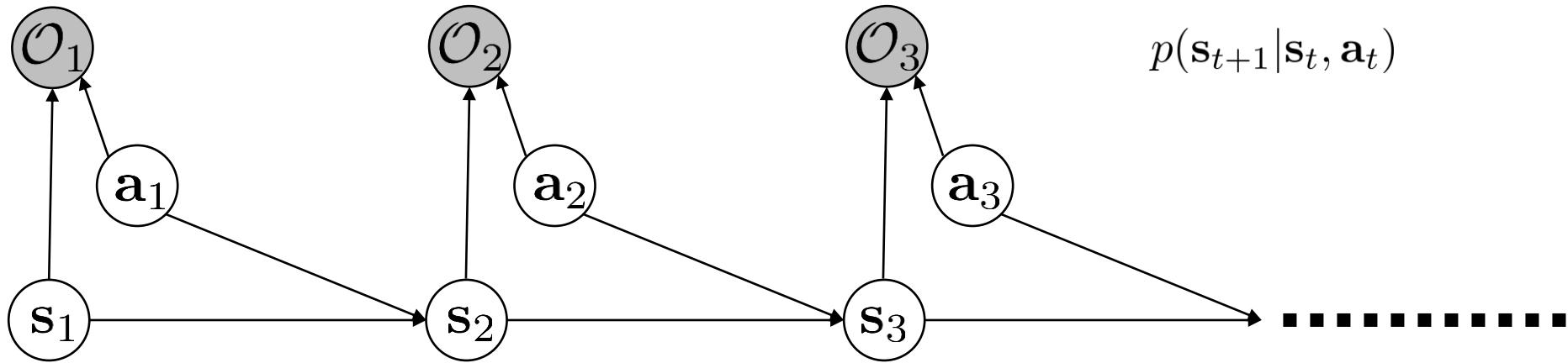
how to do inference?

1. compute backward messages $\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T} | \mathbf{s}_t, \mathbf{a}_t)$
2. compute policy $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T})$
3. compute forward messages $\alpha_t(\mathbf{s}_t) = p(\mathbf{s}_t | \mathcal{O}_{1:t-1})$

Control as Inference

Inference = planning

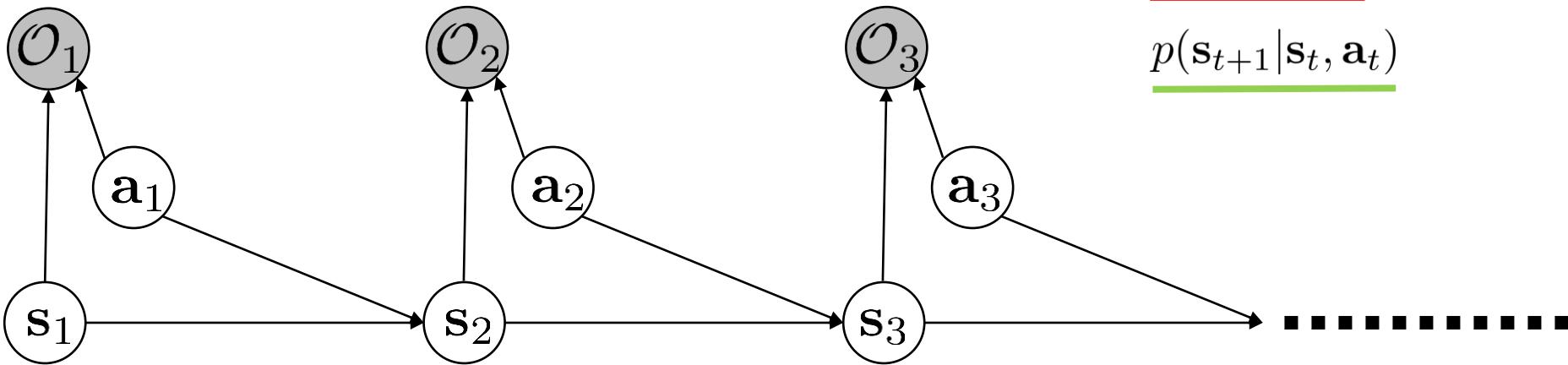
$$p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{s}_t, \mathbf{a}_t))$$



how to do inference?

1. compute backward messages $\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T} | \mathbf{s}_t, \mathbf{a}_t)$
2. compute policy $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T})$
3. compute forward messages $\alpha_t(\mathbf{s}_t) = p(\mathbf{s}_t | \mathcal{O}_{1:t-1})$

Backward messages



$$\frac{p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{s}_t, \mathbf{a}_t))}{p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}$$

$$\begin{aligned}
 \beta_t(\mathbf{s}_t, \mathbf{a}_t) &= p(\mathcal{O}_{t:T} | \mathbf{s}_t, \mathbf{a}_t) \\
 &= \int p(\mathcal{O}_{t:T}, \mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_{t+1} \\
 &= \int p(\mathcal{O}_{t+1:T} | \mathbf{s}_{t+1}) \underbrace{p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}_{\text{for } t = T-1 \text{ to } 1:} \underbrace{p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t)}_{\rightarrow} d\mathbf{s}_{t+1} \rightarrow \beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\beta_{t+1}(\mathbf{s}_{t+1})] \\
 p(\mathcal{O}_{t+1:T} | \mathbf{s}_{t+1}) &= \int \underbrace{p(\mathcal{O}_{t+1:T} | \mathbf{s}_{t+1}, \mathbf{a}_{t+1})}_{\beta_t(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})} \underbrace{p(\mathbf{a}_{t+1} | \mathbf{s}_{t+1})}_{\uparrow} d\mathbf{a}_{t+1} \rightarrow \beta_t(\mathbf{s}_t) = E_{\mathbf{a}_t \sim p(\mathbf{a}_t | \mathbf{s}_t)} [\beta_t(\mathbf{s}_t, \mathbf{a}_t)]
 \end{aligned}$$

which actions are likely *a priori*
(assume uniform for now)

A closer look at the backward pass

for $t = T - 1$ to 1:

$$\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\beta_{t+1}(\mathbf{s}_{t+1})]$$

$$\beta_t(\mathbf{s}_t) = E_{\mathbf{a}_t \sim p(\mathbf{a}_t | \mathbf{s}_t)} [\beta_t(\mathbf{s}_t, \mathbf{a}_t)]$$

let $V_t(\mathbf{s}_t) = \log \beta_t(\mathbf{s}_t)$

let $Q_t(\mathbf{s}_t, \mathbf{a}_t) = \log \beta_t(\mathbf{s}_t, \mathbf{a}_t)$

$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t)) d\mathbf{a}_t$$

$V_t(\mathbf{s}_t) \rightarrow \max_{\mathbf{a}_t} Q_t(\mathbf{s}_t, \mathbf{a}_t)$ as $Q_t(\mathbf{s}_t, \mathbf{a}_t)$ gets bigger!

value iteration algorithm:

- 
1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
 2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

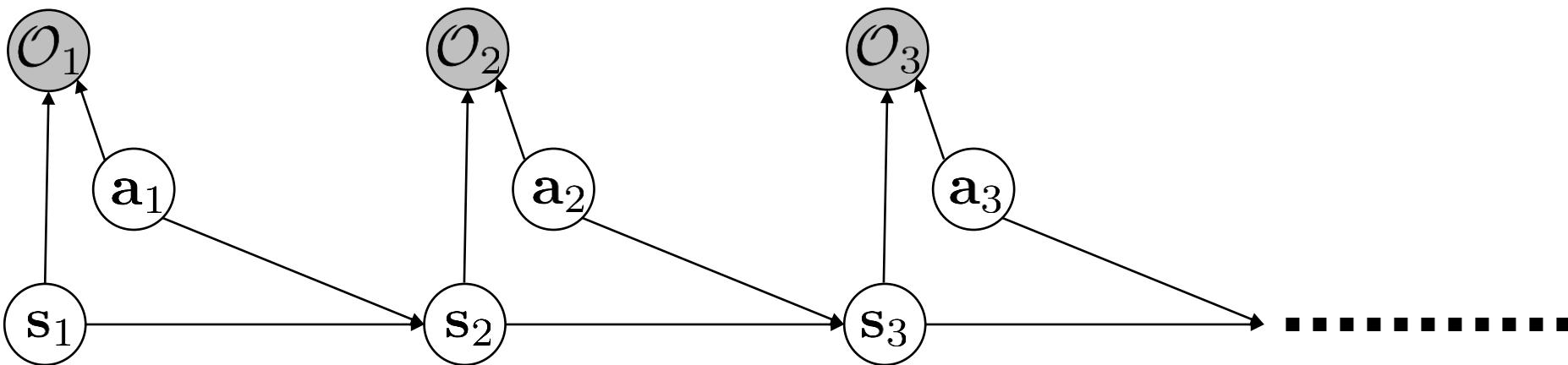
“optimistic” transition
(not a good idea!)

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \overbrace{\log E[\exp(V_{t+1}(\mathbf{s}_{t+1}))]}$$

deterministic transition: $Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + V_{t+1}(\mathbf{s}_{t+1})$

we'll come back to the stochastic case later!

Backward pass summary



$$\beta_t(s_t, a_t) = p(\mathcal{O}_{t:T} | s_t, a_t)$$

probability that we can be optimal at steps t through T given that we take action a_t in state s_t

for $t = T - 1$ to 1:

$$\beta_t(s_t, a_t) = p(\mathcal{O}_t | s_t, a_t) E_{s_{t+1} \sim p(s_{t+1} | s_t, a_t)} [\beta_{t+1}(s_{t+1})] \quad \text{compute recursively from } t = T \text{ to } t = 1$$

$$\beta_t(s_t) = E_{a_t \sim p(a_t | s_t)} [\beta_t(s_t, a_t)]$$

$$\text{let } V_t(s_t) = \log \beta_t(s_t)$$

$$\text{let } Q_t(s_t, a_t) = \log \beta_t(s_t, a_t)$$

log of β_t is “ Q -function-like”

The action prior

remember this?

$$p(\mathcal{O}_{t+1:T} | \mathbf{s}_{t+1}) = \int \underbrace{p(\mathcal{O}_{t+1:T} | \mathbf{s}_{t+1}, \mathbf{a}_{t+1}) p(\mathbf{a}_{t+1} | \mathbf{s}_{t+1}) d\mathbf{a}_{t+1}}_{\beta_t(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})} \uparrow \quad (\text{"soft max"})$$

what if the action prior is not uniform?

$$V(\mathbf{s}_t) = \log \int \exp(Q(\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{a}_t | \mathbf{s}_t)) \mathbf{a}_t$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log E[\exp(V(\mathbf{s}_{t+1}))]$$

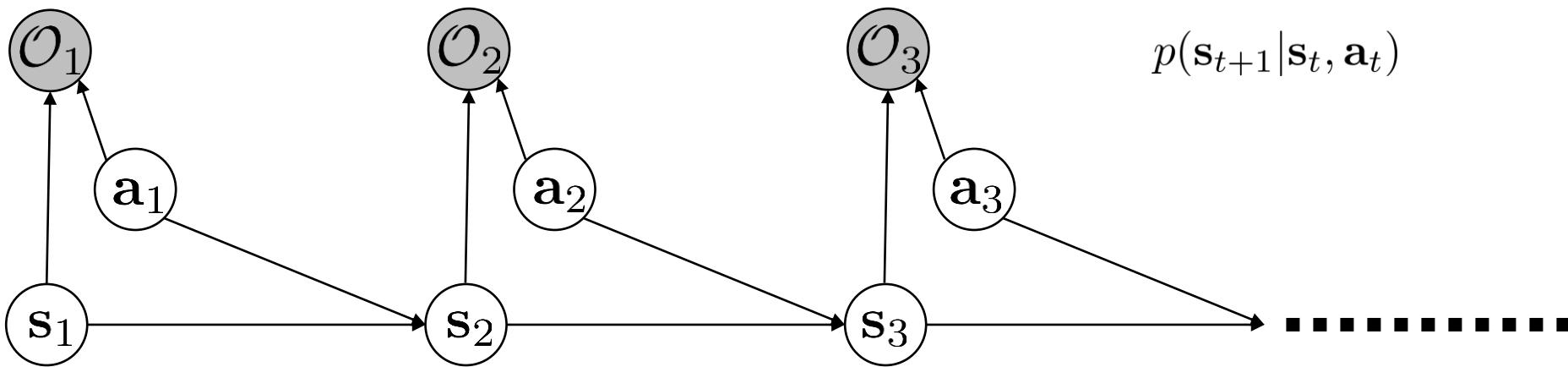
$$\text{let } \tilde{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{a}_t | \mathbf{s}_t) + \log E[\exp(V(\mathbf{s}_{t+1}))]$$

$$V(\mathbf{s}_t) = \log \int \exp(\tilde{Q}(\mathbf{s}_t, \mathbf{a}_t)) \mathbf{a}_t \Leftrightarrow V(\mathbf{s}_t) = \log \int \exp(Q(\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{a}_t | \mathbf{s}_t)) \mathbf{a}_t$$

can **always** fold the action prior into the reward! uniform action prior
can be assumed without loss of generality

Policy computation

$$p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{s}_t, \mathbf{a}_t))$$



2. compute policy $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T})$

$$\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T} | \mathbf{s}_t, \mathbf{a}_t)$$

$$p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T}) = \pi(\mathbf{a}_t | \mathbf{s}_t)$$

$$\beta_t(\mathbf{s}_t) = p(\mathcal{O}_{t:T} | \mathbf{s}_t)$$

$$= p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{t:T})$$

$$= \frac{p(\mathbf{a}_t, \mathbf{s}_t | \mathcal{O}_{t:T})}{p(\mathbf{s}_t | \mathcal{O}_{t:T})}$$

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \frac{\beta_t(\mathbf{s}_t, \mathbf{a}_t)}{\beta_t(\mathbf{s}_t)}$$

$$= \frac{p(\mathcal{O}_{t:T} | \mathbf{a}_t, \mathbf{s}_t) p(\mathbf{a}_t, \mathbf{s}_t) / p(\mathcal{O}_{t:T})}{p(\mathcal{O}_{t:T} | \mathbf{s}_t) p(\mathbf{s}_t) / p(\mathcal{O}_{t:T})}$$

$$= \frac{p(\mathcal{O}_{t:T} | \mathbf{a}_t, \mathbf{s}_t)}{p(\mathcal{O}_{t:T} | \mathbf{s}_t)} \frac{p(\mathbf{a}_t, \mathbf{s}_t)}{p(\mathbf{s}_t)} = \frac{\beta_t(\mathbf{s}_t, \mathbf{a}_t)}{\beta_t(\mathbf{s}_t)} p(\mathbf{a}_t | \mathbf{s}_t)$$

Policy computation with value functions

for $t = T - 1$ to 1:

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log E[\exp(V_{t+1}(\mathbf{s}_{t+1}))]$$

$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t)) \mathbf{a}_t$$

$$\begin{aligned}\pi(\mathbf{a}_t | \mathbf{s}_t) &= \frac{\beta_t(\mathbf{s}_t, \mathbf{a}_t)}{\beta_t(\mathbf{s}_t)} & V_t(\mathbf{s}_t) &= \log \beta_t(\mathbf{s}_t) \\ && Q_t(\mathbf{s}_t, \mathbf{a}_t) &= \log \beta_t(\mathbf{s}_t, \mathbf{a}_t)\end{aligned}$$

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t) - V_t(\mathbf{s}_t)) = \exp(A_t(\mathbf{s}_t, \mathbf{a}_t))$$

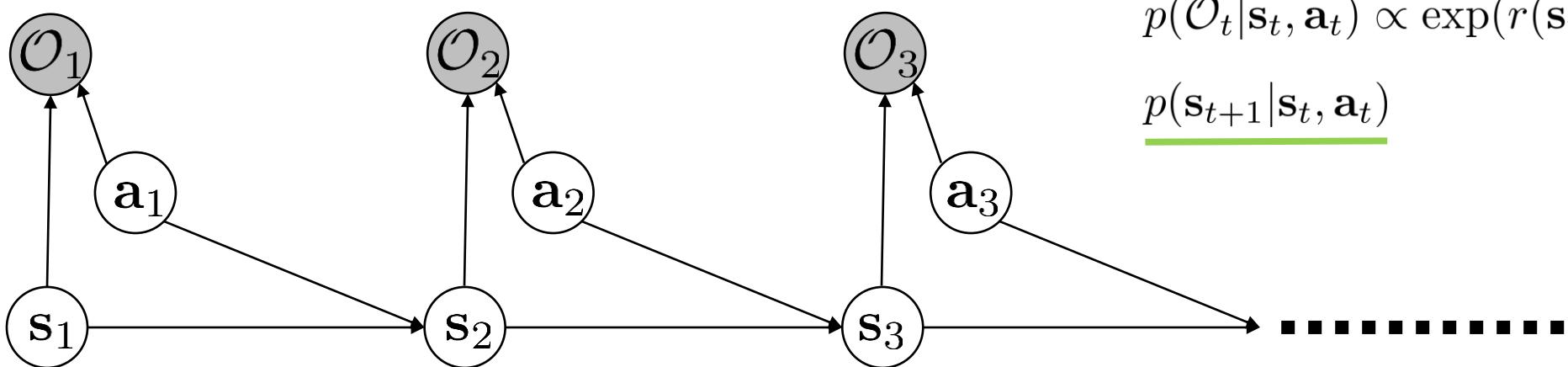
Policy computation summary

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t) - V_t(\mathbf{s}_t)) = \exp(A_t(\mathbf{s}_t, \mathbf{a}_t))$$

with temperature: $\pi(\mathbf{a}_t | \mathbf{s}_t) = \exp\left(\frac{1}{\alpha}Q_t(\mathbf{s}_t, \mathbf{a}_t) - \frac{1}{\alpha}V_t(\mathbf{s}_t)\right) = \exp\left(\frac{1}{\alpha}A_t(\mathbf{s}_t, \mathbf{a}_t)\right)$

- Natural interpretation: better actions are more probable
- Random tie-breaking
- Analogous to Boltzmann exploration
- Approaches greedy policy as temperature decreases

Forward messages



$$\alpha_1(s_1) = p(s_1) \text{ (usually known)}$$

$$\alpha_t(s_t) = p(s_t | O_{1:t-1})$$

$$\begin{aligned}
 &= \int p(s_t, s_{t-1}, a_{t-1} | O_{1:t-1}) d\mathbf{s}_{t-1} d\mathbf{a}_{t-1} = \int p(s_t | s_{t-1}, a_{t-1}, \cancel{O_{1:t-1}}) p(a_{t-1} | s_{t-1}, O_{1:t-1}) p(s_{t-1} | O_{1:t-1}) d\mathbf{s}_{t-1} d\mathbf{a}_{t-1} \\
 &= \int \underline{p(s_t | s_{t-1}, a_{t-1})} p(a_{t-1} | s_{t-1}, O_{t-1}) p(s_{t-1} | O_{1:t-1}) d\mathbf{s}_{t-1} d\mathbf{a}_{t-1} \quad \alpha_{t-1}(s_{t-1})
 \end{aligned}$$

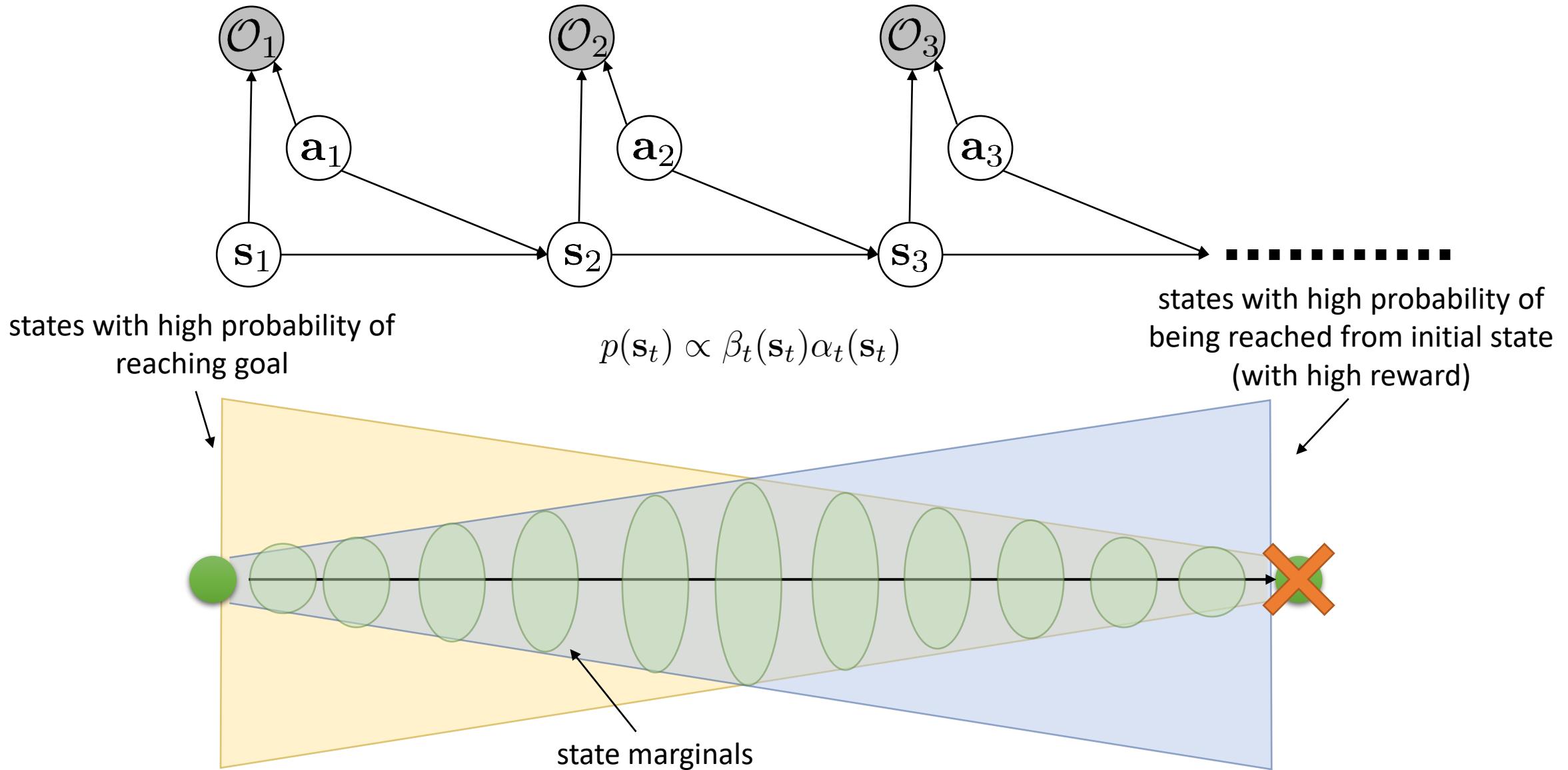
$$p(a_{t-1} | s_{t-1}, O_{t-1}) p(s_{t-1} | O_{1:t-1}) = \frac{p(O_{t-1} | s_{t-1}, a_{t-1}) p(a_{t-1} | s_{t-1})}{p(O_{t-1} | s_{t-1})} \frac{p(O_{t-1} | s_{t-1}) p(s_{t-1} | O_{1:t-2})}{p(O_{t-1} | O_{1:t-2})}$$

what if we want $p(s_t | O_{1:T})$?

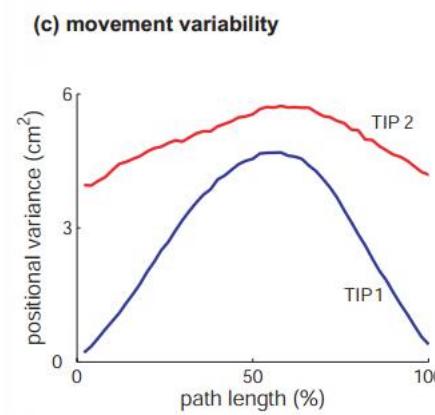
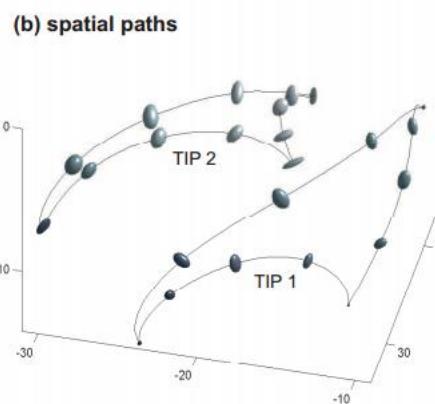
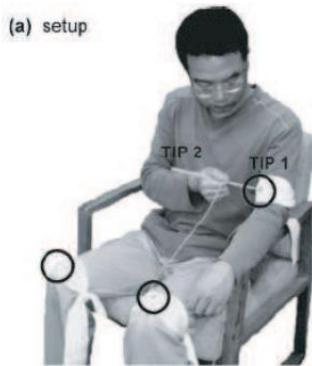
$$\beta_t(s_t)$$

$$p(s_t | O_{1:T}) = \frac{p(s_t, O_{1:T})}{p(O_{1:T})} = \frac{p(O_{1:T} | s_t) p(s_t, O_{1:t-1})}{p(O_{1:T})} \propto \beta_t(s_t) \underline{p(s_t | O_{1:t-1})} p(\cancel{O_{1:t-1}}) \propto \beta_t(s_t) \alpha_t(s_t)$$

Forward/backward message intersection



Forward/backward message intersection

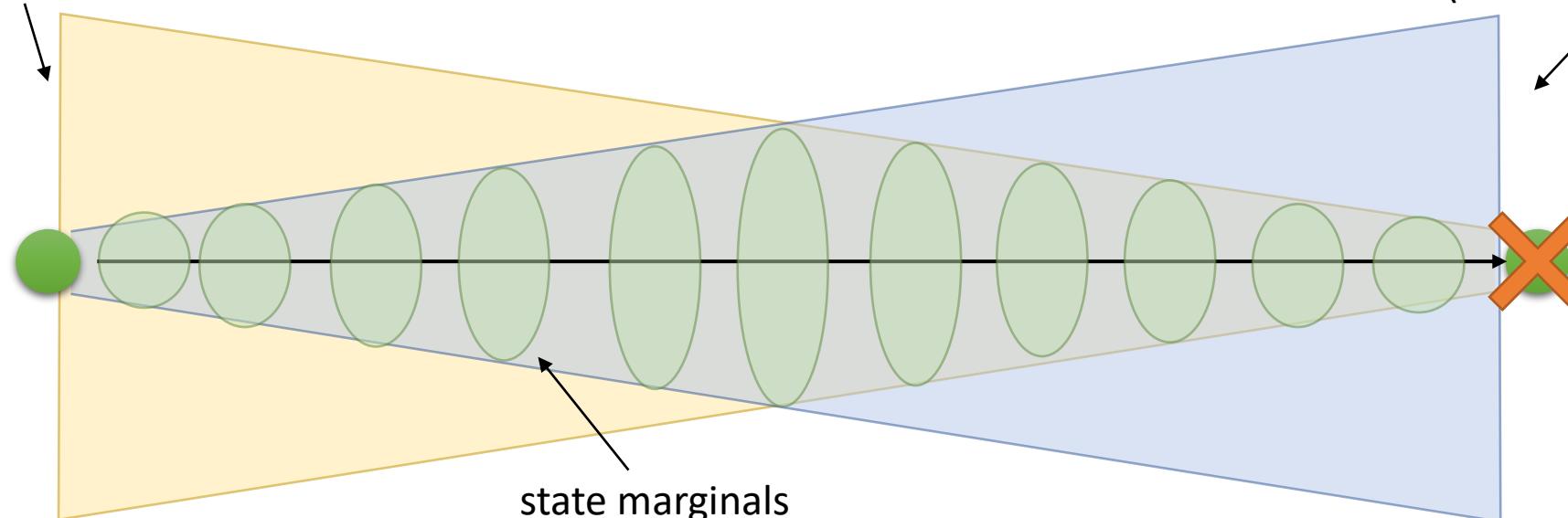


Li & Todorov, 2006

states with high probability of reaching goal

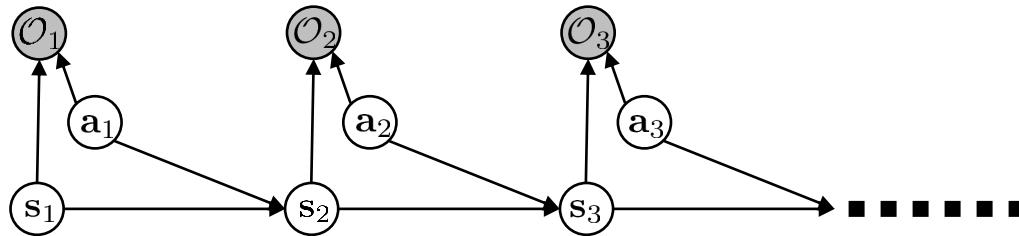
$$p(\mathbf{s}_t) \propto \beta_t(\mathbf{s}_t)\alpha_t(\mathbf{s}_t)$$

states with high probability of being reached from initial state (with high reward)



Summary

1. Probabilistic graphical model for optimal control



2. Control = inference (similar to HMM, EKF, etc.)

3. Very similar to dynamic programming, value iteration, etc. (but “soft”)

Control as Variational Inference

The optimism problem

for $t = T - 1$ to 1:

$$\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [\beta_{t+1}(\mathbf{s}_{t+1})]$$

“optimistic” transition
(not a good idea!)

$$\beta_t(\mathbf{s}_t) = E_{\mathbf{a}_t \sim p(\mathbf{a}_t | \mathbf{s}_t)} [\beta_t(\mathbf{s}_t, \mathbf{a}_t)]$$

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \overbrace{\log E[\exp(V_{t+1}(\mathbf{s}_{t+1}))]}$$

let $V_t(\mathbf{s}_t) = \log \beta_t(\mathbf{s}_t)$

let $Q_t(\mathbf{s}_t, \mathbf{a}_t) = \log \beta_t(\mathbf{s}_t, \mathbf{a}_t)$ why did this happen?

the inference problem: $p(\mathbf{s}_{1:T}, \mathbf{a}_{1:T} | \mathcal{O}_{1:T})$

marginalizing and conditioning, we get: $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T})$ (the policy)

“given that you obtained high reward, what was your action probability?”

marginalizing and conditioning, we get: $p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t, \mathcal{O}_{1:T}) \neq p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$

“given that you obtained high reward, what was your transition probability?”

Addressing the optimism problem

marginalizing and conditioning, we get: $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T})$ (the policy) ← we want this
“given that you obtained high reward, what was your action probability?”

marginalizing and conditioning, we get: $p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t, \mathcal{O}_{1:T}) \neq p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$ ← but not this!
“given that you obtained high reward, what was your transition probability?”

“given that you obtained high reward, what was your action probability,
given that your transition probability did not change?”

can we find another distribution $q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$ that is close to $p(\mathbf{s}_{1:T}, \mathbf{a}_{1:T} | \mathcal{O}_{1:T})$ but has dynamics $p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$

where have we seen this before?

let $\mathbf{x} = \mathcal{O}_{1:T}$ and $\mathbf{z} = (\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$ find $q(\mathbf{z})$ to approximate $p(\mathbf{z} | \mathbf{x})$

let's try variational inference!

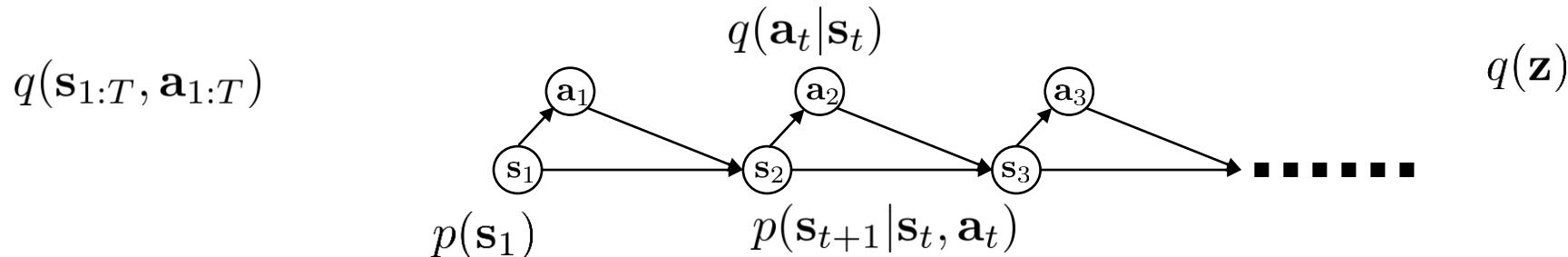
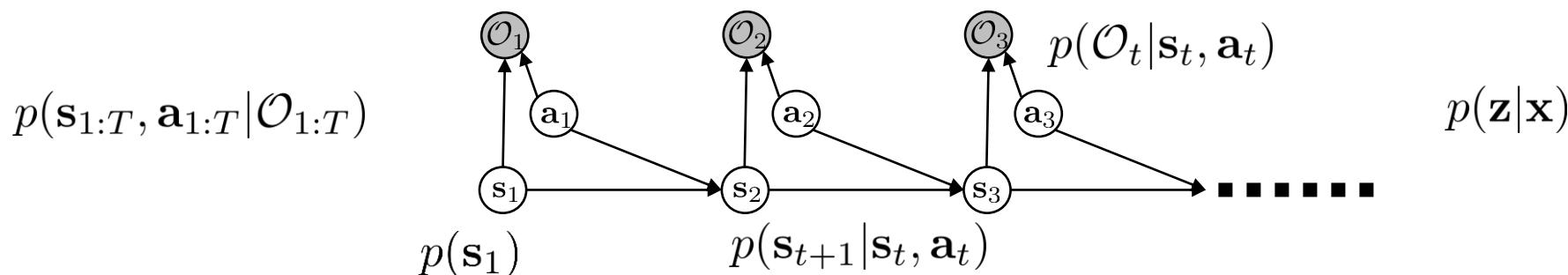
Control via variational inference

$$\text{let } q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) = p(\mathbf{s}_1) \prod_t p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) q(\mathbf{a}_t | \mathbf{s}_t)$$

same dynamics and
initial state as p

only new thing

let $\mathbf{x} = \mathcal{O}_{1:T}$ and $\mathbf{z} = (\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$



The variational lower bound

$$\log p(\mathbf{x}) \geq E_{\mathbf{z} \sim q(\mathbf{z})} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})]$$

let $\mathbf{x} = \mathcal{O}_{1:T}$ and $\mathbf{z} = (\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$

the entropy $\mathcal{H}(q)$

let $q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) = \underline{p(\mathbf{s}_1)} \prod_t \underline{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} q(\mathbf{a}_t|\mathbf{s}_t)$

$$\log p(\mathcal{O}_{1:T}) \geq E_{(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \sim q} [\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^T \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} + \sum_{t=1}^T \log p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t)]$$

$$- \cancel{\log p(\mathbf{s}_1)} - \sum_{t=1}^T \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} - \sum_{t=1}^T \log q(\mathbf{a}_t|\mathbf{s}_t)]$$

$$= E_{(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \sim q} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) - \log q(\mathbf{a}_t|\mathbf{s}_t) \right]$$

$$= \sum_t E_{(\mathbf{s}_t, \mathbf{a}_t) \sim q} [r(\mathbf{s}_t, \mathbf{a}_t) + \mathcal{H}(q(\mathbf{a}_t|\mathbf{s}_t))]$$

maximize reward and maximize action entropy!

Optimizing the variational lower bound

$$\text{let } q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) = p(\mathbf{s}_1) \prod_t p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) q(\mathbf{a}_t | \mathbf{s}_t) \quad \log p(\mathcal{O}_{1:T}) \geq \sum_t E_{(\mathbf{s}_t, \mathbf{a}_t) \sim q} [r(\mathbf{s}_t, \mathbf{a}_t) + \mathcal{H}(q(\mathbf{a}_t | \mathbf{s}_t))]$$

base case: solve for $q(\mathbf{a}_T | \mathbf{s}_T)$:

$$q(\mathbf{a}_T | \mathbf{s}_T) = \arg \max E_{\mathbf{s}_T \sim q(\mathbf{s}_T)} [E_{\mathbf{a}_T \sim q(\mathbf{a}_T | \mathbf{s}_T)} [r(\mathbf{s}_T, \mathbf{a}_T)] + \mathcal{H}(q(\mathbf{a}_T | \mathbf{s}_T))]$$

$$= \arg \max E_{\mathbf{s}_T \sim q(\mathbf{s}_T)} [E_{\mathbf{a}_T \sim q(\mathbf{a}_T | \mathbf{s}_T)} [r(\mathbf{s}_T, \mathbf{a}_T) - \log q(\mathbf{a}_T | \mathbf{s}_T)]]$$

optimized when $q(\mathbf{a}_T | \mathbf{s}_T) \propto \exp(r(\mathbf{s}_T, \mathbf{a}_T))$

$$q(\mathbf{a}_T | \mathbf{s}_T) = \frac{\exp(r(\mathbf{s}_T, \mathbf{a}_T))}{\int \exp(r(\mathbf{s}_T, \mathbf{a})) d\mathbf{a}} = \exp(Q(\mathbf{s}_T, \mathbf{a}_T) - V(\mathbf{s}_T))$$

$$V(\mathbf{s}_T) = \log \int \exp(Q(\mathbf{s}_T, \mathbf{a}_T)) d\mathbf{a}_T$$

$$E_{\mathbf{s}_T \sim q(\mathbf{s}_T)} [E_{\mathbf{a}_T \sim q(\mathbf{a}_T | \mathbf{s}_T)} [r(\mathbf{s}_T, \mathbf{a}_T) - \log q(\mathbf{a}_T | \mathbf{s}_T)]] = E_{\mathbf{s}_T \sim q(\mathbf{s}_T)} [E_{\mathbf{a}_T \sim q(\mathbf{a}_T | \mathbf{s}_T)} [V(\mathbf{s}_T)]]$$

Optimizing the variational lower bound

$$\log p(\mathcal{O}_{1:T}) \geq \sum_t E_{(\mathbf{s}_t, \mathbf{a}_t) \sim q} [r(\mathbf{s}_t, \mathbf{a}_t) + \mathcal{H}(q(\mathbf{a}_t | \mathbf{s}_t))]$$

$$q(\mathbf{a}_T | \mathbf{s}_T) = \frac{\exp(r(\mathbf{s}_T, \mathbf{a}_T))}{\int \exp(r(\mathbf{s}_T, \mathbf{a})) d\mathbf{a}} = \exp(Q(\mathbf{s}_T, \mathbf{a}_T) - V(\mathbf{s}_T))$$

$$E_{\mathbf{s}_T \sim q(\mathbf{s}_T)} [E_{\mathbf{a}_T \sim q(\mathbf{a}_T | \mathbf{s}_T)} [r(\mathbf{s}_T, \mathbf{a}_T) - \log q(\mathbf{a}_T | \mathbf{s}_T)]] = E_{\mathbf{s}_T \sim q(\mathbf{s}_T)} [E_{\mathbf{a}_T \sim q(\mathbf{a}_T | \mathbf{s}_T)} [V(\mathbf{s}_T)]]$$

$$\begin{aligned} q(\mathbf{a}_t | \mathbf{s}_t) &= \arg \max E_{\mathbf{s}_t \sim q(\mathbf{s}_t)} [E_{\mathbf{a}_t \sim q(\mathbf{a}_t | \mathbf{s}_t)} [r(\mathbf{s}_t, \mathbf{a}_t) + E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} [V(\mathbf{s}_{t+1})]] + \mathcal{H}(q(\mathbf{a}_t | \mathbf{s}_t))] \\ &= \arg \max E_{\mathbf{s}_t \sim q(\mathbf{s}_t)} [E_{\mathbf{a}_t \sim q(\mathbf{a}_t | \mathbf{s}_t)} [Q(\mathbf{s}_t, \mathbf{a}_t)] + \mathcal{H}(q(\mathbf{a}_t | \mathbf{s}_t))] \quad \text{regular Bellman backup} \\ &= \arg \max E_{\mathbf{s}_t \sim q(\mathbf{s}_t)} [E_{\mathbf{a}_t \sim q(\mathbf{a}_t | \mathbf{s}_t)} [Q(\mathbf{s}_t, \mathbf{a}_t) - \log q(\mathbf{a}_t | \mathbf{s}_t)]] \quad \text{not optimistic} \\ \text{optimized when } q(\mathbf{a}_t | \mathbf{s}_t) &\propto \exp(Q(\mathbf{s}_t, \mathbf{a}_t)) \end{aligned}$$

$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t)) d\mathbf{a}_t$$

$$q(\mathbf{a}_t | \mathbf{s}_t) = \exp(Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t))$$

Backward pass summary - variational

for $t = T - 1$ to 1:

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + E[(V_{t+1}(\mathbf{s}_{t+1})]$$

$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t)) d\mathbf{a}_t$$

value iteration algorithm:

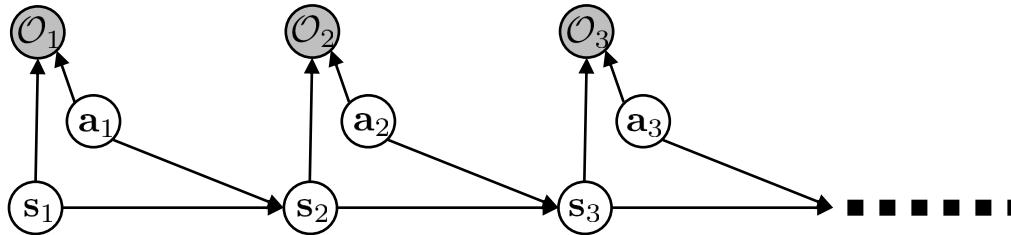
- 
1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]])$
 2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

soft value iteration algorithm:

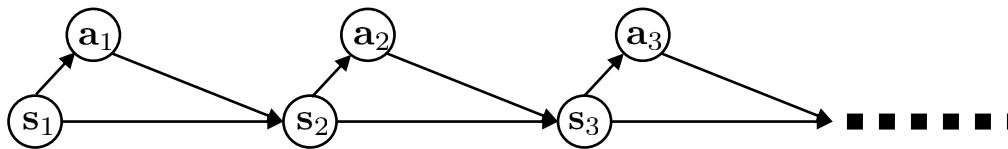
- 
1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]])$
 2. set $V(\mathbf{s}) \leftarrow \text{soft max}_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

Summary

$$p(\mathbf{s}_{1:T}, \mathbf{a}_{1:T} | \mathcal{O}_{1:T})$$



$$q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$$



$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t)) d\mathbf{a}_t \quad Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + E[V_{t+1}(\mathbf{s}_{t+1})]$$

variants:

discounted SOC: $Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma E[V_{t+1}(\mathbf{s}_{t+1})]$

explicit temperature: $V_t(\mathbf{s}_t) = \alpha \log \int \exp \left(\frac{1}{\alpha} Q_t(\mathbf{s}_t, \mathbf{a}_t) \right) d\mathbf{a}_t$

For more details, see: Levine. (2018). Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review.

Algorithms for RL as Inference

Q-learning with soft optimality

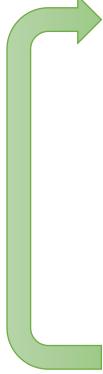
standard Q-learning: $\phi \leftarrow \phi + \alpha \nabla_\phi Q_\phi(\mathbf{s}, \mathbf{a})(r(\mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') - Q_\phi(\mathbf{s}, \mathbf{a}))$

target value: $V(\mathbf{s}') = \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$

soft Q-learning: $\phi \leftarrow \phi + \alpha \nabla_\phi Q_\phi(\mathbf{s}, \mathbf{a})(r(\mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') - Q_\phi(\mathbf{s}, \mathbf{a}))$

target value: $V(\mathbf{s}') = \text{soft max}_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}') = \log \int \exp(Q_\phi(\mathbf{s}', \mathbf{a}')) d\mathbf{a}'$

$\pi(\mathbf{a}|\mathbf{s}) = \exp(Q_\phi(\mathbf{s}, \mathbf{a}) - V(\mathbf{s})) = \exp(A(\mathbf{s}, \mathbf{a}))$

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{R}
 2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{R} uniformly
 3. compute $y_j = r_j + \gamma \text{soft max}_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
 5. update ϕ' : copy ϕ every N steps, or Polyak average $\phi' \leftarrow \tau\phi' + (1 - \tau)\phi$

Policy gradient with soft optimality

$\pi(\mathbf{a}|\mathbf{s}) = \exp(Q_\phi(\mathbf{s}, \mathbf{a}) - V(\mathbf{s}))$ optimizes $\sum_t E_{\pi(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] + E_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]$

policy entropy

intuition: $\pi(\mathbf{a}|\mathbf{s}) \propto \exp(Q_\phi(\mathbf{s}, \mathbf{a}))$ when π minimizes $D_{\text{KL}}(\pi(\mathbf{a}|\mathbf{s}) \| \frac{1}{Z} \exp(Q(\mathbf{s}, \mathbf{a})))$

$$D_{\text{KL}}(\pi(\mathbf{a}|\mathbf{s}) \| \frac{1}{Z} \exp(Q(\mathbf{s}, \mathbf{a}))) = E_{\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s}, \mathbf{a})] - \mathcal{H}(\pi)$$

often referred to as “entropy regularized” policy gradient

combats premature entropy collapse

turns out to be closely related to soft Q-learning:

see Haarnoja et al. ‘17 and Schulman et al. ‘17

Policy gradient vs Q-learning

policy gradient derivation:

$$J(\theta) = \sum_t E_{\pi(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] + E_{\pi(\mathbf{s}_t)}[\underbrace{\mathcal{H}(\pi(\mathbf{a}|\mathbf{s}_t))}_{t}] = \sum_t E_{\pi(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t)]$$

$$E_{\pi(\mathbf{a}_t|\mathbf{s}_t)}[-\log \pi(\mathbf{a}_t|\mathbf{s}_t)]$$

$$\nabla_\theta \left[\sum_t E_{\pi(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t) - \underbrace{\log \pi(\mathbf{a}_t|\mathbf{s}_t)}_{\text{can ignore (baseline)}}] \right]$$

$$\approx \frac{1}{N} \sum_i \sum_t \nabla_\theta \log \pi(\mathbf{a}_t|\mathbf{s}_t) \left(r(\mathbf{s}_t, \mathbf{a}_t) + \underbrace{\left(\sum_{t'=t+1}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \log \pi(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \right)}_{\text{can ignore (baseline)}} - \log \pi(\mathbf{a}_t|\mathbf{s}_t) - \underbrace{1}_{\text{can ignore (baseline)}} \right)$$

$$\text{recall: } \log \pi(\mathbf{a}_t|\mathbf{s}_t) = Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t) \quad \approx Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$$

$$\approx \frac{1}{N} \sum_i \sum_t (\underbrace{\nabla_\theta Q(\mathbf{a}_t|\mathbf{s}_t) - \nabla_\theta V(\mathbf{s}_t)}_{\text{Q-learning descent (vs ascent)}}) (r(\mathbf{s}_t, \mathbf{a}_t) + Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q(\mathbf{s}_t, \mathbf{a}_t) + \cancel{V(\mathbf{s}_t)})$$

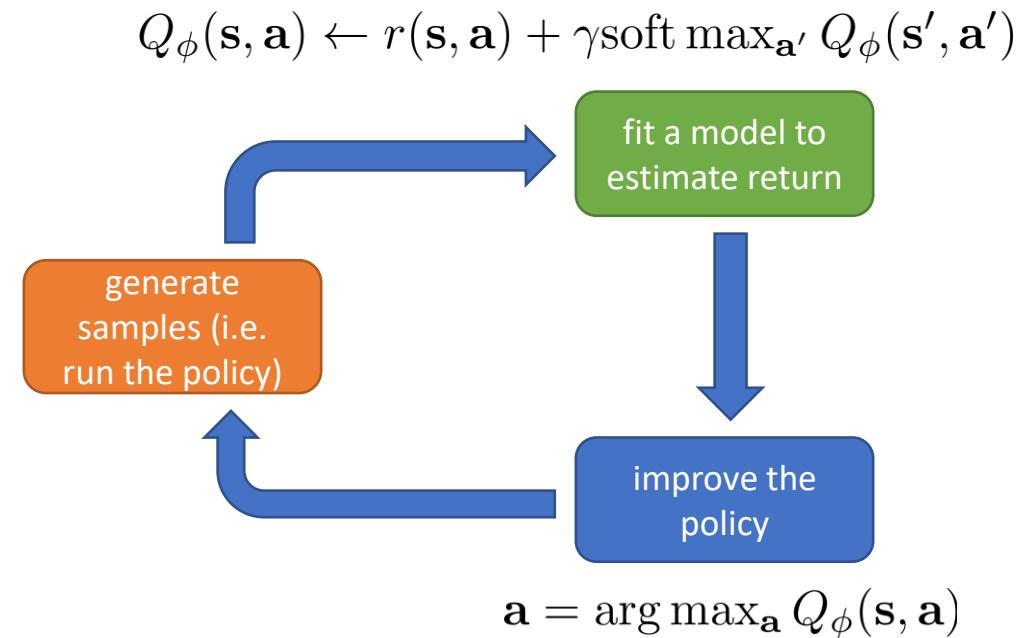
$$\text{Q-learning descent (vs ascent)} \quad \cancel{\frac{1}{N} \sum_i \sum_t \nabla_\theta Q(\mathbf{a}_t|\mathbf{s}_t) \left(r(\mathbf{s}_t, \mathbf{a}_t) + \underbrace{\text{soft max}_{\mathbf{a}_{t+1}} Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q(\mathbf{s}_t, \mathbf{a}_t)}_{\text{off-policy correction}} \right)}$$

Benefits of soft optimality

- Improve exploration and prevent entropy collapse
- Easier to specialize (finetune) policies for more specific tasks
- Principled approach to break ties
- Better robustness (due to wider coverage of states)
- Can reduce to hard optimality as reward magnitude increases
- Good model for modeling human behavior (more on this later)

Review

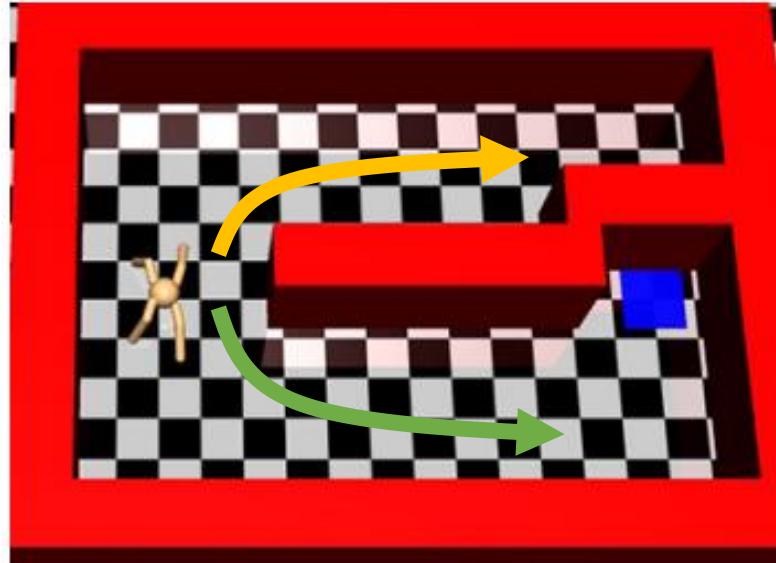
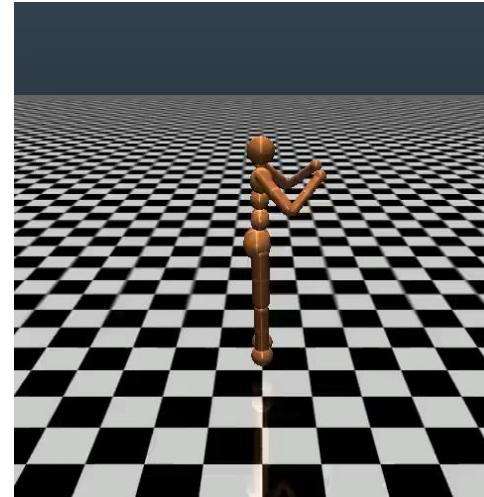
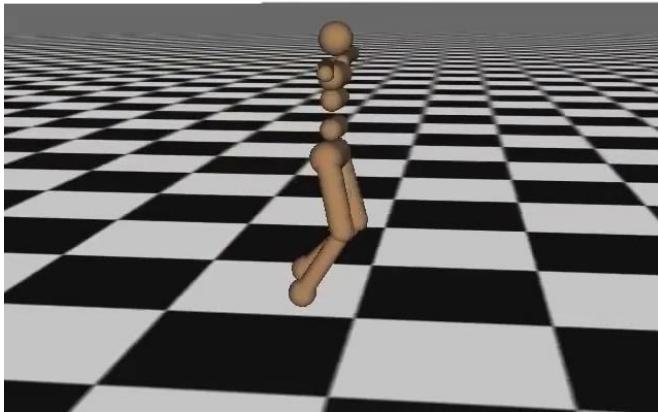
- Reinforcement learning can be viewed as inference in a graphical model
 - Value function is a backward message
 - Maximize reward and entropy (the bigger the rewards, the less entropy matters)
 - Variational inference to remove optimism
- Soft Q-learning
- Entropy-regularized policy gradient



Example Methods

Stochastic models for learning control

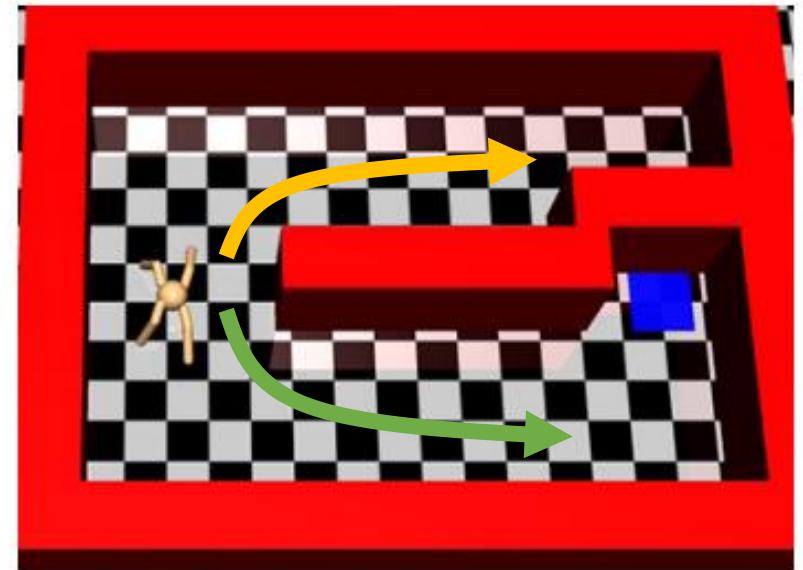
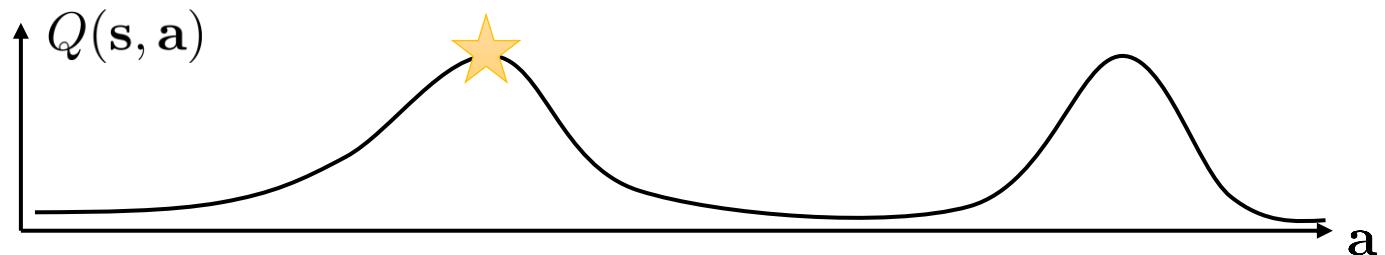
Iteration 2000



- How can we track *both* hypotheses?

Stochastic energy-based policies

Q-function: $Q(\mathbf{s}, \mathbf{a}) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$



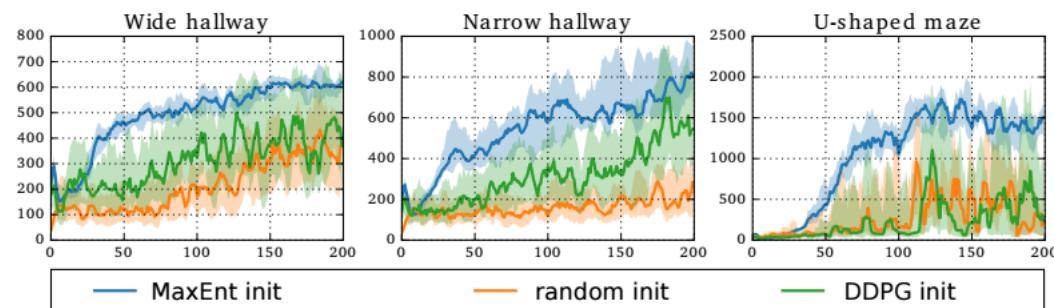
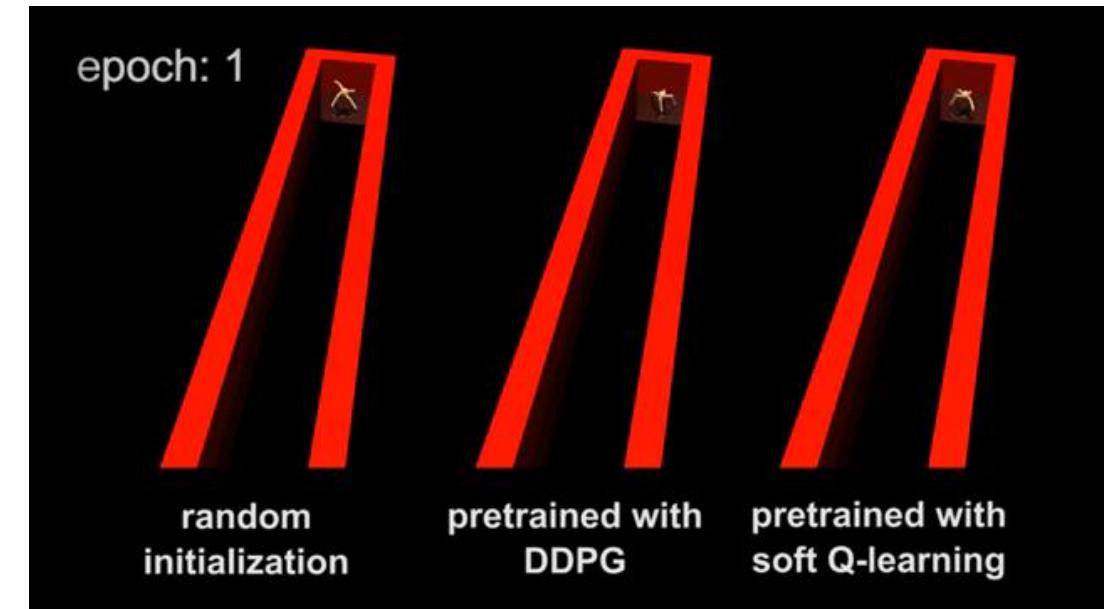
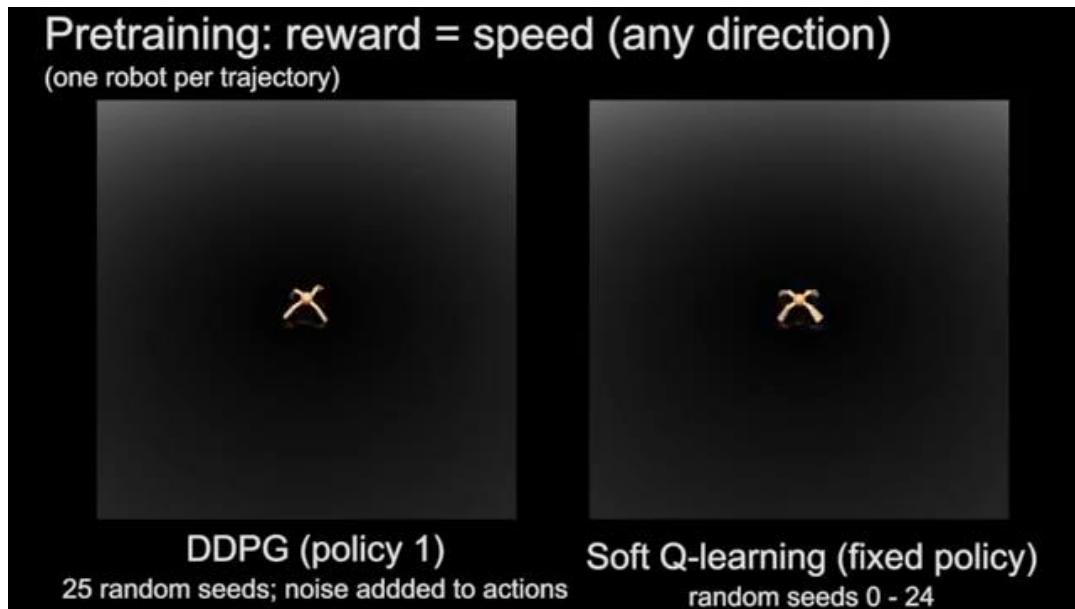
$$\pi(\mathbf{a}|\mathbf{s}) \propto \exp(Q(\mathbf{s}, \mathbf{a}))$$

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t) - V_t(\mathbf{s}_t)) = \exp(A_t(\mathbf{s}_t, \mathbf{a}_t))$$

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + E[V_{t+1}(\mathbf{s}_{t+1})]$$

$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t)) \mathbf{a}_t$$

Stochastic energy-based policies provide pretraining



Soft actor-critic

1. Q-function update

Update Q-function to evaluate current policy:

$$Q(s, a) \leftarrow r(s, a) + \mathbb{E}_{s' \sim p_s, a' \sim \pi} [Q(s', a') - \log \pi(a'|s')]$$

This converges to Q^π

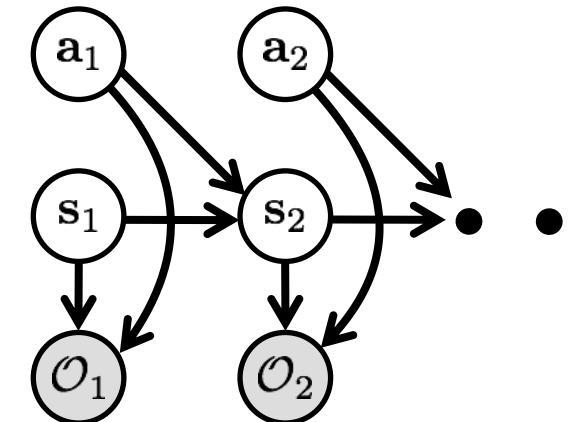
2. Update policy

Update the policy with gradient of information projection:

$$\pi_{\text{new}} = \arg \min_{\pi'} D_{\text{KL}} \left(\pi'(\cdot | s) \parallel \frac{1}{Z} \exp Q^{\pi_{\text{old}}}(s, \cdot) \right)$$

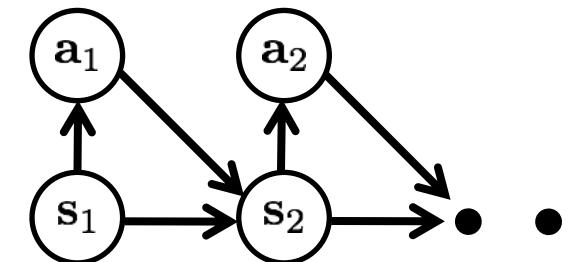
In practice, only take one gradient step on this objective

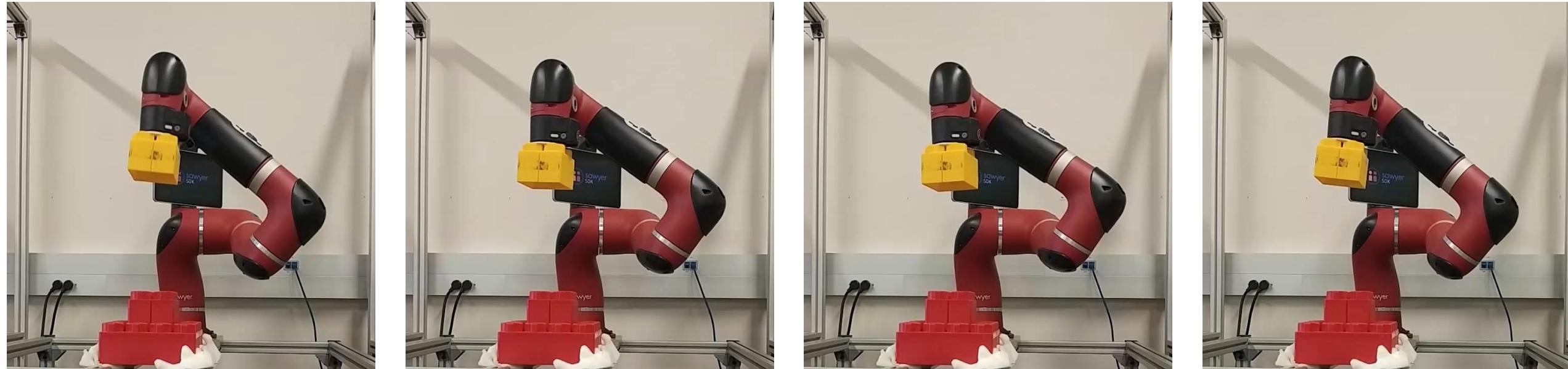
3. Interact with the world, collect more data



update messages

fit variational distribution





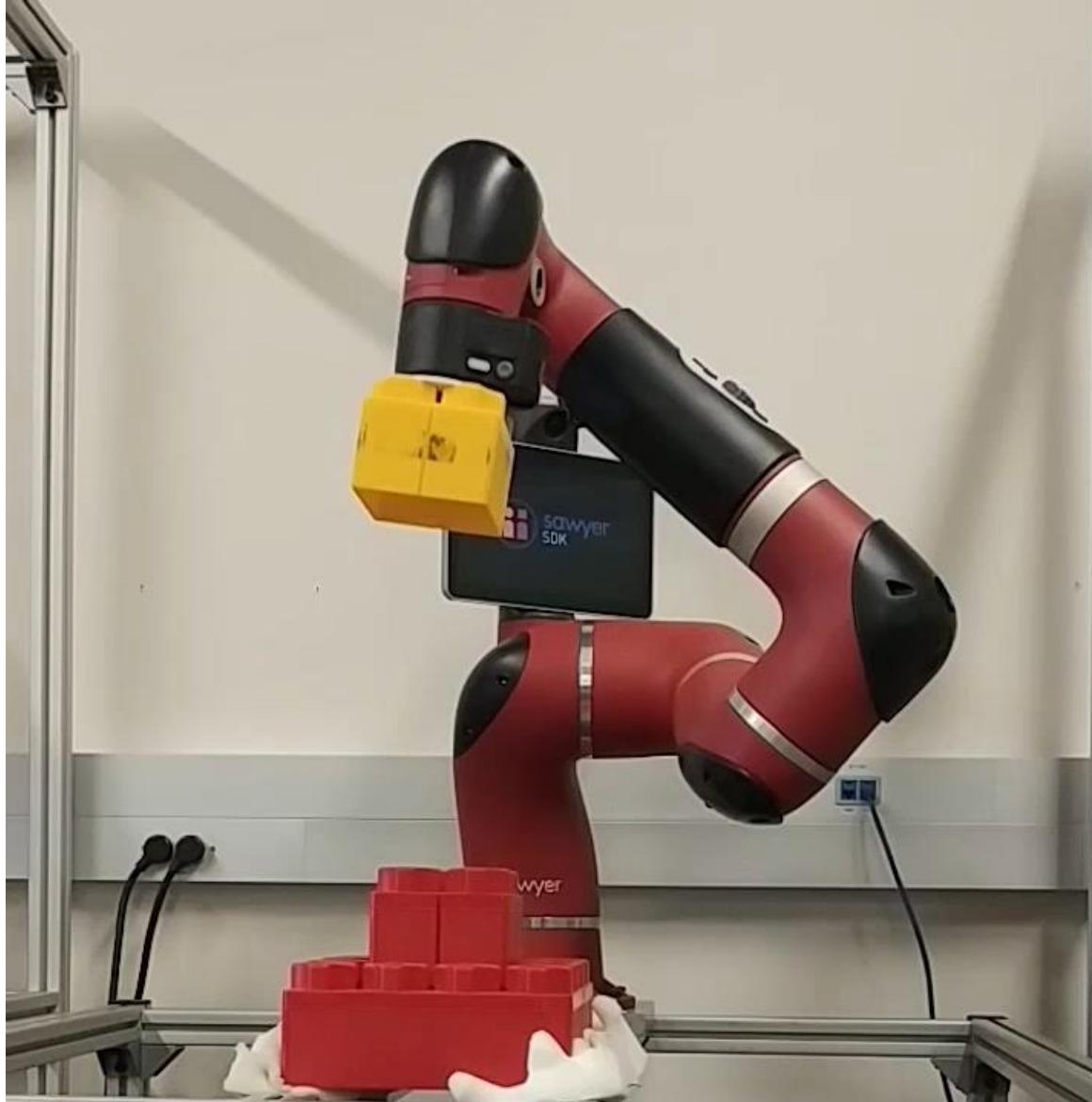
0 min

12 min

30 min

2 hours

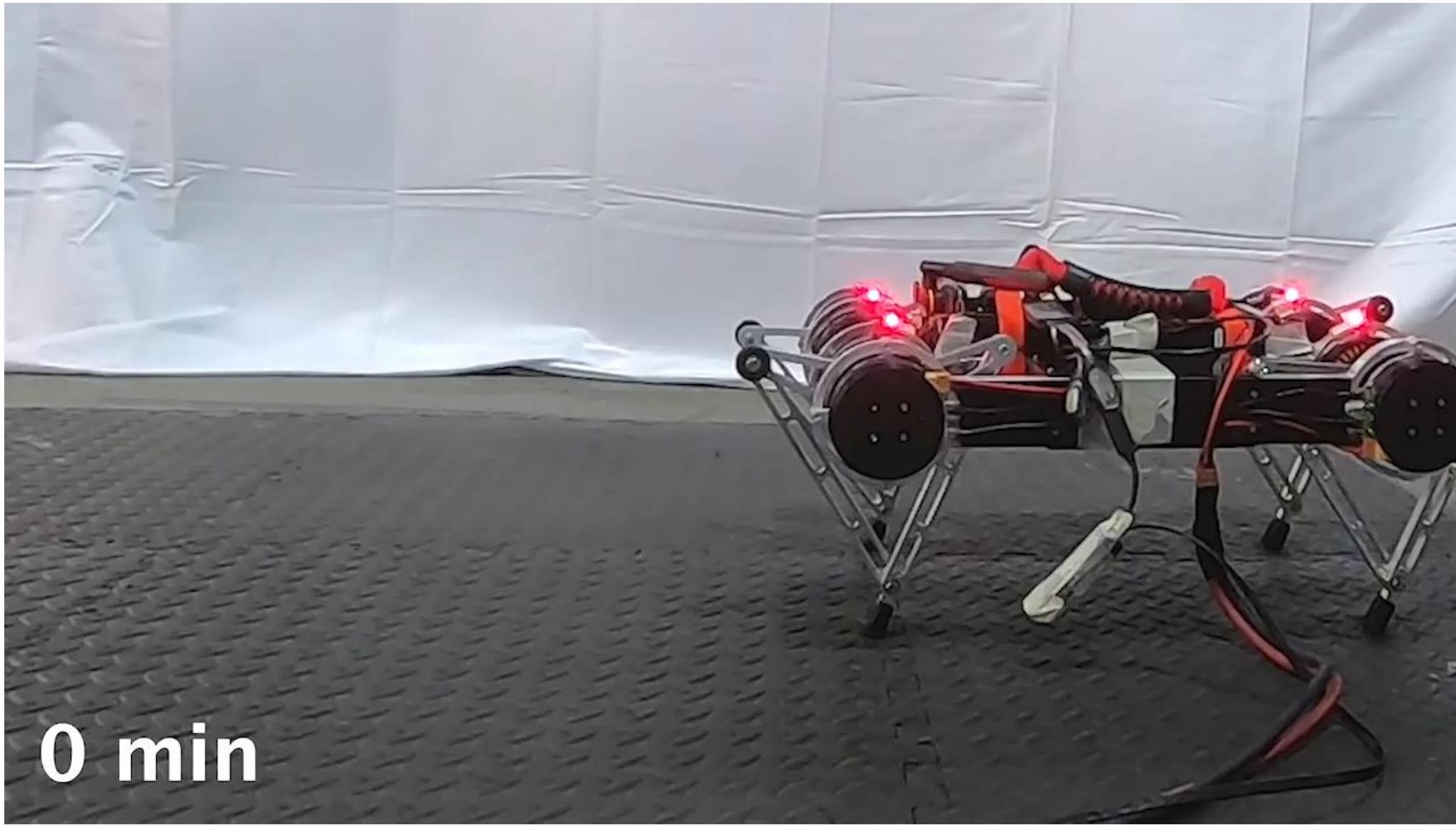
Training time



After 2 hours of training

sites.google.com/view/composing-real-world-policies/

Haarnoja, Pong, Zhou, Dalal, Abbeel, L. **Composable Deep Reinforcement Learning for Robotic Manipulation.** '18



0 min



Soft optimality suggested readings

- Todorov. (2006). Linearly solvable Markov decision problems: one framework for reasoning about soft optimality.
- Todorov. (2008). General duality between optimal control and estimation: primer on the equivalence between inference and control.
- Kappen. (2009). Optimal control as a graphical model inference problem: frames control as an inference problem in a graphical model.
- Ziebart. (2010). Modeling interaction via the principle of maximal causal entropy: connection between soft optimality and maximum entropy modeling.
- Rawlik, Toussaint, Vijaykumar. (2013). On stochastic optimal control and reinforcement learning by approximate inference: temporal difference style algorithm with soft optimality.
- Haarnoja*, Tang*, Abbeel, L. (2017). Reinforcement learning with deep energy based models: soft Q-learning algorithm, deep RL with continuous actions and soft optimality
- Nachum, Norouzi, Xu, Schuurmans. (2017). Bridging the gap between value and policy based reinforcement learning.
- Schulman, Abbeel, Chen. (2017). Equivalence between policy gradients and soft Q-learning.
- Haarnoja, Zhou, Abbeel, L. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.
- Levine. (2018). Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review

Inverse Reinforcement Learning

CS 285

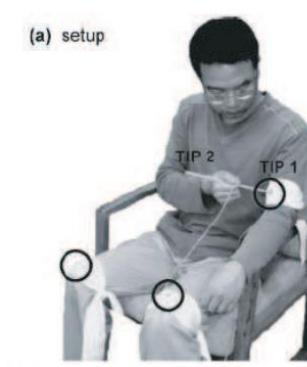
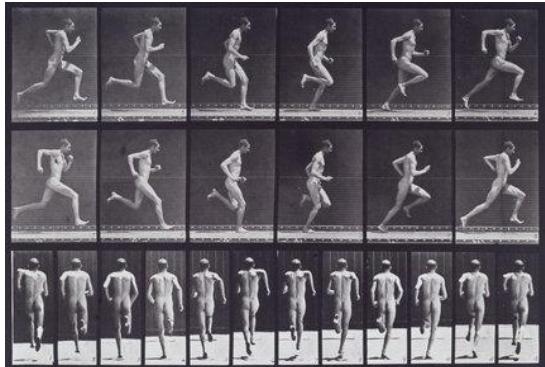
Instructor: Sergey Levine
UC Berkeley



Today's Lecture

1. So far: manually design reward function to define a task
 2. What if we want to *learn* the reward function from observing an expert, and then use reinforcement learning?
 3. Apply approximate optimality model from last time, but now learn the reward!
- Goals:
 - Understand the inverse reinforcement learning problem definition
 - Understand how probabilistic models of behavior can be used to derive inverse reinforcement learning algorithms
 - Understand a few practical inverse reinforcement learning algorithms we can use

Optimal Control as a Model of Human Behavior



$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg \max_{\pi} E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t), \mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

$$\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)$$

optimize this to explain the data

Why should we worry about learning rewards?

The imitation learning perspective

Standard imitation learning:

- copy the *actions* performed by the expert
- no reasoning about outcomes of actions



Human imitation learning:

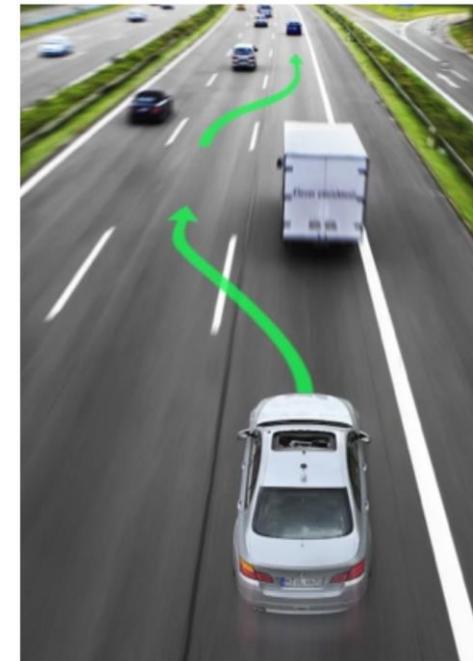
- copy the *intent* of the expert
- might take very different actions!



© Warneken & Tomasello

Why should we worry about learning rewards?

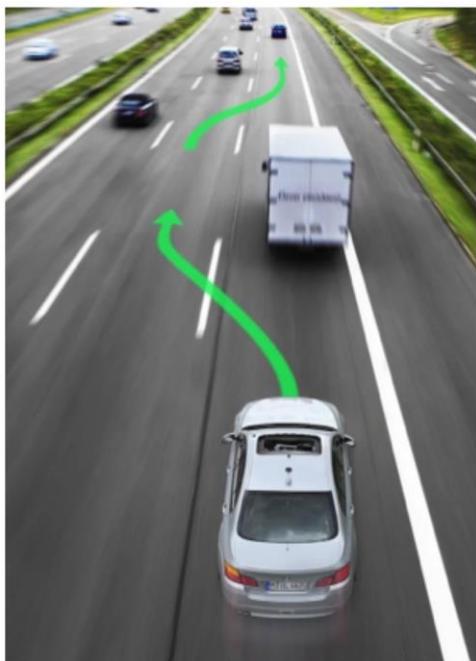
The reinforcement learning perspective



what is the reward?

Inverse reinforcement learning

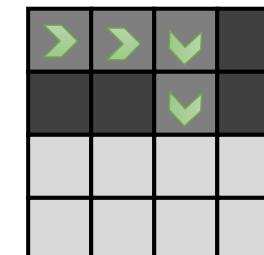
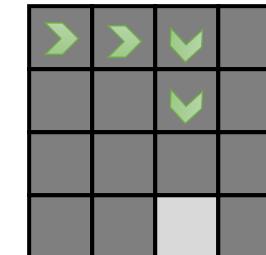
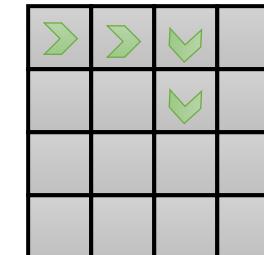
Infer reward functions from demonstrations



$$r(s, a)$$

by itself, this is an **underspecified** problem

many reward functions can explain the **same** behavior



A bit more formally

"forward" reinforcement learning

given:

states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

(sometimes) transitions $p(s'|s, a)$

reward function $r(s, a)$

learn $\pi^*(a|s)$

inverse reinforcement learning

given:

states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

(sometimes) transitions $p(s'|s, a)$

samples $\{\tau_i\}$ sampled from $\pi^*(\tau)$

learn $r_\psi(s, a)$

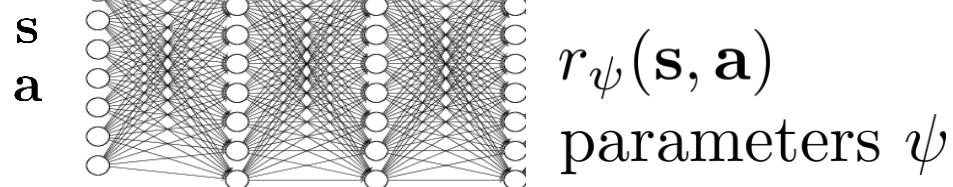
reward parameters

...and then use it to learn $\pi^*(a|s)$

neural net reward function:

linear reward function:

$$r_\psi(s, a) = \sum_i \psi_i f_i(s, a) = \psi^T \mathbf{f}(s, a)$$



Feature matching IRL

linear reward function:

$$r_\psi(\mathbf{s}, \mathbf{a}) = \sum_i \psi_i f_i(\mathbf{s}, \mathbf{a}) = \psi^T \mathbf{f}(\mathbf{s}, \mathbf{a})$$

if features \mathbf{f} are important, what if we match their expectations?

let π^{r_ψ} be the optimal policy for r_ψ

pick ψ such that $E_{\pi^{r_\psi}}[\mathbf{f}(\mathbf{s}, \mathbf{a})] = E_{\pi^*}[\mathbf{f}(\mathbf{s}, \mathbf{a})]$

state-action marginal under π^{r_ψ}

unknown optimal policy
approximate using expert samples

still ambiguous!

maximum margin principle:

$$\max_{\psi, m} \text{such that } \psi^T E_{\pi^*}[\mathbf{f}(\mathbf{s}, \mathbf{a})] \geq \max_{\pi \in \Pi} \psi^T E_\pi[\mathbf{f}(\mathbf{s}, \mathbf{a})] + m$$



need to somehow “weight” by similarity between π^* and π

Feature matching IRL & maximum margin

remember the “SVM trick”:

$$\max_{\psi, m} m \quad \text{such that } \psi^T E_{\pi^*}[\mathbf{f}(s, a)] \geq \max_{\pi \in \Pi} \psi^T E_\pi[\mathbf{f}(s, a)] + m$$



$$\min_{\psi} \frac{1}{2} \|\psi\|^2 \quad \text{such that } \psi^T E_{\pi^*}[\mathbf{f}(s, a)] \geq \max_{\pi \in \Pi} \psi^T E_\pi[\mathbf{f}(s, a)] + D(\pi, \pi^*)$$

e.g., difference in feature expectations!

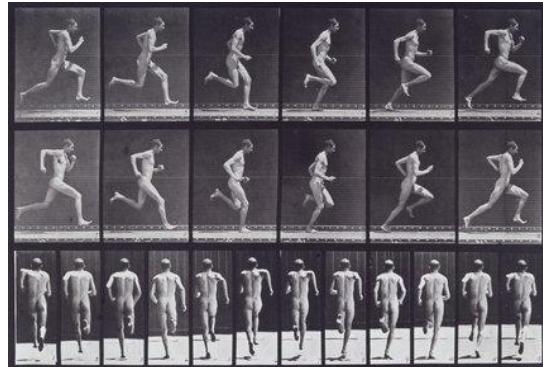
Issues:

- Maximizing the margin is a bit arbitrary
- No clear model of expert suboptimality (can add slack variables...)
- Messy constrained optimization problem – not great for deep learning!

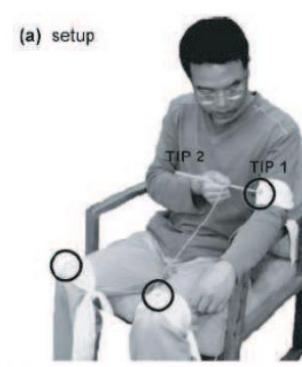
Further reading:

- Abbeel & Ng: Apprenticeship learning via inverse reinforcement learning
- Ratliff et al: Maximum margin planning

Optimal Control as a Model of Human Behavior



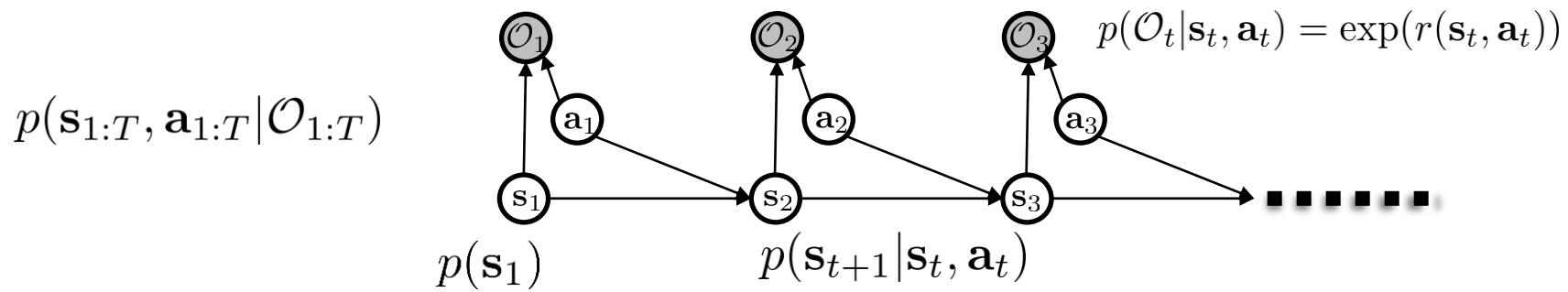
Mombaur et al. '09



Li & Todorov '06



Ziebart '08

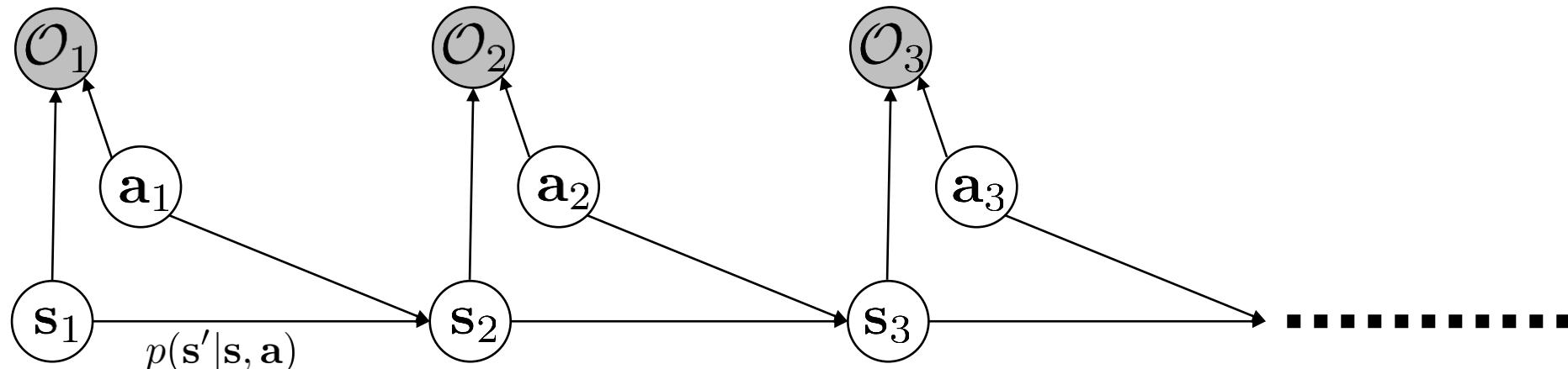


A probabilistic graphical model of decision making

$$p(\underbrace{\mathbf{s}_{1:T}, \mathbf{a}_{1:T}}_{\tau}) = ?? \quad \text{no assumption of optimal behavior!}$$

$$p(\tau | \mathcal{O}_{1:T}) \qquad \qquad p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t))$$

$$\begin{aligned} p(\tau | \mathcal{O}_{1:T}) &= \frac{p(\tau, \mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})} \\ &\propto p(\tau) \prod_t \exp(r(\mathbf{s}_t, \mathbf{a}_t)) = p(\tau) \exp \left(\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right) \end{aligned}$$



Learning the Reward Function

Learning the optimality variable

$$p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t, \psi) \propto \exp(r_\psi(\mathbf{s}_t, \mathbf{a}_t))$$

reward parameters

given:

samples $\{\tau_i\}$ sampled from $\pi^*(\tau)$

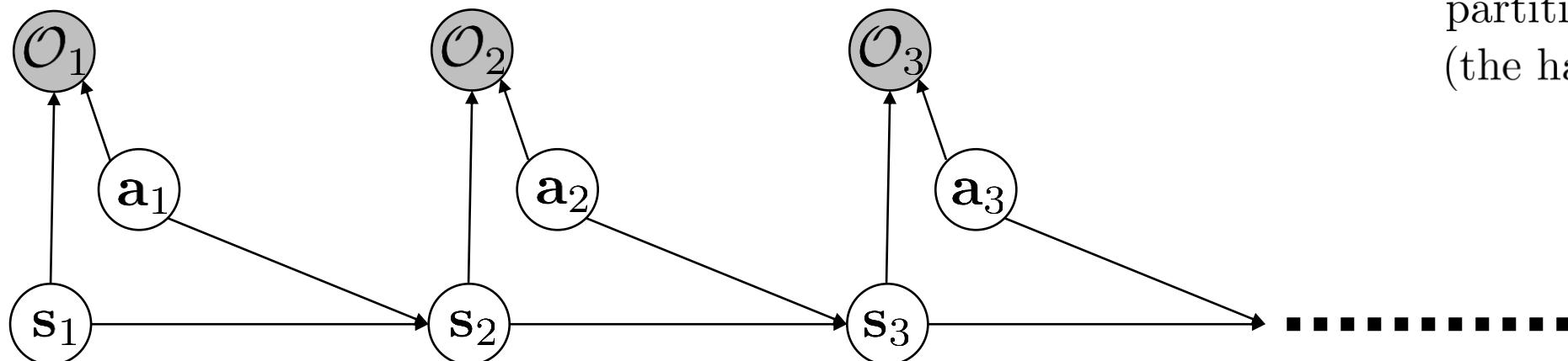
$$p(\tau | \mathcal{O}_{1:T}, \psi) \propto p(\tau) \exp \left(\sum_t r_\psi(\mathbf{s}_t, \mathbf{a}_t) \right)$$

can ignore (independent of ψ)

maximum likelihood learning:

$$\max_{\psi} \frac{1}{N} \sum_{i=1}^N \log p(\tau_i | \mathcal{O}_{1:T}, \psi) = \max_{\psi} \frac{1}{N} \sum_{i=1}^N r_\psi(\tau_i) - \log Z$$

partition function
(the hard part)



The IRL partition function

$$\max_{\psi} \frac{1}{N} \sum_{i=1}^N r_{\psi}(\tau_i) - \log Z \quad Z = \int p(\tau) \exp(r_{\psi}(\tau)) d\tau$$

$$\nabla_{\psi} \mathcal{L} = \frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \underbrace{\frac{1}{Z} \int p(\tau) \exp(r_{\psi}(\tau)) \nabla_{\psi} r_{\psi}(\tau) d\tau}_{p(\tau | \mathcal{O}_{1:T}, \psi)}$$

$$\nabla_{\psi} \mathcal{L} = E_{\tau \sim \pi^*(\tau)} [\nabla_{\psi} r_{\psi}(\tau_i)] - E_{\tau \sim p(\tau | \mathcal{O}_{1:T}, \psi)} [\nabla_{\psi} r_{\psi}(\tau)]$$

estimate with expert samples

soft optimal policy under current reward

Estimating the expectation

$$\nabla_{\psi} \mathcal{L} = E_{\tau \sim \pi^*(\tau)} [\nabla_{\psi} r_{\psi}(\tau_i)] - E_{\tau \sim p(\tau | \mathcal{O}_{1:T}, \psi)} [\nabla_{\psi} r_{\psi}(\tau)]$$



$$E_{\tau \sim p(\tau | \mathcal{O}_{1:T}, \psi)} \left[\nabla_{\psi} \sum_{t=1}^T r_{\psi}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$= \sum_{t=1}^T E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p(\mathbf{s}_t, \mathbf{a}_t | \mathcal{O}_{1:T}, \psi)} [\nabla_{\psi} r_{\psi}(\mathbf{s}_t, \mathbf{a}_t)]$$

$$p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T}, \psi) p(\mathbf{s}_t | \mathcal{O}_{1:T}, \psi)$$

where have we seen this before?

$$= \frac{\beta(\mathbf{s}_t, \mathbf{a}_t)}{\beta(\mathbf{s}_t)} \qquad \qquad \propto \alpha(\mathbf{s}_t) \beta(\mathbf{s}_t)$$

$$p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T}, \psi) p(\mathbf{s}_t | \mathcal{O}_{1:T}, \psi) \propto \beta(\mathbf{s}_t, \mathbf{a}_t) \alpha(\mathbf{s}_t)$$

backward message

↑

↑

forward message

Estimating the expectation

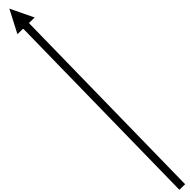
$$\nabla_{\psi} \mathcal{L} = E_{\tau \sim \pi^*(\tau)} [\nabla_{\psi} r_{\psi}(\tau_i)] - E_{\tau \sim p(\tau | \mathcal{O}_{1:T}, \psi)} [\nabla_{\psi} r_{\psi}(\tau)]$$

let $\mu_t(\mathbf{s}_t, \mathbf{a}_t) \propto \beta(\mathbf{s}_t, \mathbf{a}_t) \alpha(\mathbf{s}_t)$



$$\sum_{t=1}^T \int \int \mu_t(\mathbf{s}_t, \mathbf{a}_t) \nabla_{\psi} r_{\psi}(\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_t d\mathbf{a}_t$$

$$= \sum_{t=1}^T \vec{\mu}_t^T \nabla_{\psi} \vec{r}_{\psi}$$



state-action visitation probability for each $(\mathbf{s}_t, \mathbf{a}_t)$

The MaxEnt IRL algorithm

- 
1. Given ψ , compute backward message $\beta(\mathbf{s}_t, \mathbf{a}_t)$ (see previous lecture)
 2. Given ψ , compute forward message $\alpha(\mathbf{s}_t)$ (see previous lecture)
 3. Compute $\mu_t(\mathbf{s}_t, \mathbf{a}_t) \propto \beta(\mathbf{s}_t, \mathbf{a}_t)\alpha(\mathbf{s}_t)$
 4. Evaluate $\nabla_\psi \mathcal{L} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\psi r_\psi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - \sum_{t=1}^T \int \int \mu_t(\mathbf{s}_t, \mathbf{a}_t) \nabla_\psi r_\psi(\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_t d\mathbf{a}_t$
 5. $\psi \leftarrow \psi + \eta \nabla_\psi \mathcal{L}$

Why MaxEnt?

in the case where $r_\psi(\mathbf{s}_t, \mathbf{a}_t) = \psi^T \mathbf{f}(\mathbf{s}_t, \mathbf{a}_t)$, we can show that it optimizes

$$\max_{\psi} \mathcal{H}(\pi^{r_\psi}) \text{ such that } E_{\pi^{r_\psi}}[\mathbf{f}] = E_{\pi^*}[\mathbf{f}]$$

optimal max-ent policy under r^ψ

as random as possible
while matching features

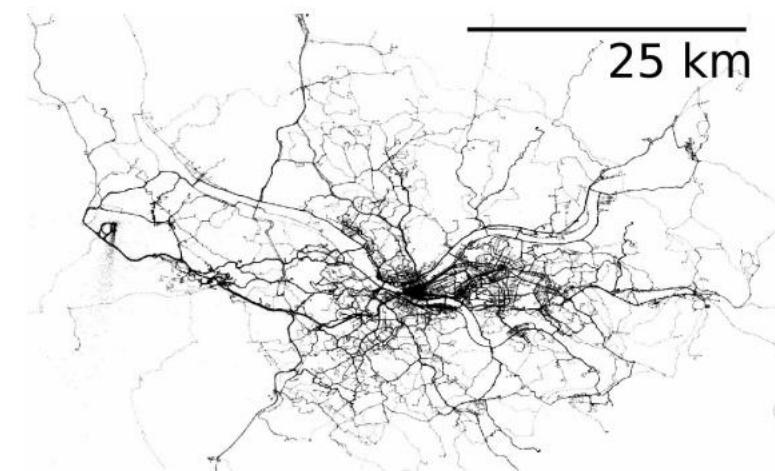
unknown expert policy
estimated with samples

Maximum Entropy Inverse Reinforcement Learning

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

bziebart@cs.cmu.edu, amaas@andrew.cmu.edu, dbagnell@ri.cmu.edu, anind@cs.cmu.edu



Feature	Value
Highway	3.3 miles
Major Streets	2.0 miles
Local Streets	0.3 miles
Above 55mph	4.0 miles
35-54mph	1.1 miles
25-34 mph	0.5 miles
Below 24mph	0 miles
3+ Lanes	0.5 miles
2 Lanes	3.3 miles
1 Lane	1.8 miles

Feature	Value
Hard left turn	1
Soft left turn	3
Soft right turn	5
Hard right turn	0
No turn	25
U-turn	0

Approximations in High Dimensions

What's missing so far?

- MaxEnt IRL so far requires...
 - Solving for (soft) optimal policy in the inner loop
 - Enumerating all state-action tuples for visitation frequency and gradient
- To apply this in practical problem settings, we need to handle...
 - Large and continuous state and action spaces
 - States obtained via sampling only
 - Unknown dynamics

Unknown dynamics & large state/action spaces

Assume we don't know the dynamics, but we can sample, like in standard RL

recall:

$$\nabla_{\psi} \mathcal{L} = E_{\tau \sim \pi^*(\tau)} [\nabla_{\psi} r_{\psi}(\tau_i)] - E_{\tau \sim p(\tau | \mathcal{O}_{1:T}, \psi)} [\nabla_{\psi} r_{\psi}(\tau)]$$



estimate with expert samples



soft optimal policy under current reward

idea: learn $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T}, \psi)$ using any max-ent RL algorithm

then run this policy to sample $\{\tau_j\}$

$$J(\theta) = \sum_t E_{\pi(\mathbf{s}_t, \mathbf{a}_t)} [r_{\psi}(\mathbf{s}_t, \mathbf{a}_t)] + E_{\pi(\mathbf{s}_t)} [\mathcal{H}(\pi(\mathbf{a} | \mathbf{s}_t))]$$

$$\nabla_{\psi} \mathcal{L} \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{M} \sum_{j=1}^M \nabla_{\psi} r_{\psi}(\tau_j)$$



sum over expert samples



sum over policy samples

More efficient sample-based updates

$$\nabla_{\psi} \mathcal{L} \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{M} \sum_{j=1}^M \nabla_{\psi} r_{\psi}(\tau_j)$$

↑ ↑
sum over expert samples sum over policy samples

improve ~~learn~~ $p(\mathbf{a}_t | \mathbf{s}_t, \mathcal{O}_{1:T}, \psi)$ using any max-ent RL algorithm
(a little)
then run this policy to sample $\{\tau_j\}$

looks expensive! what if we use “lazy” policy optimization?

problem: estimator is now biased! wrong distribution!

solution 1: use importance sampling

$$\nabla_{\psi} \mathcal{L} \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{\sum_j w_j} \sum_{j=1}^M w_j \nabla_{\psi} r_{\psi}(\tau_j) \quad w_j = \frac{p(\tau) \exp(r_{\psi}(\tau_j))}{\pi(\tau_j)}$$

Importance sampling

$$\nabla_{\psi} \mathcal{L} \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\psi} r_{\psi}(\tau_i) - \frac{1}{\sum_j w_j} \sum_{j=1}^M w_j \nabla_{\psi} r_{\psi}(\tau_j)$$
$$w_j = \frac{p(\tau) \exp(r_{\psi}(\tau_j))}{\pi(\tau_j)}$$

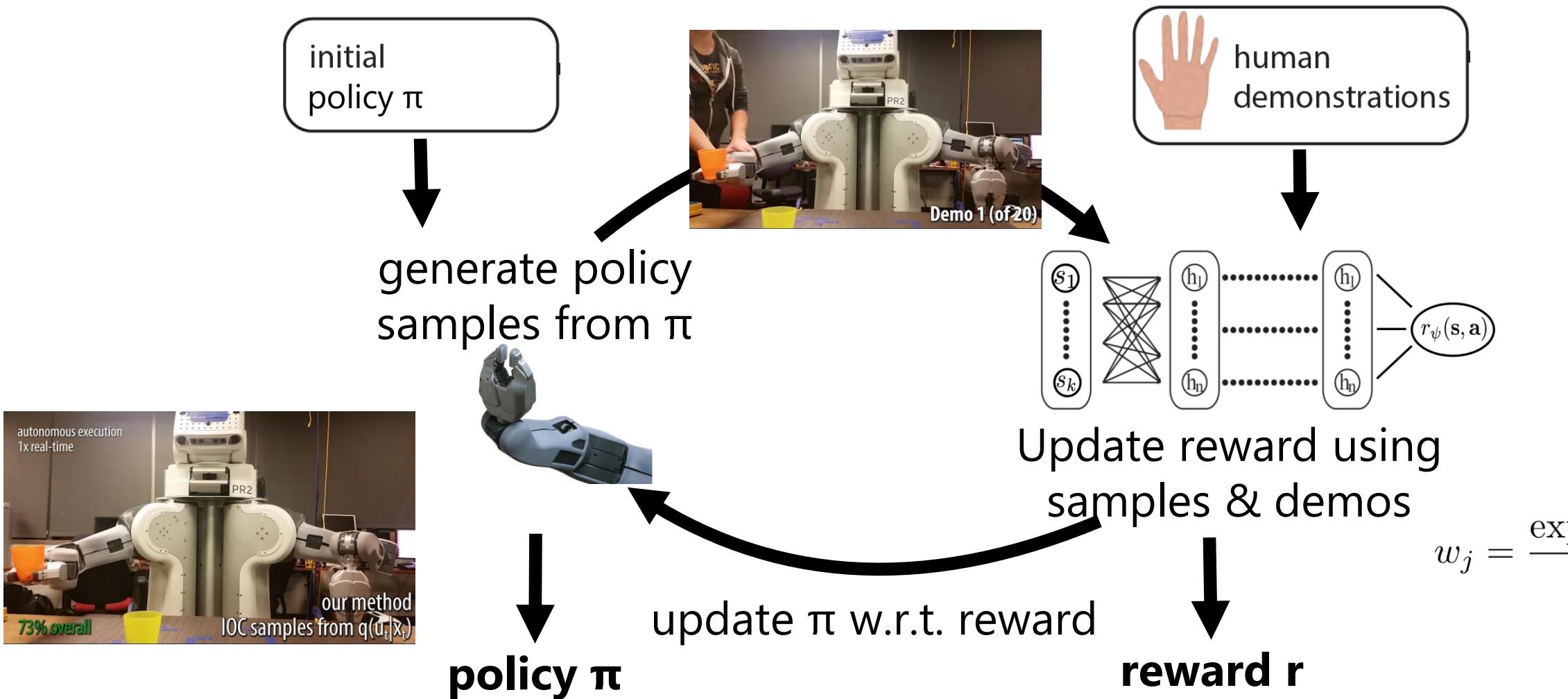


$$= \frac{\frac{p(s_1) \prod_t p(s_{t+1}|s_t, a_t) \exp(r_{\psi}(s_t, a_t))}{p(s_1) \prod_t p(s_{t+1}|s_t, a_t) \pi(a_t|s_t)}}{\prod_t \pi(a_t|s_t)}$$

each policy update w.r.t. r_{ψ} brings us closer to the target distribution!

guided cost learning algorithm

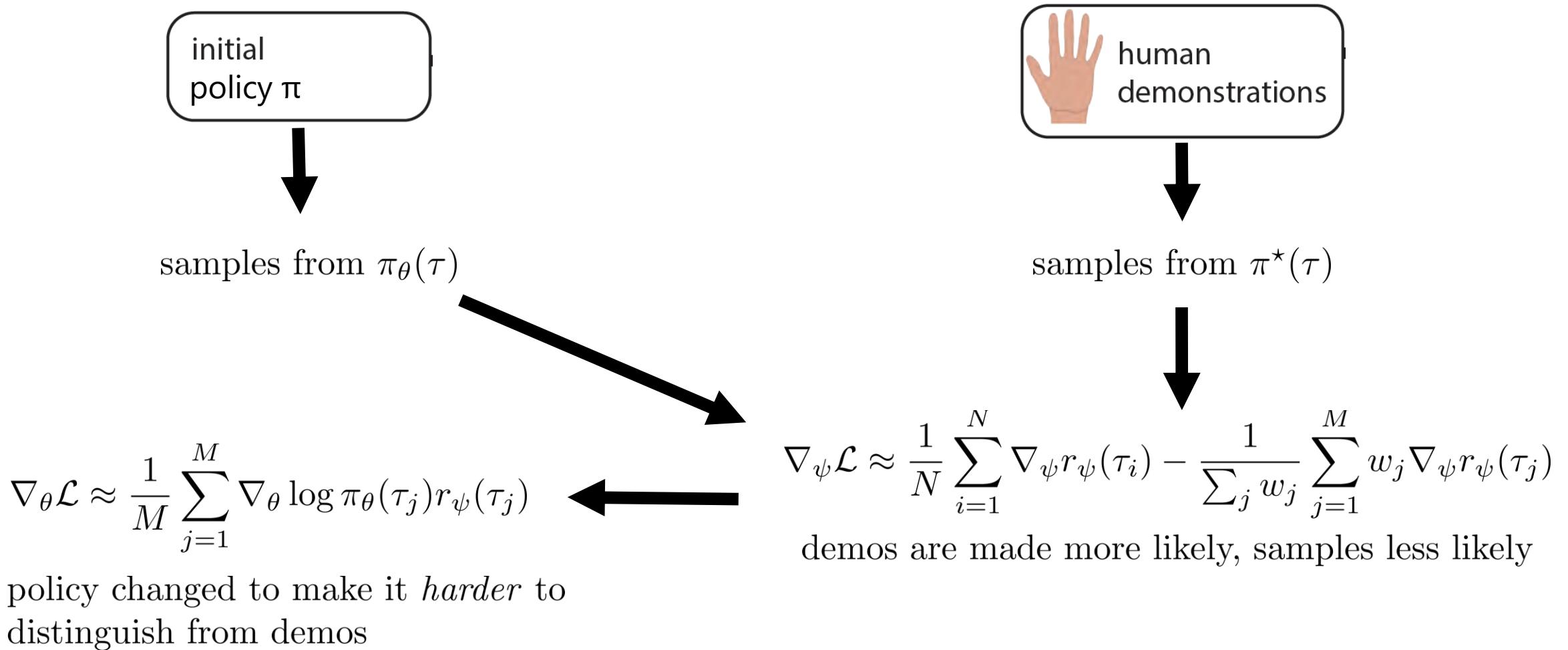
(Finn et al. ICML '16)



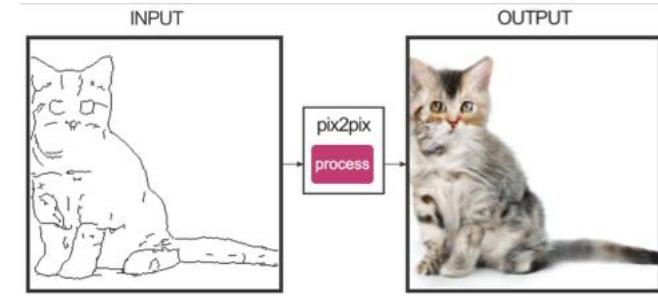
$$\nabla_\psi \mathcal{L} \approx \frac{1}{N} \sum_{i=1}^N \nabla_\psi r_\psi(\tau_i) - \frac{1}{\sum_j w_j} \sum_{j=1}^M w_j \nabla_\psi r_\psi(\tau_j)$$

IRL and GANs

It looks a bit like a game...



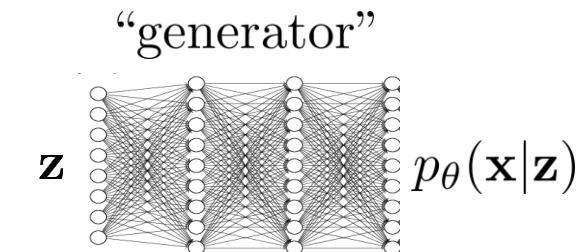
Generative Adversarial Networks



Zhu et al. '17

Arjovsky et al. '17

Isola et al. '17



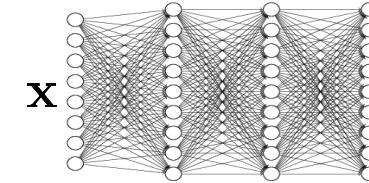
samples from $p_{\theta}(\mathbf{x})$

data (“demonstrations”)



$$D(\mathbf{x}) = p_{\psi}(\text{real image}|\mathbf{x})$$

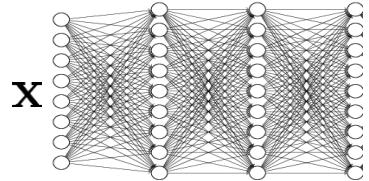
samples from $p^*(\mathbf{x})$



$$\begin{aligned} \psi &= \arg \max_{\psi} \frac{1}{N} \sum_{\mathbf{x} \sim p^*} \log D_{\psi}(\mathbf{x}) + \frac{1}{M} \sum_{\mathbf{x} \sim p_{\theta}} \log(1 - D_{\psi}(\mathbf{x})) \\ \theta &\leftarrow \arg \max_{\theta} E_{\mathbf{x} \sim p_{\theta}} \log D_{\psi}(\mathbf{x}) \end{aligned}$$

Inverse RL as a GAN

$$D(\mathbf{x}) = p_\psi(\text{real image}|\mathbf{x})$$



which discriminator is best?

$$D^*(\mathbf{x}) = \frac{p^*(\mathbf{x})}{p_\theta(\mathbf{x}) + p^*(\mathbf{x})}$$

for IRL, optimal policy approaches $\pi_\theta(\tau) \propto p(\tau) \exp(r_\psi(\tau))$

choose this parameterization for discriminator:

optimize Z w.r.t. same objective as ψ !

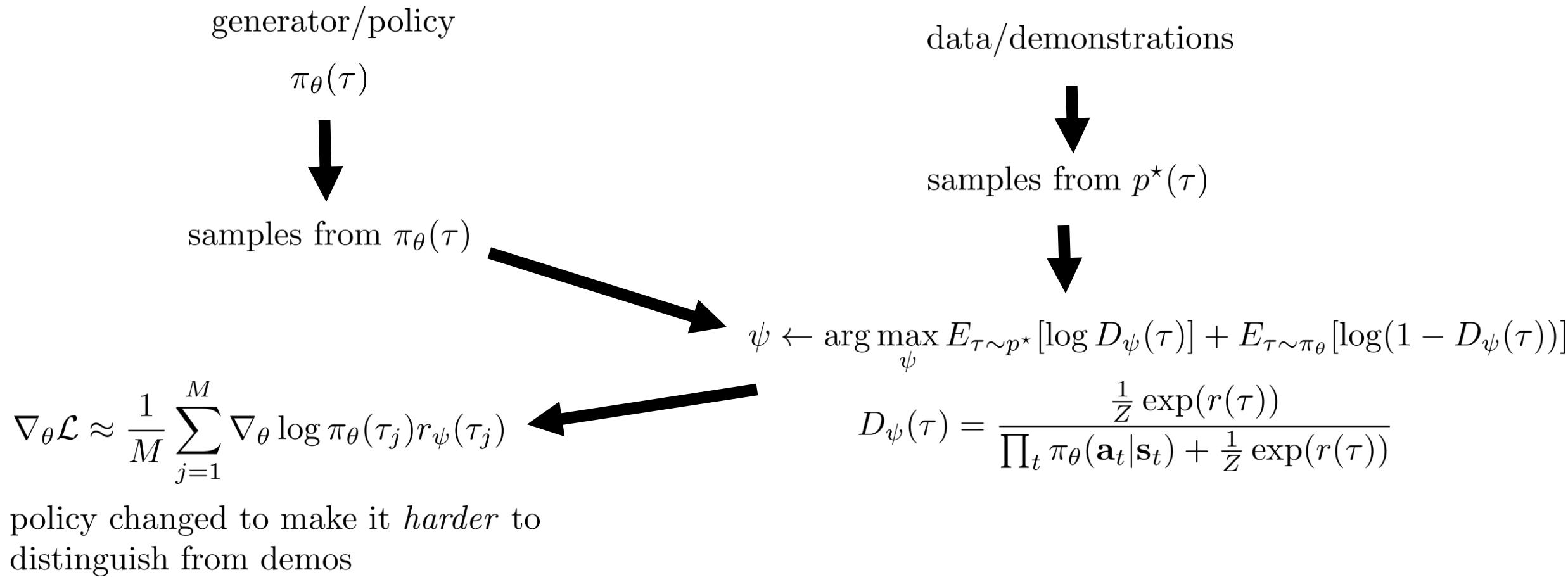
$$D_\psi(\tau) = \frac{p(\tau) \frac{1}{Z} \exp(r(\tau))}{p_\theta(\tau) + p(\tau) \frac{1}{Z} \exp(r(\tau))} = \frac{\cancel{p(\tau)} \frac{1}{Z} \exp(r(\tau))}{\cancel{p(\tau)} \prod_t \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{p(\tau)} \frac{1}{Z} \exp(r(\tau))} = \frac{\frac{1}{Z} \exp(r(\tau))}{\prod_t \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \frac{1}{Z} \exp(r(\tau))}$$

optimize this w.r.t. ψ

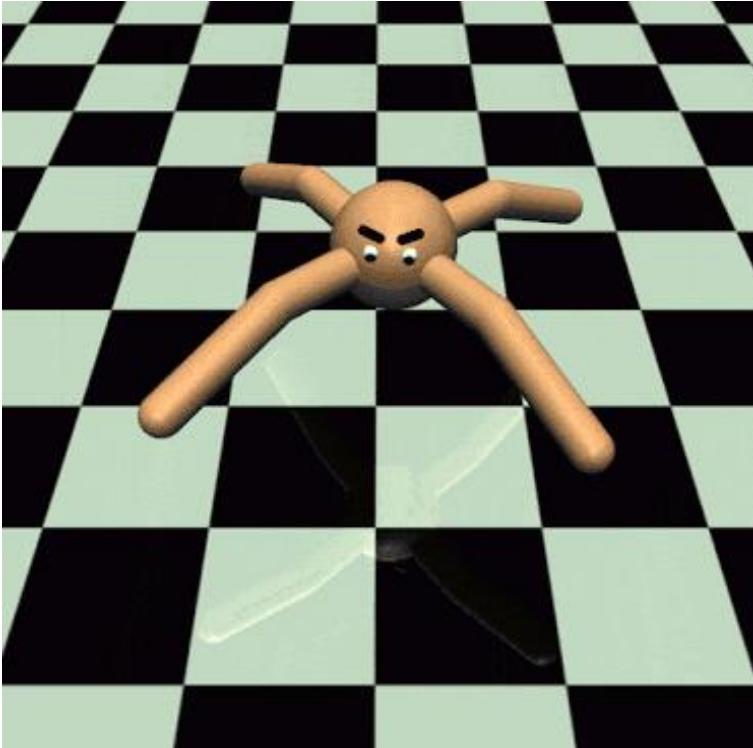
$$\psi \leftarrow \arg \max_{\psi} E_{\tau \sim p^*} [\log D_\psi(\tau)] + E_{\tau \sim \pi_\theta} [\log(1 - D_\psi(\tau))]$$

we don't need importance weights anymore – they are subsumed into Z

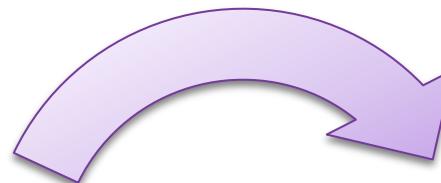
Inverse RL as a GAN



Generalization via inverse RL



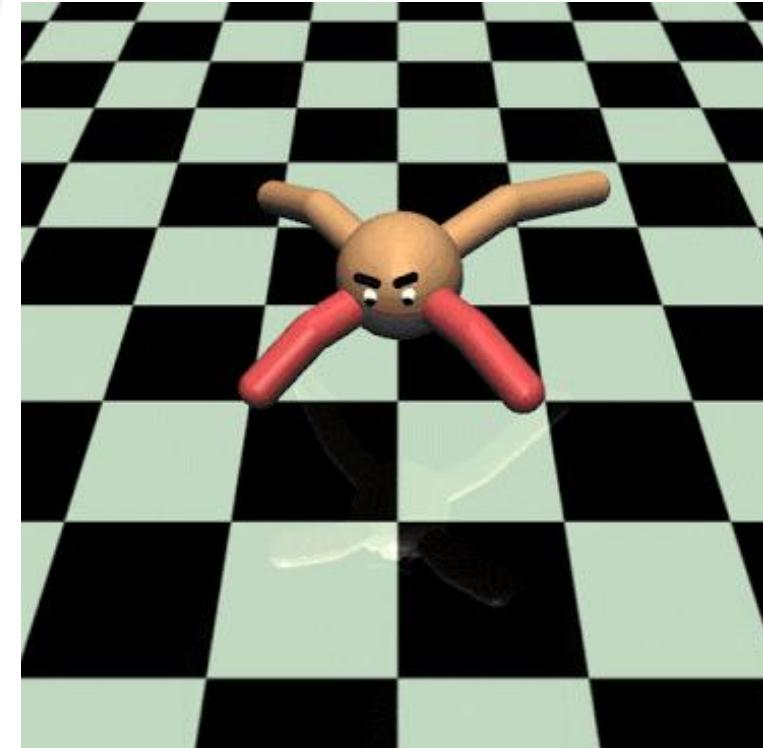
demonstration



what can we
learn from the
demonstration
to enable
better **transfer**?

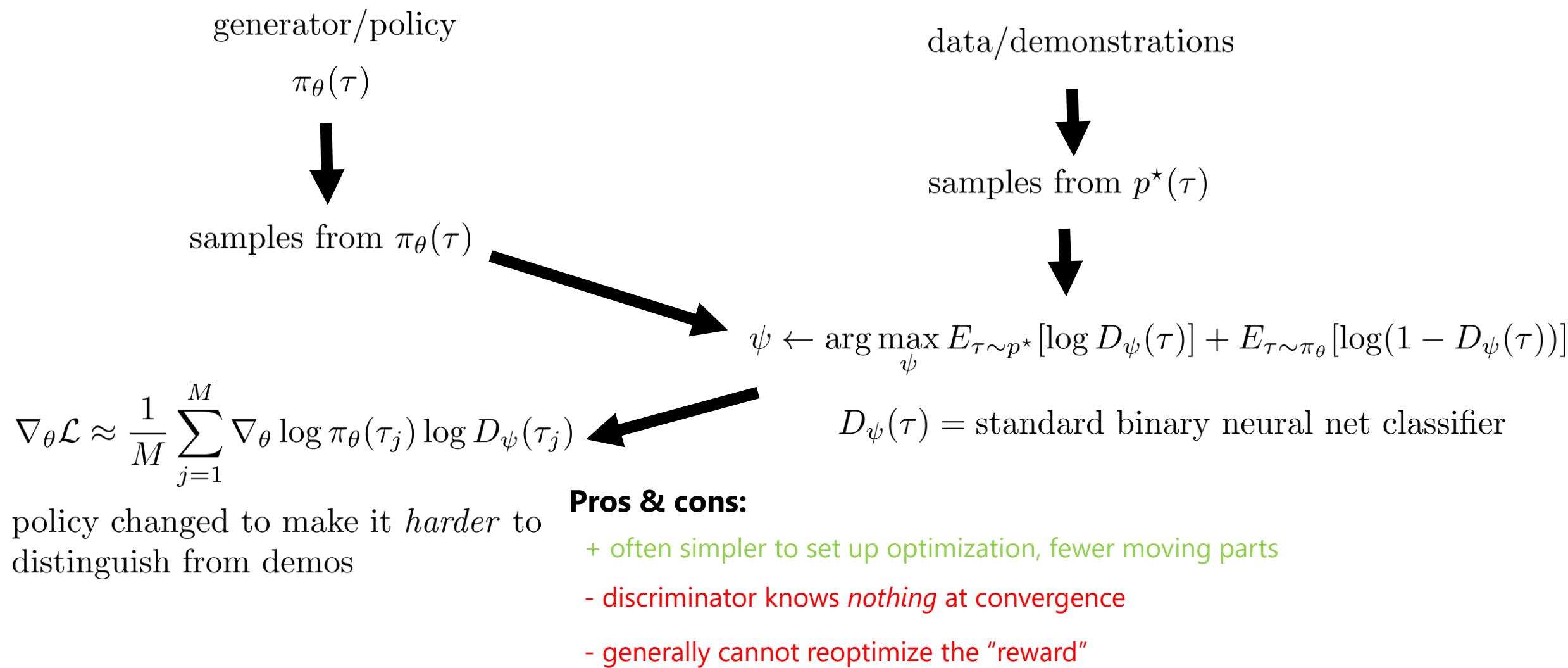
need to
decouple the
goal from the
dynamics!

policy =
reward +
dynamics



reproduce behavior under different conditions

Can we just use a regular discriminator?

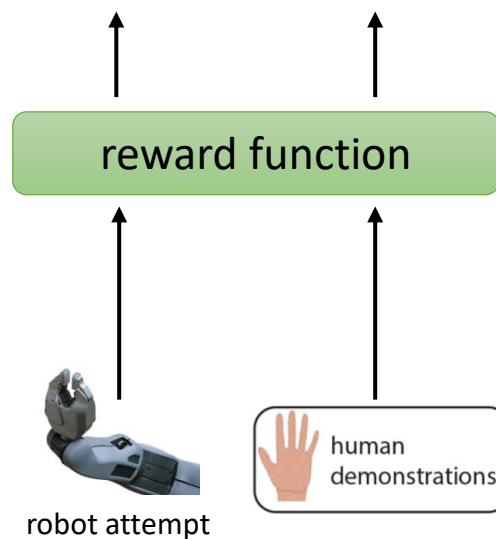


IRL as adversarial optimization

Guided Cost Learning

Finn et al., ICML 2016

minimized maximized



learns distribution $p(\tau)$ such that
demos have max likelihood

$p(\tau) \propto \exp(r(\tau))$ (MaxEnt model)

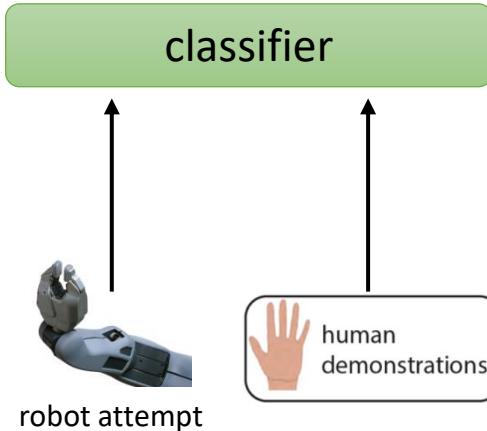
$$D(\tau) = \frac{\frac{1}{Z} \exp(r(\tau))}{\frac{1}{Z} \exp(r(\tau)) + \pi(\tau)}$$

actually the
same thing!

Generative Adversarial Imitation Learning

Ho & Ermon, NIPS 2016

False True



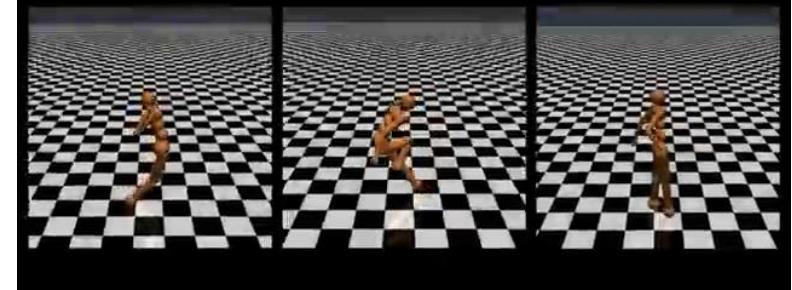
$D(\tau) = \text{probability } \tau \text{ is a demo}$

use $\log D(\tau)$ as “reward”

$D(\tau) = \text{some classifier}$

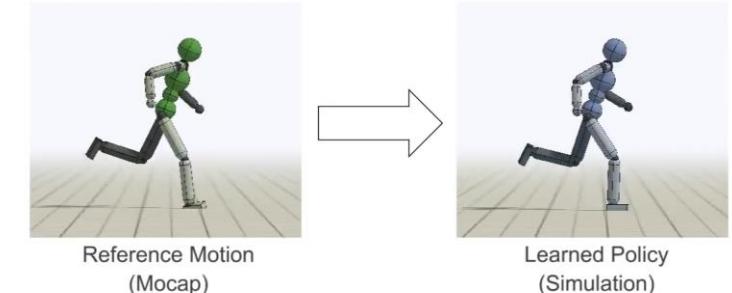
Humanoid expert policies

Run forwards Run backwards Balance



Hausman, Chebotar, Schaal, Sukhatme, Lim

Motion Imitation



the goal is to train a simulated character to imitate the motion.

Peng, Kanazawa, Toyer, Abbeel, Levine

Suggested Reading on Inverse RL

Classic Papers:

Abbeel & Ng ICML '04. *Apprenticeship Learning via Inverse Reinforcement Learning.*

Good introduction to inverse reinforcement learning

Ziebart et al. AAAI '08. *Maximum Entropy Inverse Reinforcement Learning.* Introduction to probabilistic method for inverse reinforcement learning

Modern Papers:

Finn et al. ICML '16. *Guided Cost Learning.* Sampling based method for MaxEnt IRL that handles unknown dynamics and deep reward functions

Wulfmeier et al. arXiv '16. *Deep Maximum Entropy Inverse Reinforcement Learning.* MaxEnt inverse RL using deep reward functions

Ho & Ermon NIPS '16. *Generative Adversarial Imitation Learning.* Inverse RL method using generative adversarial networks

Fu, Luo, Levine ICLR '18. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning

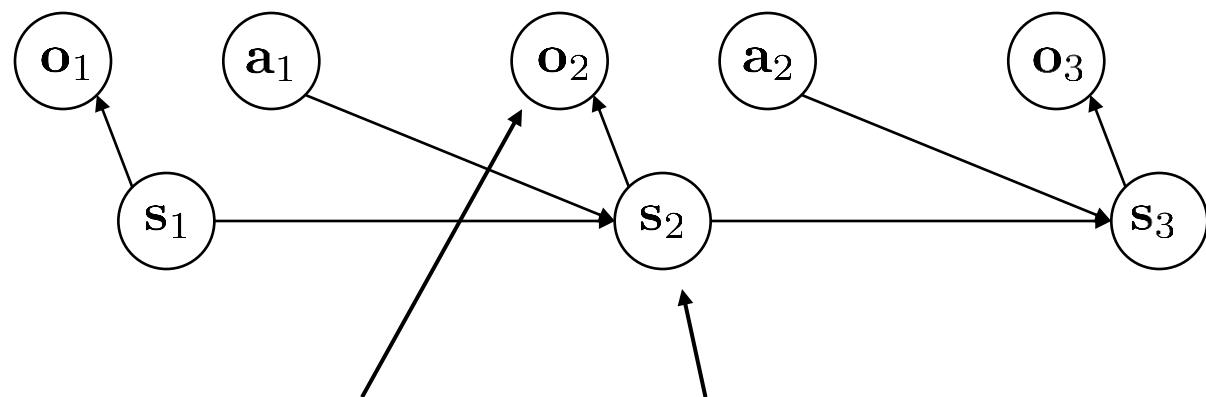
RL with Sequence Models

CS 285

Instructor: Sergey Levine
UC Berkeley



Beyond MDPs



doesn't obey the
Markov property

not known

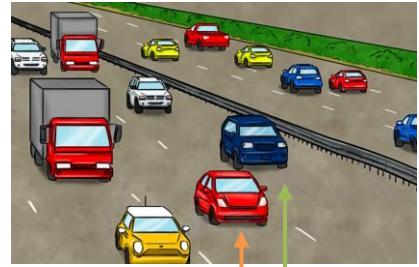


\mathbf{o}_t – observation



\mathbf{s}_t – state

most real-world problems are like this!



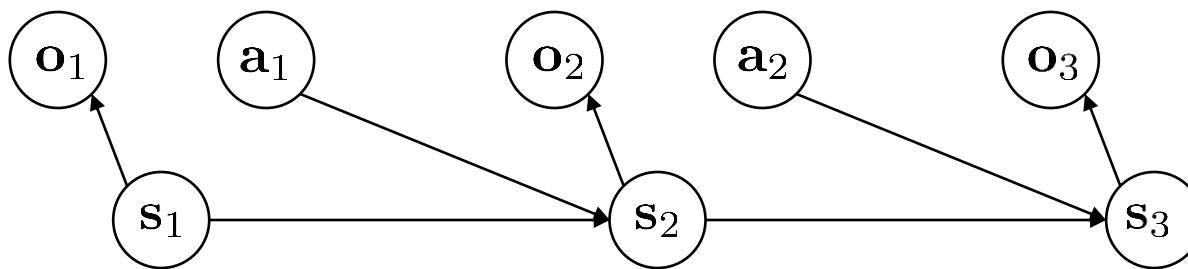
Q1: Any people in the shot?

A1: No, there aren't any.

Q10: Are they facing each other?

A10: They aren't.

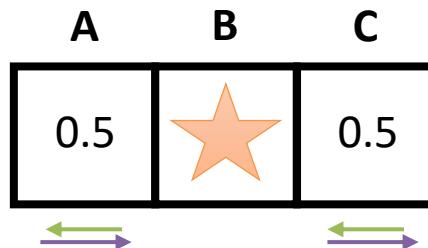
Partially observed MDPs can be weird



Example 1: information-gathering actions



Example 2: stochastic optimal policies



Which methods handle partial observability?

Policy gradients

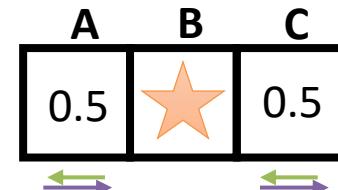
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) A(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \quad ?? \quad \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t}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

Value-based methods

$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') \quad ?? \quad Q(\mathbf{o}, \mathbf{a}) \leftarrow r(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{o}', \mathbf{a}') \quad \text{Trick question: what does "handle" mean?}$$

Model-based RL methods

$$\hat{p}(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \quad ?? \quad \hat{p}(\mathbf{o}' | \mathbf{o}, \mathbf{a})$$



handle = find best policy in policy class

Which methods handle partial observability?

Policy gradients

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

this is OK (no Markov property!)

Key point: advantage is a function of the state \mathbf{s}_t

it does *not* depend on \mathbf{s}_{t-1}

that's why it's OK to use $r_t + \gamma \hat{V}(\mathbf{s}_{t+1}) - \hat{V}(\mathbf{s}_t)$

it is *not* OK to train $\hat{V}(\mathbf{o}_t)$

this takes some care

every time we see this state,
we expect to get this value,
regardless of past states

past observations **do**
matter for this value

Which methods handle partial observability?

Policy gradients

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

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{o}_{i,t}) \right) \left(\sum_{t=1}^T r(\mathbf{o}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{o}_{i,t}) (r_{i,t} + \gamma \hat{V}(\mathbf{o}_{i,t+1}) - \hat{V}(\mathbf{o}_{i,t})) \quad \text{X}$$

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

Which methods handle partial observability?

Value-based methods

$$Q(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q(s', a') \quad ?? \quad Q(o, a) \leftarrow r(o, a) + \gamma \max_{a'} Q(o', a')$$

Remember:

Value-based methods do not work
without the Markov property

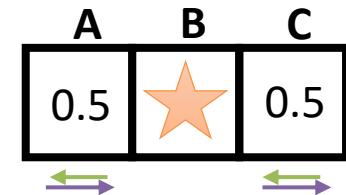
Key point: advantage is a function of the state s_t

it does *not* depend on s_{t-1}

that's why it's OK to use $r_t + \hat{V}(s_{t+1}) - \hat{V}(s_t)$

it is *not* OK to train $\hat{V}(o_t)$

past observations **do**
matter for this value



every time we see this state,
we expect to get this value,
regardless of past states

Which methods handle partial observability?

Model-based RL methods

$$\hat{p}(s'|s, a) \xrightarrow{\text{??}} \hat{p}(o'|o, a) \times$$

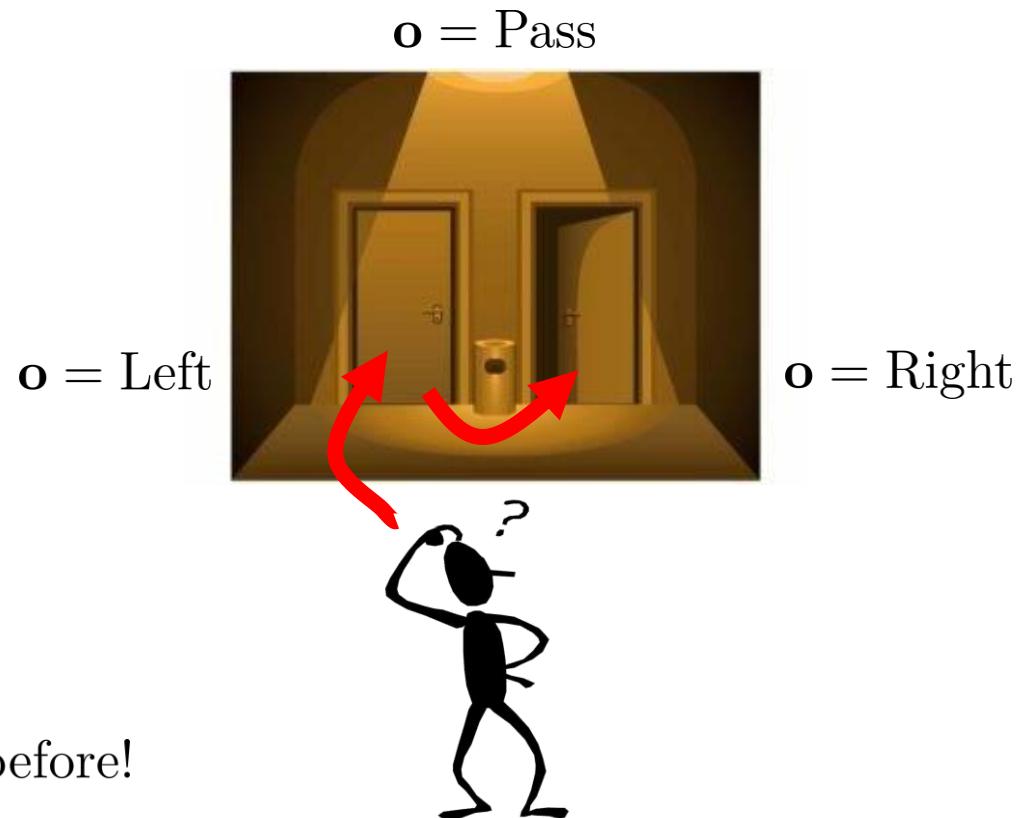
Example showing why this is such a bad idea:

$$p(o' = \text{Pass} | o = \text{Left}, a = \text{Open}) = 0.5$$

50% probability to open *each time* you try

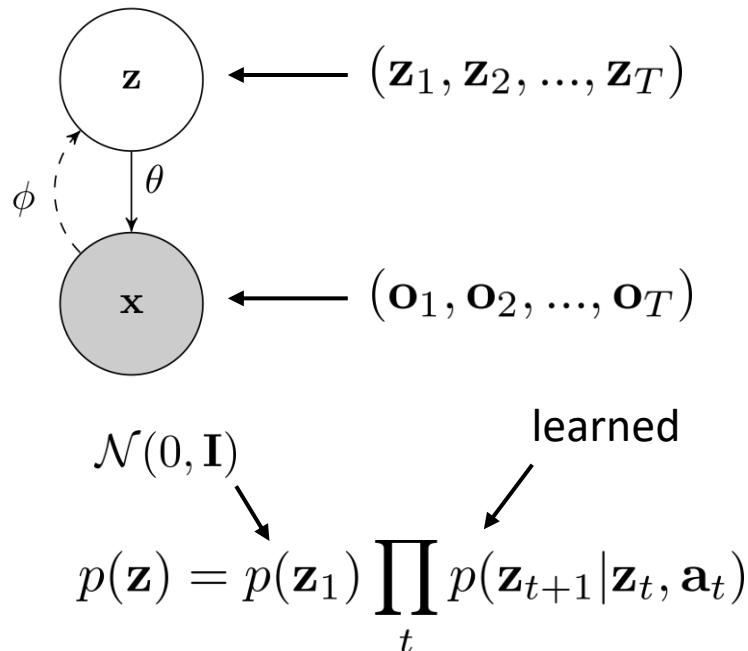
Just keep trying!

$$p(o' = \text{Pass} | o = \text{Left}, a = \text{Open}) = 0 \text{ if it didn't open before!}$$



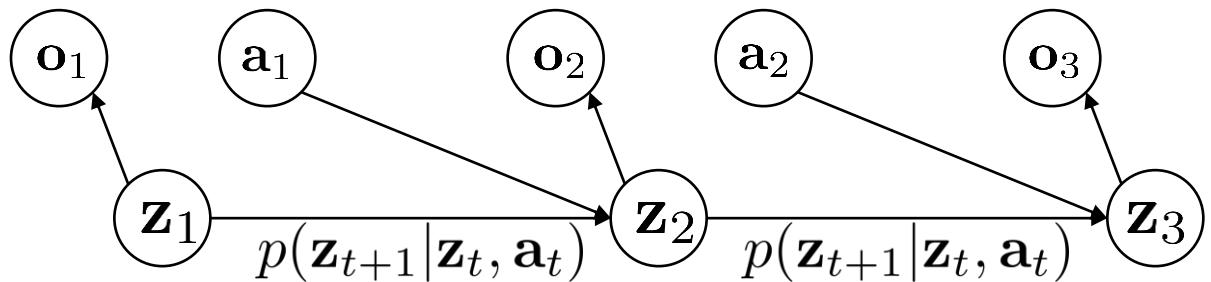
State space models

Can we *learn* a Markovian state space?



$$p_\theta(\mathbf{o}|\mathbf{z}) = \prod_t p(\mathbf{o}_t|\mathbf{z}_t)$$

$$q_\phi(\mathbf{z}|\mathbf{o}) = \prod_t q_\phi(\mathbf{z}_t|\mathbf{o}_{1:t})$$



This can work quite well!

$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$$



$$Q(\mathbf{o}, \mathbf{a}) \leftarrow r(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{o}', \mathbf{a}')$$

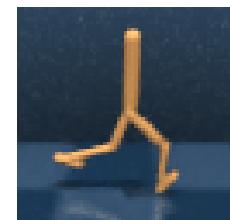


$$Q(\mathbf{z}, \mathbf{a}) \leftarrow r(\mathbf{z}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{z}', \mathbf{a}')$$



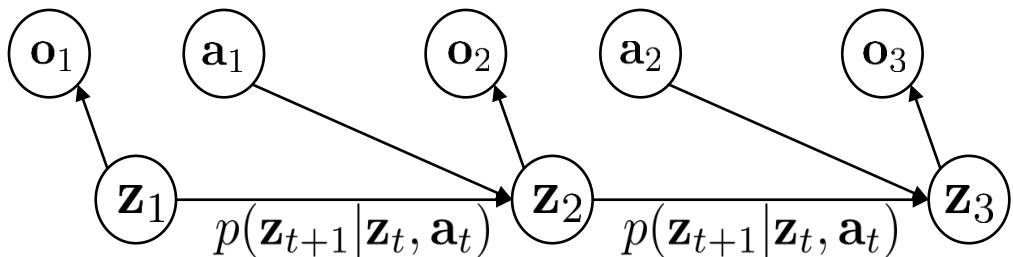
Why might this not be enough?

Prediction can be *hard*

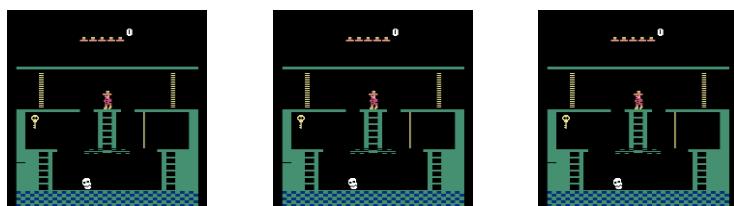


Maybe we don't need good prediction to get high rewards

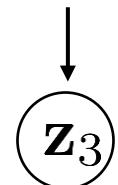
History states



$$q_{\phi}(\mathbf{z}|\mathbf{o}) = \prod_t q_{\phi}(\mathbf{z}_t|\mathbf{o}_{1:t})$$



sequence model



state is inferred
from a *history*

state is a *function*
of history

can we just give
the history to our
value function?

$$\mathbf{s}_t = (\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t)$$

Does that work?

Does that obey the Markov property?

tells *nothing* we didn't know from \mathbf{s}_t !

$$\mathbf{s}_{t+1} \perp \mathbf{s}_{t-1} | \mathbf{s}_t$$

$(\mathbf{o}_1, \dots, \mathbf{o}_{t-1})$ $(\mathbf{o}_1, \dots, \mathbf{o}_t)$

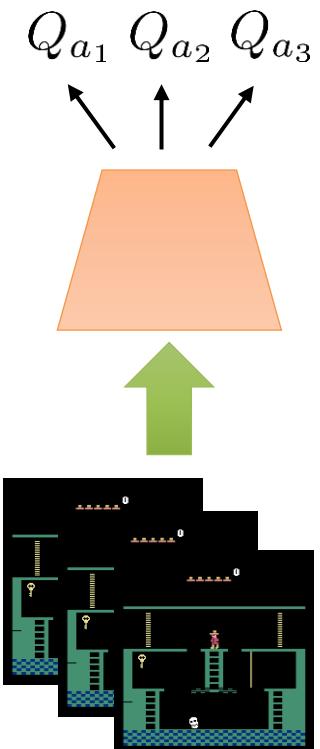
$$Q(\mathbf{o}_1, \dots, \mathbf{o}_t, \mathbf{a}) \leftarrow r(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{o}_1, \dots, \mathbf{o}_{t+1}, \mathbf{a}')$$



Model architectures

$$Q(\mathbf{o}_1, \dots, \mathbf{o}_t, \mathbf{a}) \leftarrow r(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{o}_1, \dots, \mathbf{o}_{t+1}, \mathbf{a}')$$

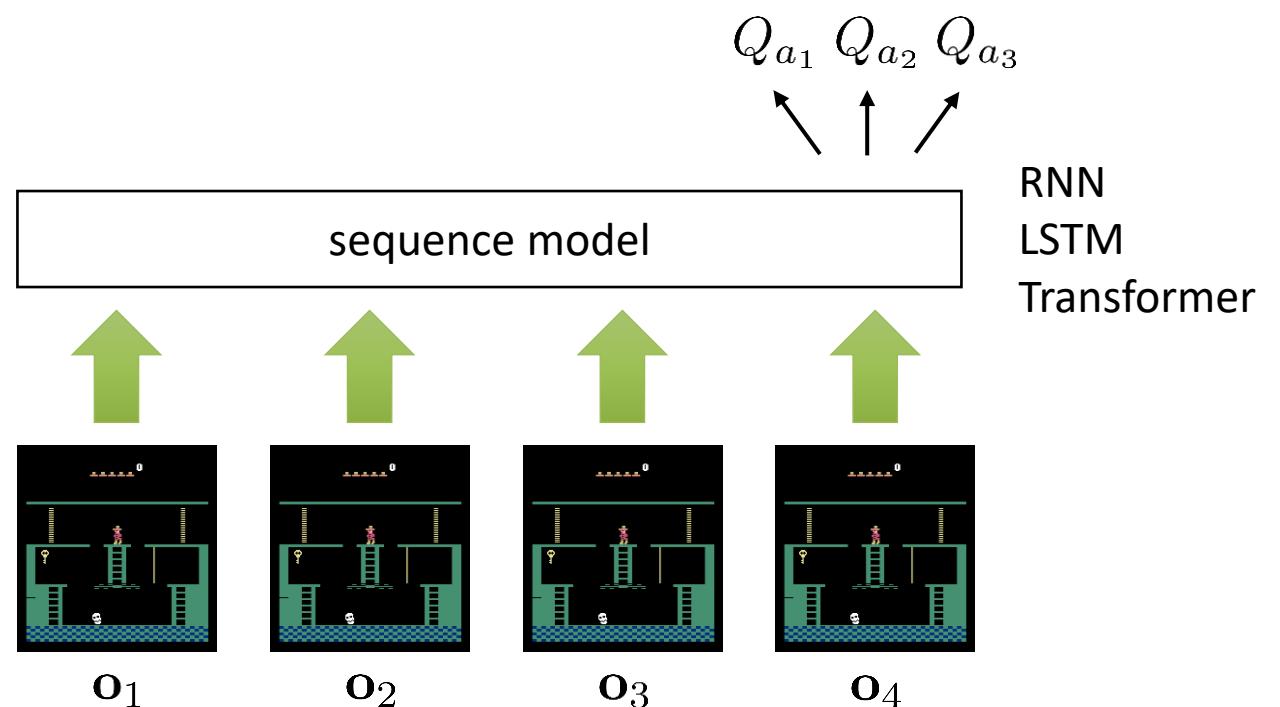
how to represent this?



fixed (short)
history

Is that bad?

Sometimes...



A practical detail...

Standard deep Q-learning:

1. Collect transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, add to \mathcal{R}
2. Sample batch $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)\}_{i=1}^n$ from \mathcal{R}
3. Update Q-function on batch

Deep Q-learning with history states:

1. Collect $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$, get history by cat'ing $\mathbf{o}_1, \dots, \mathbf{o}_{t-1}$, add to \mathcal{R}
2. Sample batch $\{(\mathbf{o}_{1,i}, \dots, \mathbf{o}_{t,i}, \mathbf{a}_{t,i}, \mathbf{o}_{1,i}, \dots, \mathbf{o}_{t+1,i})\}_{i=1}^n$ from \mathcal{R}
3. Update Q-function on batch

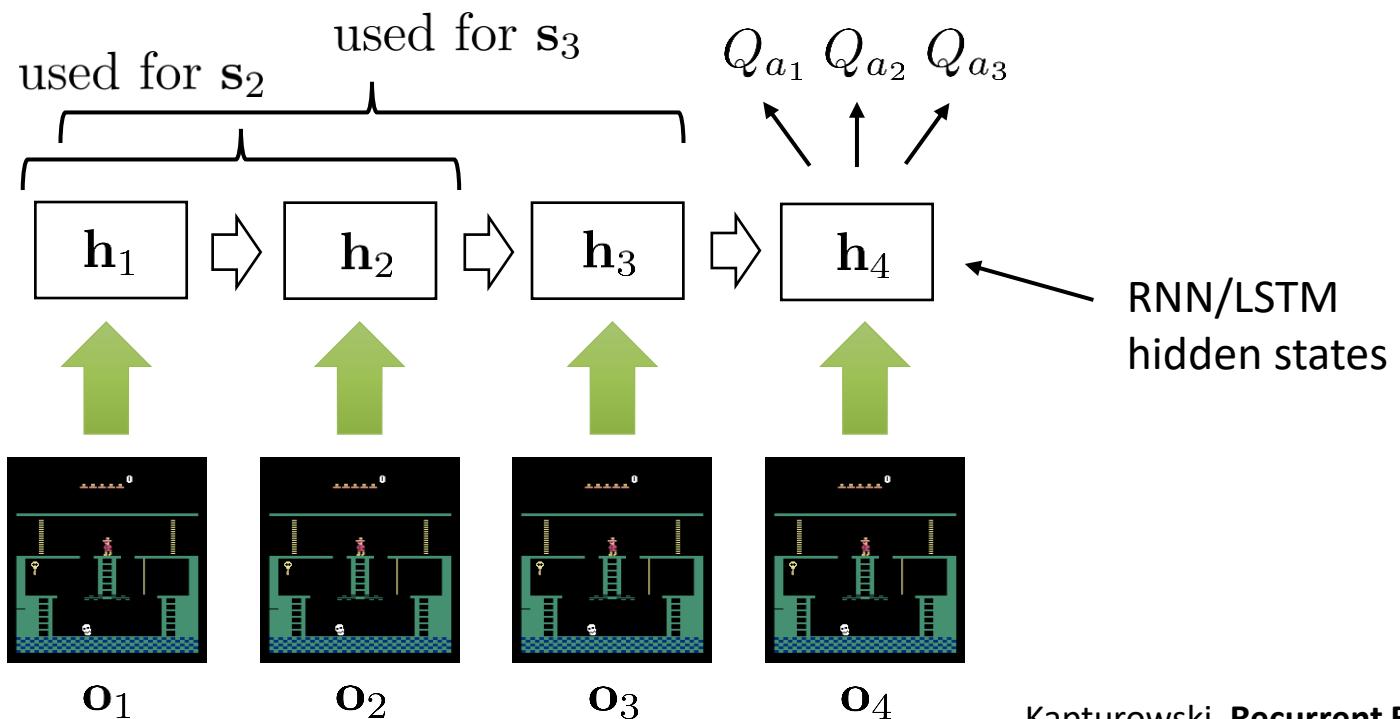


Super expensive

A practical detail...

Deep Q-learning with history states:

1. Collect $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$, get history by cat'ing $\mathbf{o}_1, \dots, \mathbf{o}_{t-1}$, add to \mathcal{R}
2. Sample batch $\{(\mathbf{o}_{1,i}, \dots, \mathbf{o}_{t,i}, \mathbf{a}_{t,i}, \mathbf{o}_{1,i}, \dots, \mathbf{o}_{t+1,i})\}_{i=1}^n$ from \mathcal{R}
3. Update Q-function on batch

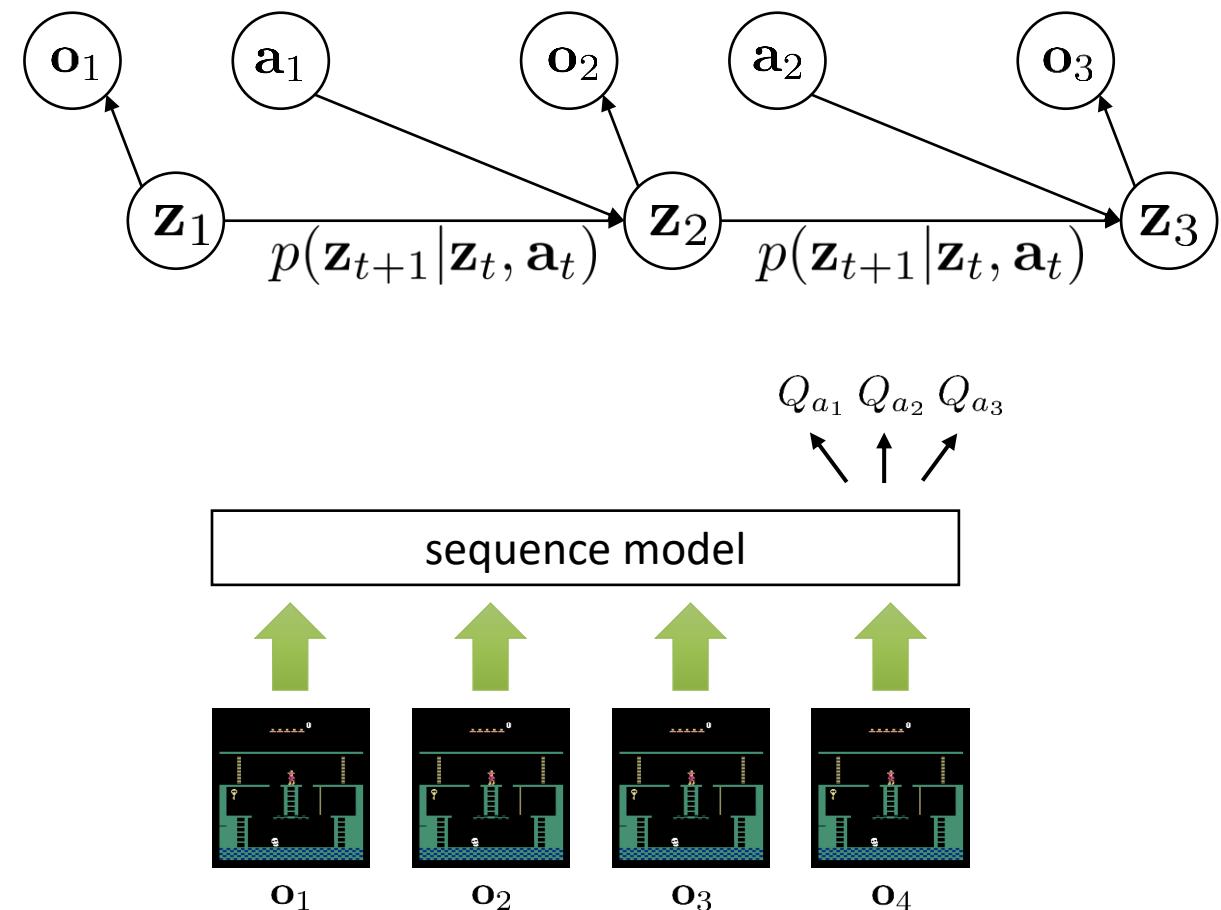
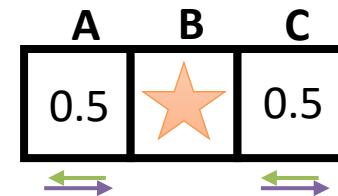


can we reuse \mathbf{h}_t ?

Key idea: store \mathbf{h}_t in \mathcal{R}
details a little subtle, see paper
not clear how to do w/ transformer

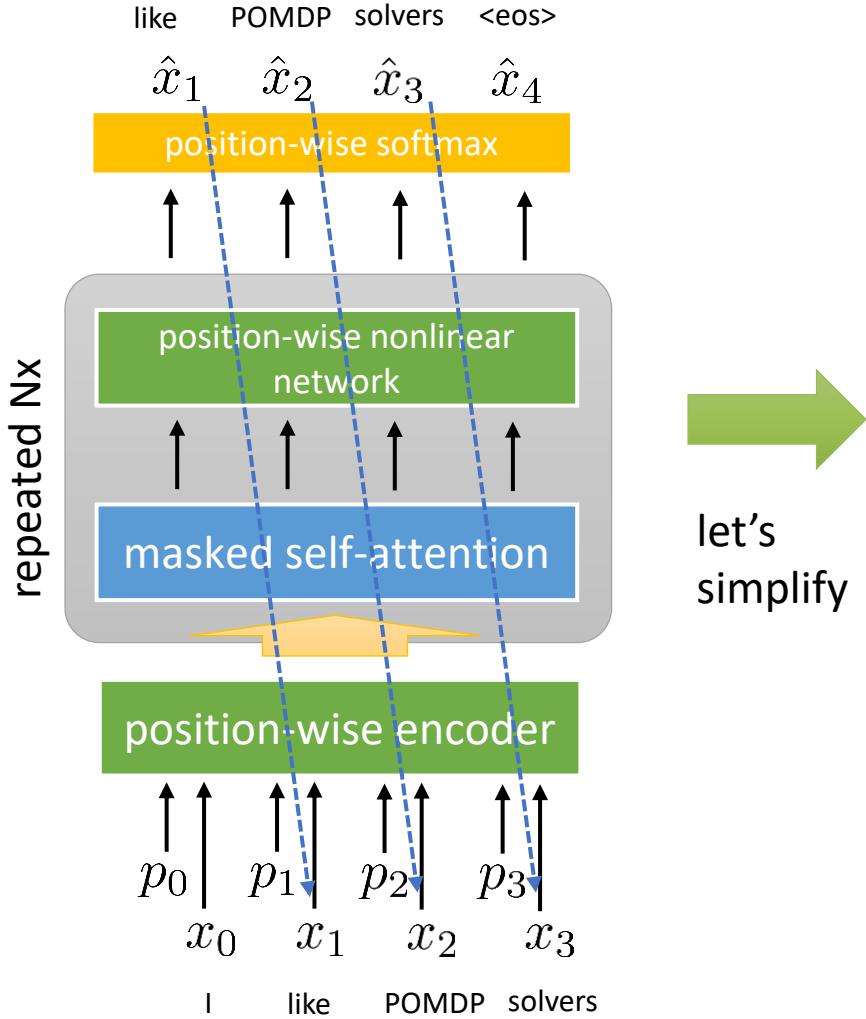
Recap & overview

- POMDPs are *weird*
- Some methods “just do it”
 - But most efficient ones don’t, because they require value functions
 - Even those that do only get the best *memoryless* policy
- We could *learn* a Markovian state space with models
- We could also just use *history states*, which just means using a sequence model to read in observation histories

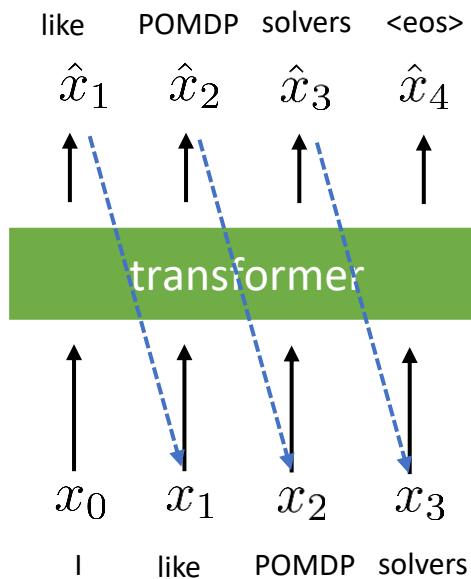


RL and language models

Language models



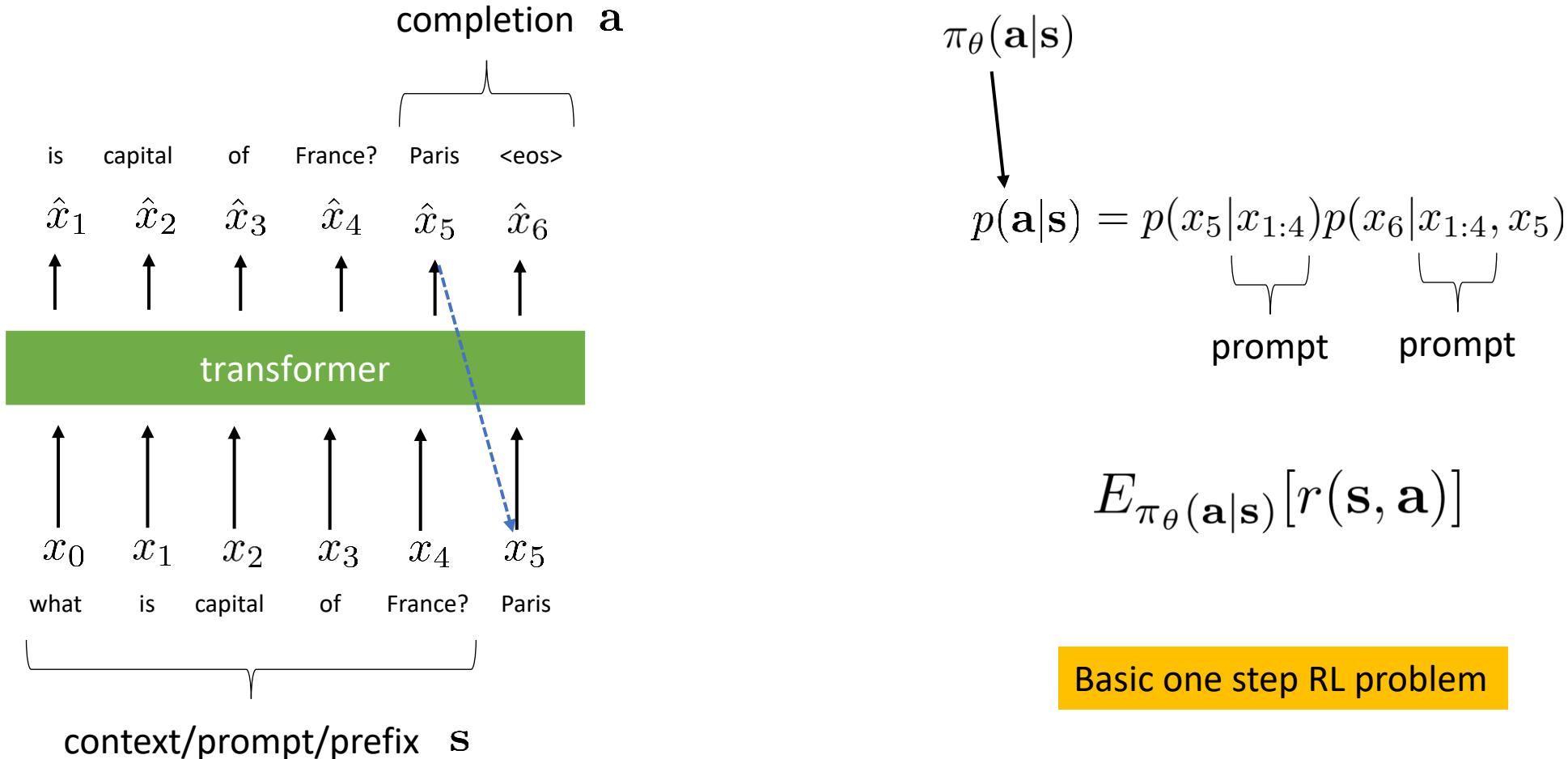
let's
simplify



- Language models are *typically* trained with supervised learning
- But we can also train them with RL if what we want is to maximize some reward function, rather than just represent the data distribution
 - Why?
- Some questions:
 - What is the (PO)MDP?
 - What is the reward?
 - What algorithm to use?

We have a few choices to make!

A basic formulation



Language models and policy gradients

$$\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) = \nabla_{\theta} \log p(x_5|x_{1:4}) + \nabla_{\theta} \log p(x_6|x_{1:4}, x_5)$$

$$\begin{aligned}\nabla_{\theta} E_{\pi_{\theta}(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a})] &= E_{\pi_{\theta}(\mathbf{a}|\mathbf{s})}[\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|\mathbf{s}) r(\mathbf{s}, \mathbf{a})] \\ &\approx \frac{1}{N} \sum_i \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_i|\mathbf{s}) r(\mathbf{s}, \mathbf{a}_i) \quad \text{samples from } \pi_{\theta}(\mathbf{a}|\mathbf{s}) \\ &\approx \frac{1}{N} \sum_i \frac{\pi_{\theta}(\mathbf{a}_i|\mathbf{s})}{\bar{\pi}(\mathbf{a}_i|\mathbf{s})} \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_i|\mathbf{s}) r(\mathbf{s}, \mathbf{a}_i) \quad \text{samples from } \bar{\pi}(\mathbf{a}|\mathbf{s})\end{aligned}$$

Why might we prefer this?

Language models and policy gradients

$$\nabla_{\theta} E_{\pi_{\theta}(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a})] \approx \frac{1}{N} \sum_i \underbrace{\frac{\pi_{\theta}(\mathbf{a}_i|\mathbf{s})}{\bar{\pi}(\mathbf{a}_i|\mathbf{s})} \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_i|\mathbf{s}) r(\mathbf{s}, \mathbf{a}_i)}_{\hat{\nabla}(\theta, \bar{\pi}, \{\mathbf{a}_i\})}$$

importance-weighted estimator (e.g., PPO)



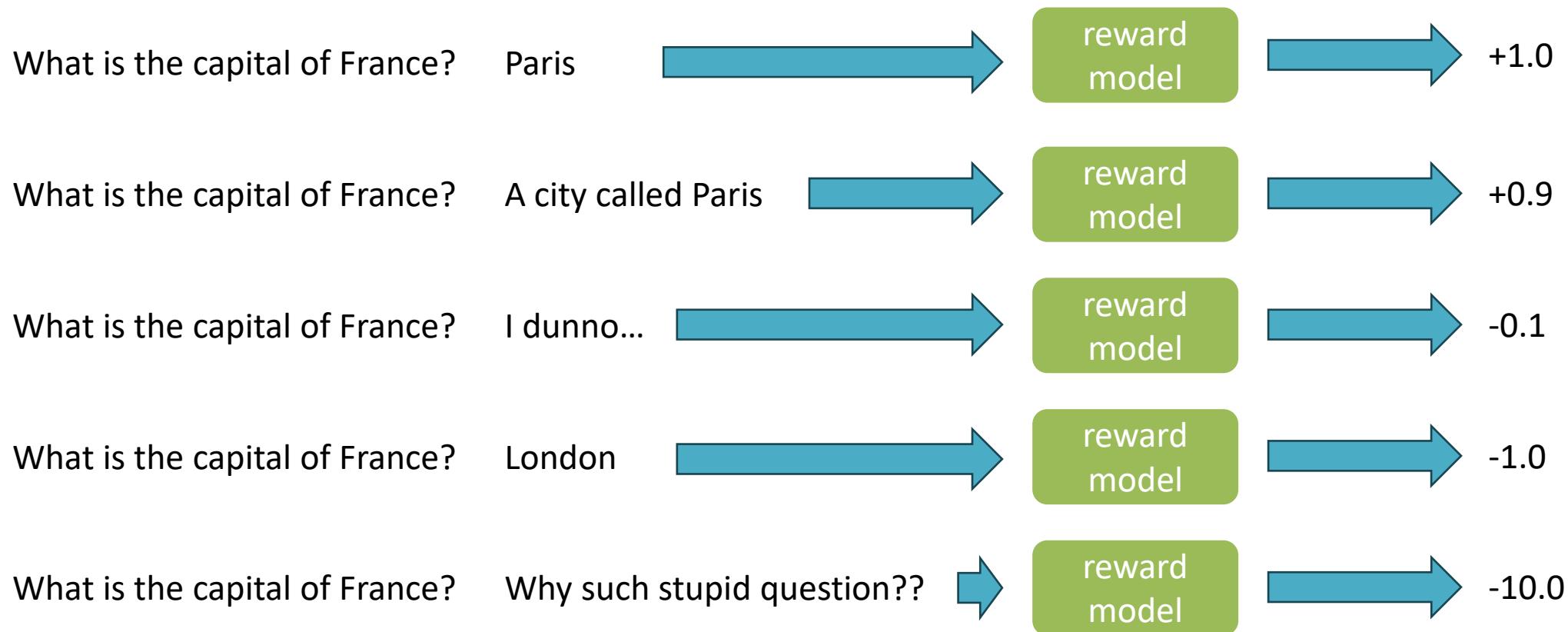
1. sample batch $\mathcal{B} = \{\mathbf{a}_i\}$, $\mathbf{a}_i \sim \pi_{\theta}(\mathbf{a}|\mathbf{s})$
2. evaluate $r(\mathbf{s}, \mathbf{a}_i)$ for each $\mathbf{a}_i \in \mathcal{B}$
3. $\bar{\pi} \leftarrow \pi_{\theta}$
4. sample *minibatch* $\mathcal{M} \subset \mathcal{B}$
5. $\theta \leftarrow \theta + \alpha \hat{\nabla}(\theta, \bar{\pi}, \mathcal{M})$

repeat K times

what is this?

Learned rewards

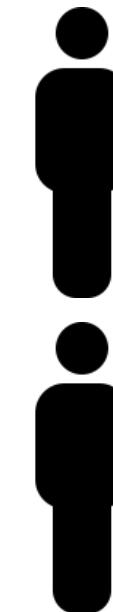
What if $r(s, a)$ is itself a neural network?



RL from human feedback

How do we train the reward model $r_\psi(\mathbf{s}, \mathbf{a})$?

What is the capital of France? Paris



+1.0

How do people know
these numbers?

What is the capital of France? Why such stupid question??



-10.0

("What is the capital of France? Paris", +1.0)

("What is the capital of France? Why such stupid question??", -10.0)

...



supervised
learning



reward
model

Rewards from preferences

What is the capital of France?

s

A: Paris

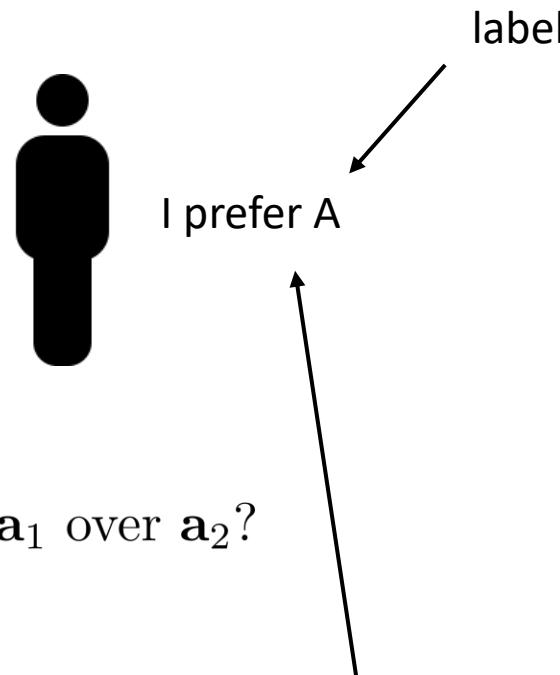
a₁

B: Why such stupid question??

a₂

Prediction: given (**s, a₁, a₂**), how likely is a person to prefer **a₁** over **a₂**?

$$p(\mathbf{a}_1 \text{ preferred over } \mathbf{a}_2) = \frac{\exp(r_\psi(\mathbf{s}, \mathbf{a}_1))}{\exp(r_\psi(\mathbf{s}, \mathbf{a}_1)) + \exp(r_\psi(\mathbf{s}, \mathbf{a}_2))}$$



maximize likelihood
w.r.t. params ψ of r

Overall method

- 
1. Run supervised training (or finetuning) to get initial $\pi_\theta(\mathbf{a}|\mathbf{s})$
 2. For each \mathbf{s} sample K answers $\mathbf{a}_k \sim \pi(\mathbf{a}|\mathbf{s})$, add to dataset $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_{i,1}, \dots, \mathbf{a}_{i,K})\}$
 3. Get humans to label which $\mathbf{a}_{i,k}$ they prefer for each \mathbf{s}_i
 4. Train r_ψ using labeled dataset \mathcal{D}
 5. Update π_θ using RL with reward $r_\psi(\mathbf{s}, \mathbf{a})$

Ziegler et al. **Fine-Tuning Language Models from Human Preferences**. 2019.

Ouyang et al. **Training language models to follow instructions with human feedback**. 2019.

Some issues...

- Human preferences are expensive

This is **model-based RL!**

Most preference data comes from the initial supervised-trained model, each iteration of RL typically adds a smaller set of preferences

Many iterations of RL (including generating new samples from policy) per each iteration of preference gathering

Offline (model-based) RL if we only collect preferences once

- “Overoptimization”

original
supervised policy

$$E_{\pi_\theta(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a})] - \beta D_{\text{KL}}(\pi_\theta \parallel \pi_\beta) = E_{\pi_\theta(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a}) + \beta \log \pi_\beta(\mathbf{a}|\mathbf{s}) - \beta \log \pi_\theta(\mathbf{a}|\mathbf{s})]$$

- Reward model needs to be very good

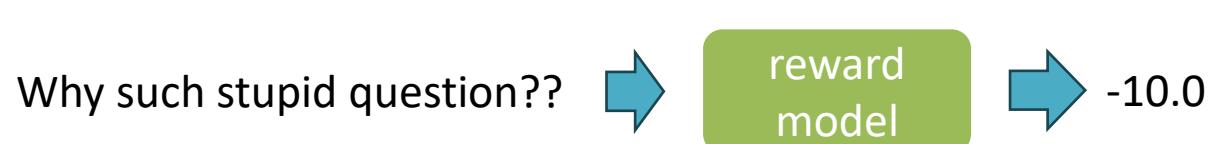
Reward model is typically itself a large transformer

So what's the problem?

Recap & overview

- We can train language models with policy gradients
 - It's a bandit problem (for now)
- We can use a reward model
 - Typically this needs to be learned!
- We can learn the reward model from human preferences
 - This can be more convenient than direct supervision
 - This ends up being (technically) a model-based RL algorithm
 - Potentially an offline model-based RL algorithm
- Details to take care of
 - Minimize human labeling
 - Overoptimization
 - Use powerful reward models

$$\sum_i \frac{\pi_\theta(\mathbf{a}_i | \mathbf{s})}{\bar{\pi}(\mathbf{a}_i | \mathbf{s})} \nabla_\theta \log \pi_\theta(\mathbf{a}_i | \mathbf{s}) r(\mathbf{s}, \mathbf{a}_i)$$

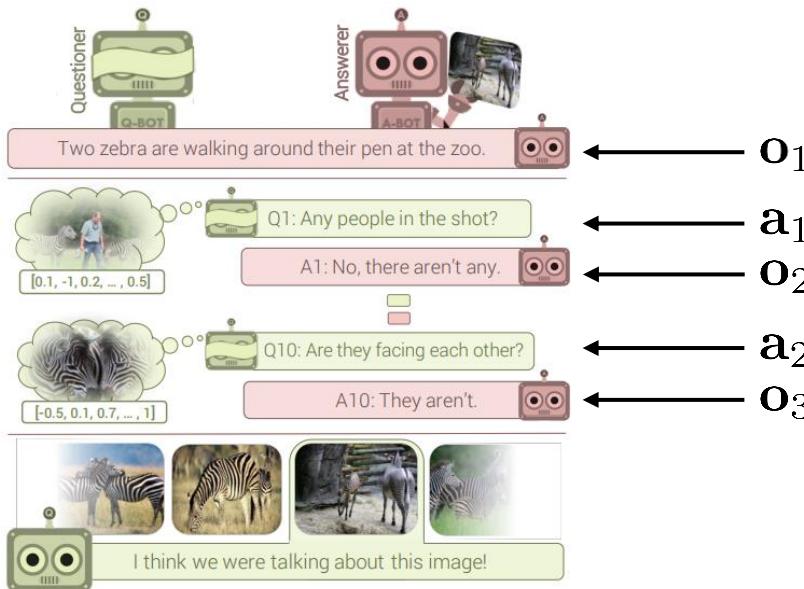


$$\frac{\exp(r_\psi(\mathbf{s}, \mathbf{a}_1))}{\exp(r_\psi(\mathbf{s}, \mathbf{a}_1)) + \exp(r_\psi(\mathbf{s}, \mathbf{a}_2))}$$

$$E_{\pi_\theta(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a}) + \beta \log \pi_\beta(\mathbf{a}|\mathbf{s}) - \beta \log \pi_\theta(\mathbf{a}|\mathbf{s})]$$

Multi-step RL and language models

Multi-step RL with language models



action: what the bot says

observation: what the human says

state: the history $s_3 = \{o_1, a_1, o_2, a_2, o_3\}$

reward: dialogue outcome

- Dialogue systems
- Assistant chat bots
- Tool use (e.g., using command line tools)
- Playing games

Das et al. Learning Cooperative Visual Dialog Agents with Deep Reinforcement Learning. 2017.

This is **not** RLHF

RLHF

learn from human preferences

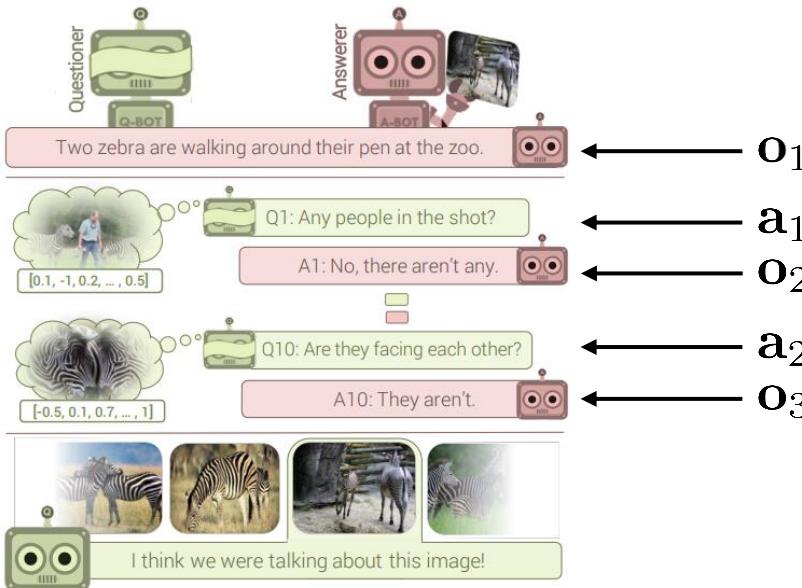
episode = single answer

sequential decision making

learn from **dialogue outcome**

episode = whole dialogue

Multi-step RL with language models



Das et al. Learning Cooperative Visual Dialog Agents with Deep Reinforcement Learning. 2017.

action: what the bot says

observation: what the human says

state: the history $s_3 = \{o_1, a_1, o_2, a_2, o_3\}$

reward: dialogue outcome

How to train?

Policy gradients

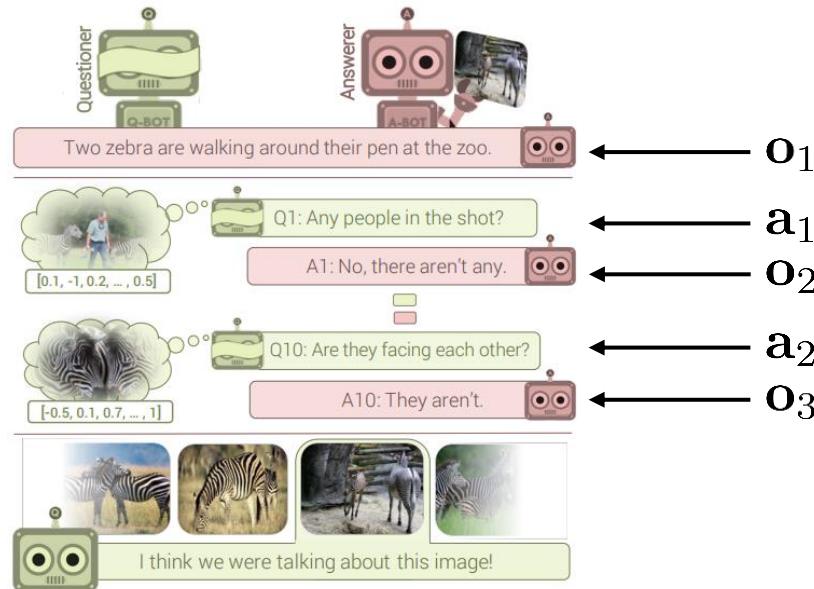
requires samples from human
could work, but expensive

Value-based methods

could learn offline from data!

What is a time step

Per-utterance time step

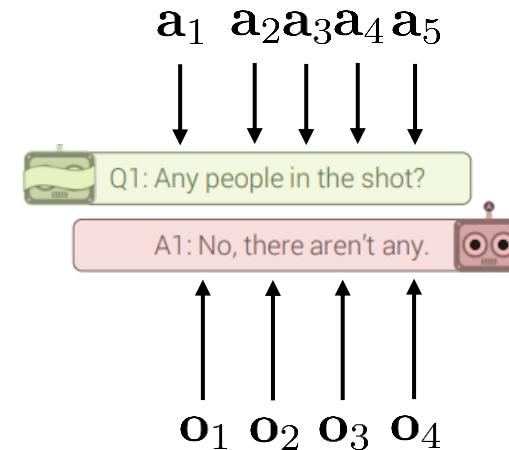


$$\mathbf{s}_3 = \{\mathbf{o}_1, \mathbf{a}_1, \mathbf{o}_2, \mathbf{a}_2, \mathbf{o}_3\}$$

+ natural choice, short horizons

- huge action space

Per-token time step



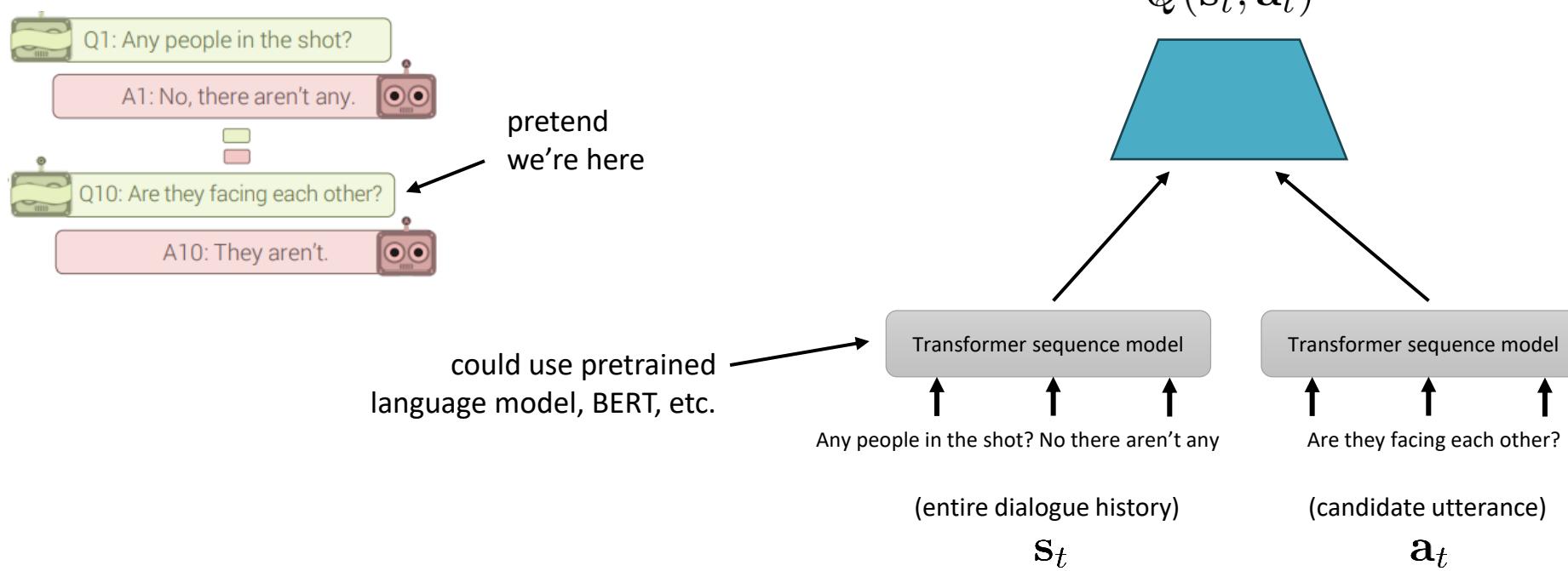
+ simple discrete actions

- very long horizons

Value-based RL with language models

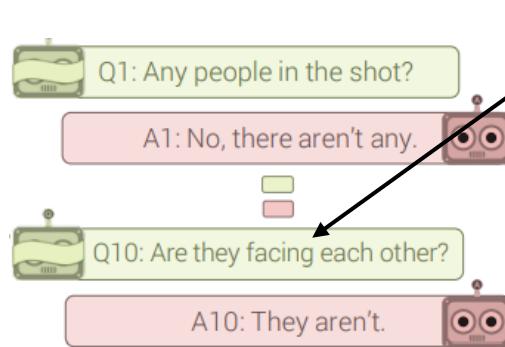
With per-utterance time steps

$$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$$



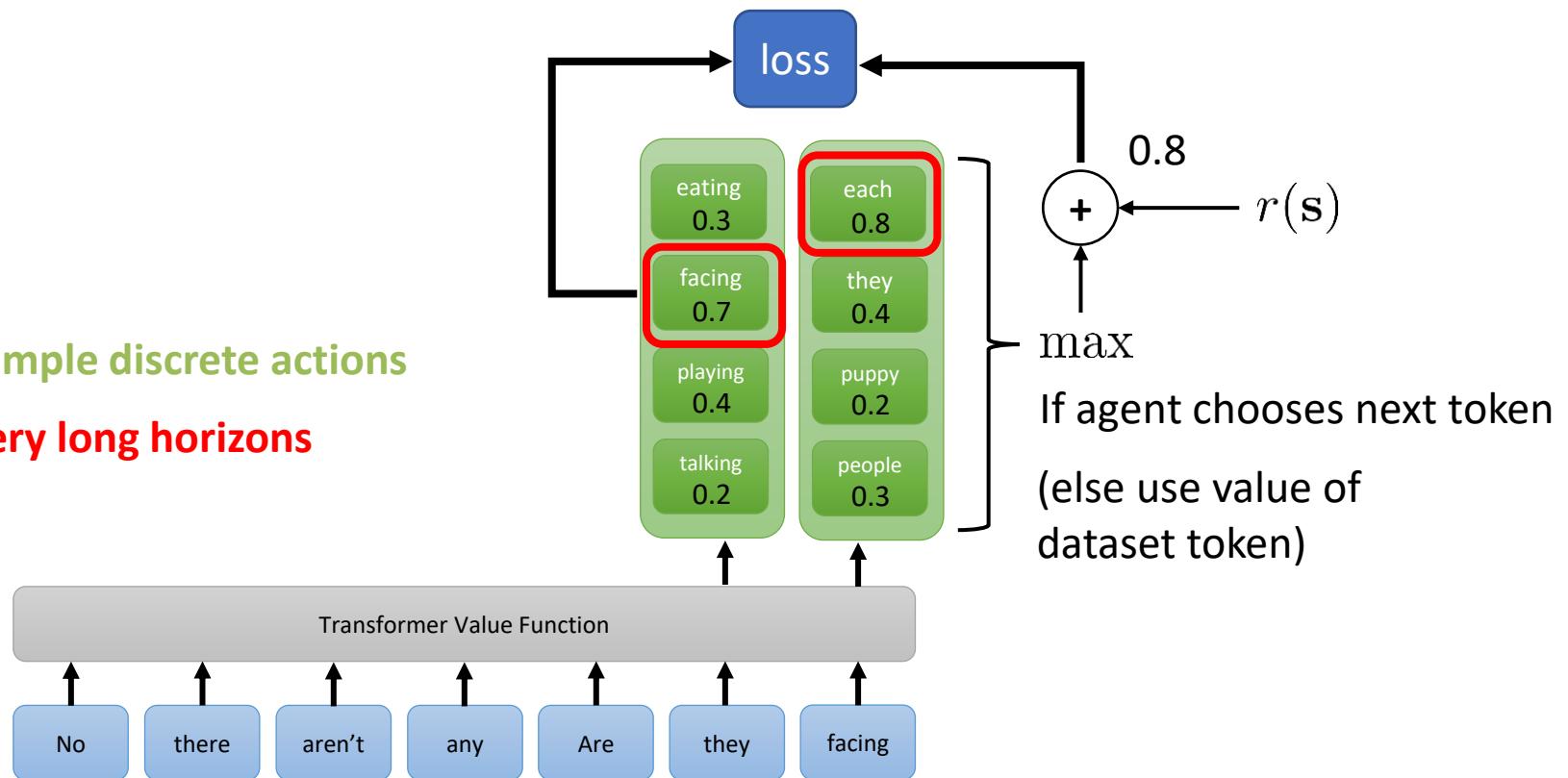
Value-based RL with language models

With per-token time steps



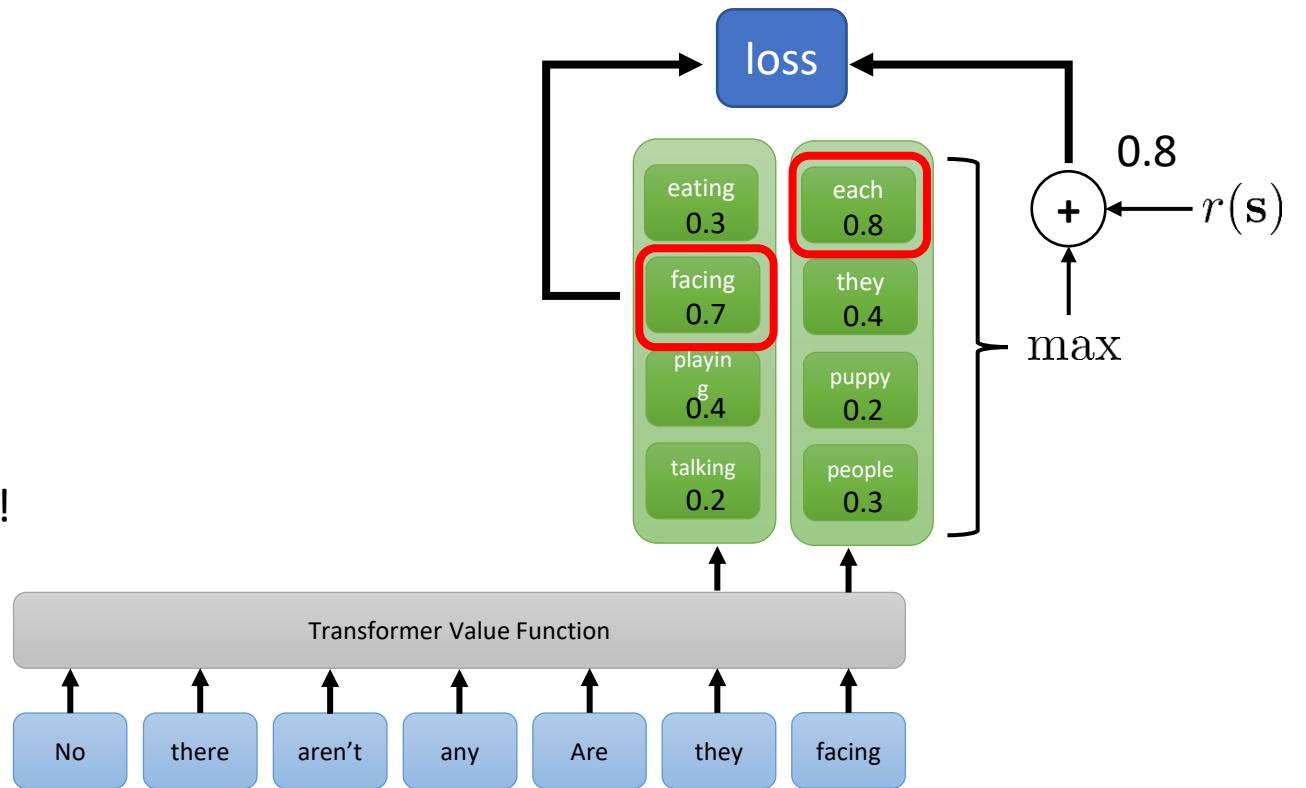
pretend
we're here

+ simple discrete actions
- very long horizons



Putting it all together

- Usual value-based details apply
 - Target network
 - Replay buffer
 - Double-Q trick
 - Etc.
- Can be used with either online or offline RL
- But value-based methods particularly useful in the offline setting
- That means that we need to take care of the details!
 - Handling distributional shift
 - Policy constraint: KL-divergence on actor
 - CQL-style penalty
 - IQN-style backup
 - No single best answer (yet)

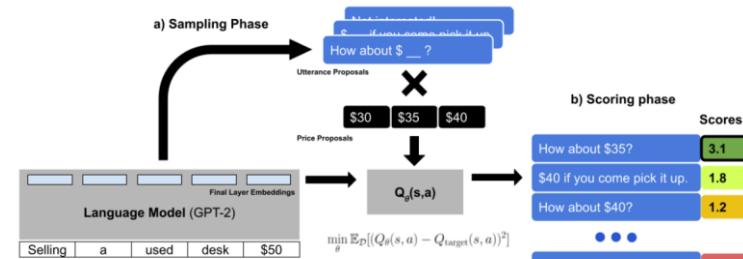


Some examples

Human-Centric Dialog Training via Offline Reinforcement Learning
Jaques et al. 2020



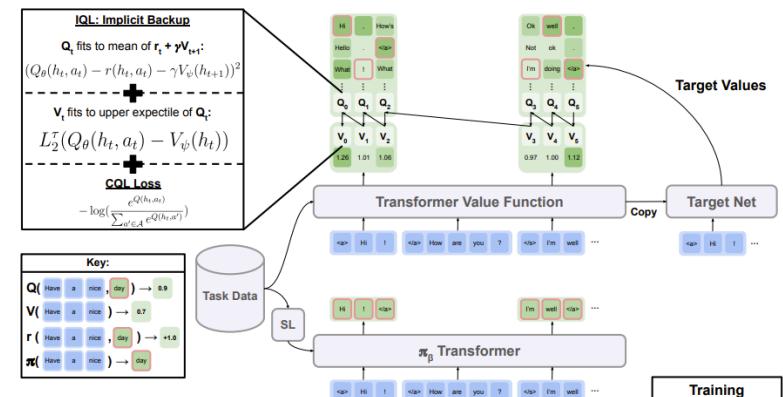
CHAI: A Chatbot AI for Task-Oriented Dialogue with Offline Reinforcement Learning
Verma et al. 2022



- Actor-critic + policy constraint (KL divergence)
- Reward from human user sentiment
- Time step = utterance

- Q-function + CQL
- Reward from task (Craigslist negotiation)
- Time step = utterance

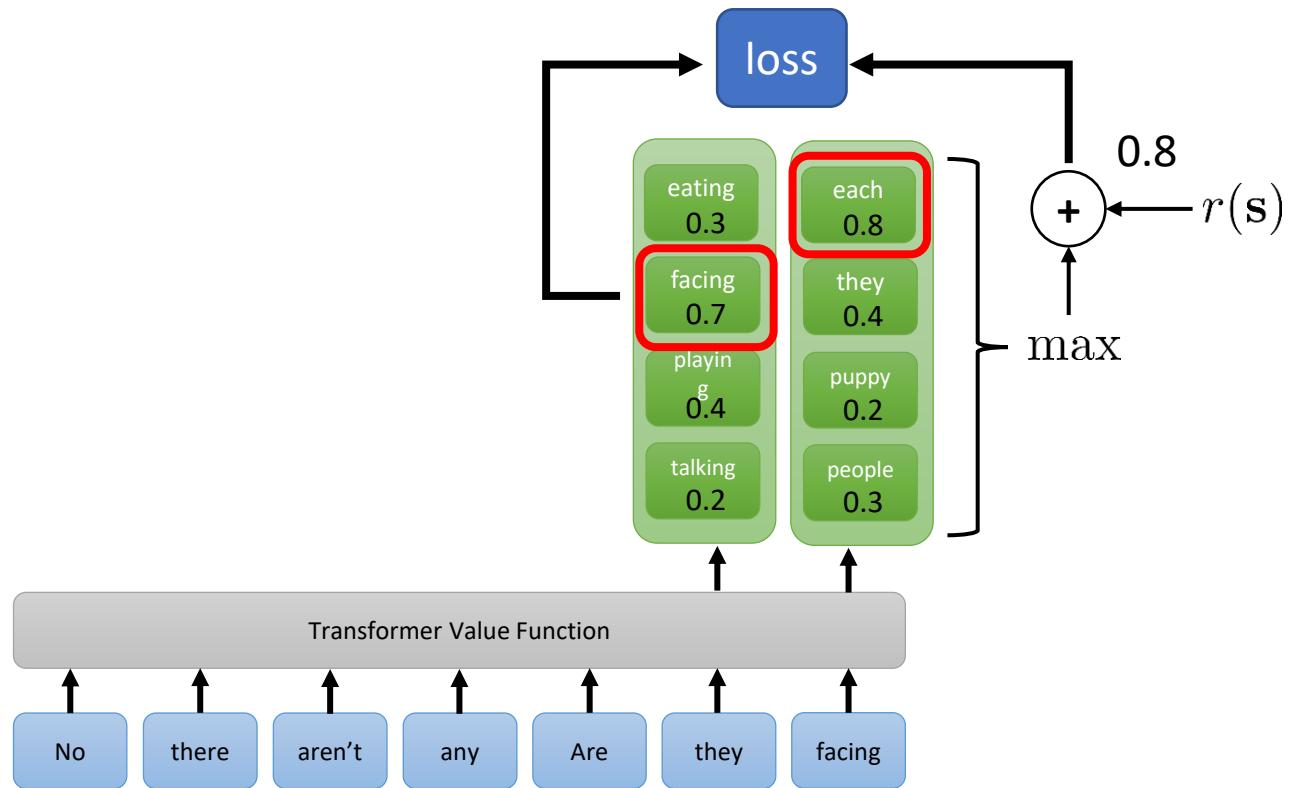
Offline RL for Natural Language Generation with Implicit Language Q Learning
Snell et al. 2022



- Q-function with IQL + CQL
- Policy extraction with BC actor
- Reward from task (visual dialogue)
- Time step = token

Recap & overview

- Multi-step language interactions (e.g., dialogue) are a POMDP
 - Can be defined as per-utterance or per-token
- In principle, any RL method can be used (with history states)
- In practice, we might really want an offline RL method
 - But not necessarily (e.g., text games, tools)
- Value-based methods treat either **utterances** or **tokens** as actions, build Q-functions with **history states**
- Same details & tricks as regular (offline) value-based methods apply



Meta-Learning & Transfer Learning

CS 285

Instructor: Sergey Levine
UC Berkeley



What's the problem?

this is easy (mostly)

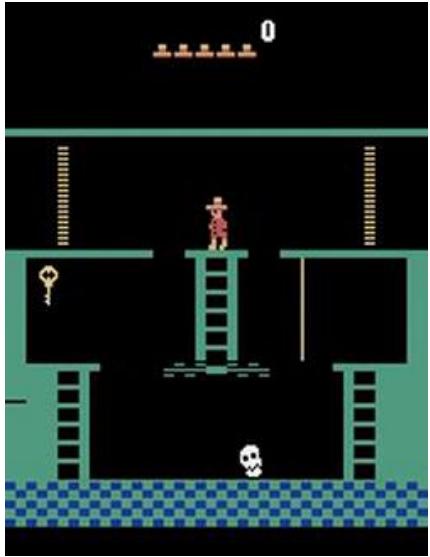


this is impossible



Why?

Montezuma's revenge



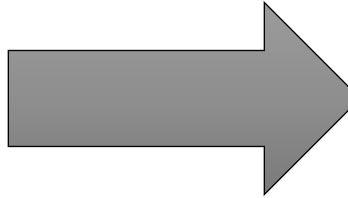
- Getting key = reward
- Opening door = reward
- Getting killed by skull = bad

Montezuma's revenge



- We know what to do because we **understand** what these sprites mean!
- Key: we know it opens doors!
- Ladders: we know we can climb them!
- Skull: we don't know what it does, but we know it can't be good!
- **Prior understanding of problem structure can help us solve complex tasks quickly!**

Can RL use the same prior knowledge as us?



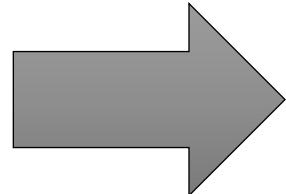
- If we've solved prior tasks, we might acquire useful knowledge for solving a new task
- How is the knowledge stored?
 - Q-function: tells us which actions or states are good
 - Policy: tells us which actions are potentially useful
 - some actions are never useful!
 - Models: what are the laws of physics that govern the world?
 - Features/hidden states: provide us with a good representation
 - Don't underestimate this!

Transfer learning terminology

transfer learning: using experience from one set of tasks for faster learning and better performance on a new task

in RL, **task = MDP!**

source domain



target domain



“shot”: number of attempts in the target domain

0-shot: just run a policy trained in the source domain

1-shot: try the task once

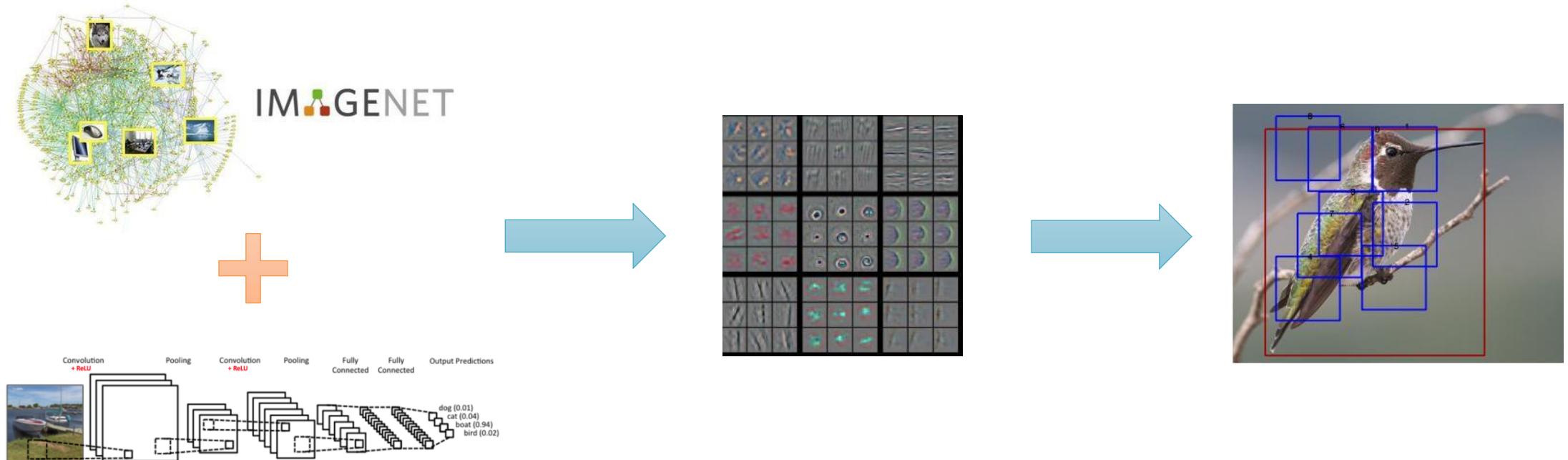
few shot: try the task a few times

How can we frame transfer learning problems?

1. Forward transfer: learn policies that transfer effectively
 - a) Train on source task, then run on target task (or finetune)
 - b) Relies on the tasks being quite similar!
2. Multi-task transfer: train on many tasks, transfer to a new task
 - a) Sharing representations and layers across tasks in multi-task learning
 - b) New task needs to be similar to the *distribution* of training tasks
3. Meta-learning: learn to *learn* on many tasks
 - a) Accounts for the fact that we'll be adapting to a new task during training!

Pretraining + Finetuning

The most popular transfer learning method in (supervised) deep learning!



What issues are we likely to face?

- **Domain shift:** representations learned in the source domain might not work well in the target domain

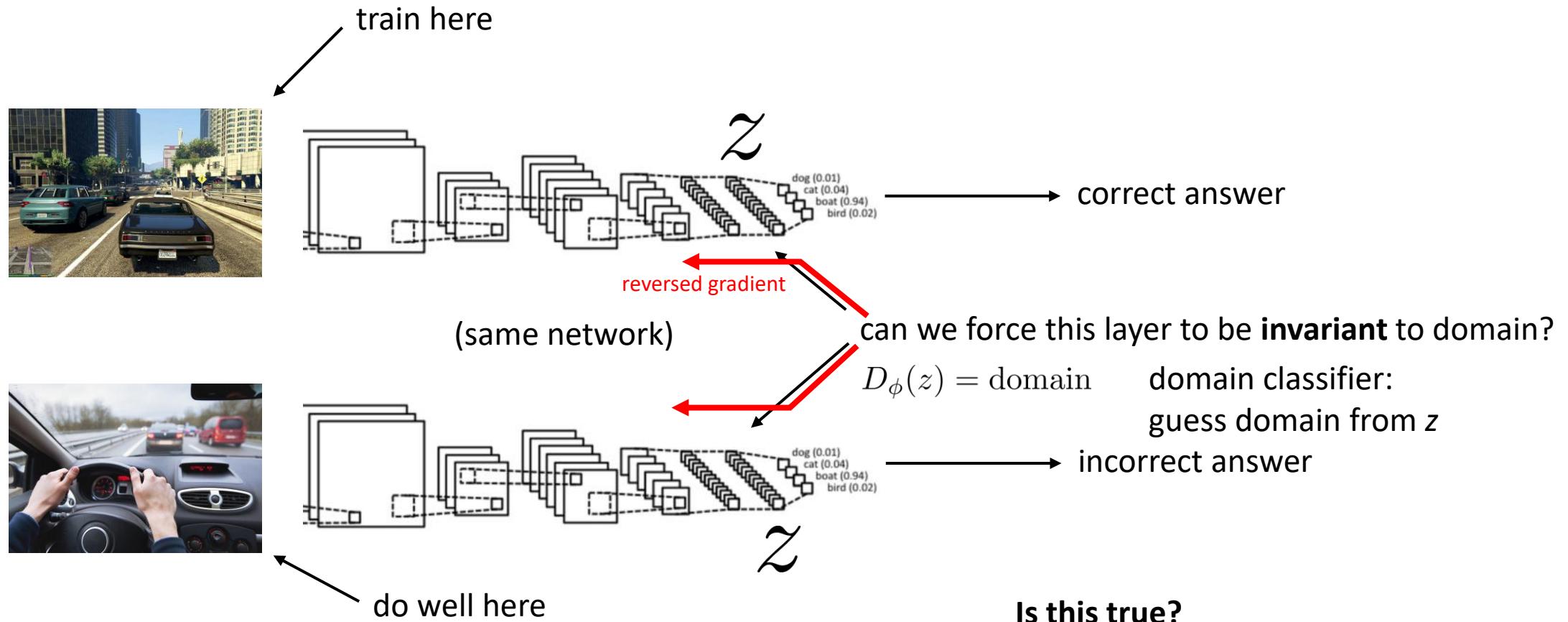


- **Difference in the MDP:** some things that are possible to do in the source domain are not possible to do in the target domain



- **Finetuning issues:** if pretraining & finetuning, the finetuning process may still need to explore, but optimal policy during finetuning may be deterministic!

Domain adaptation in computer vision



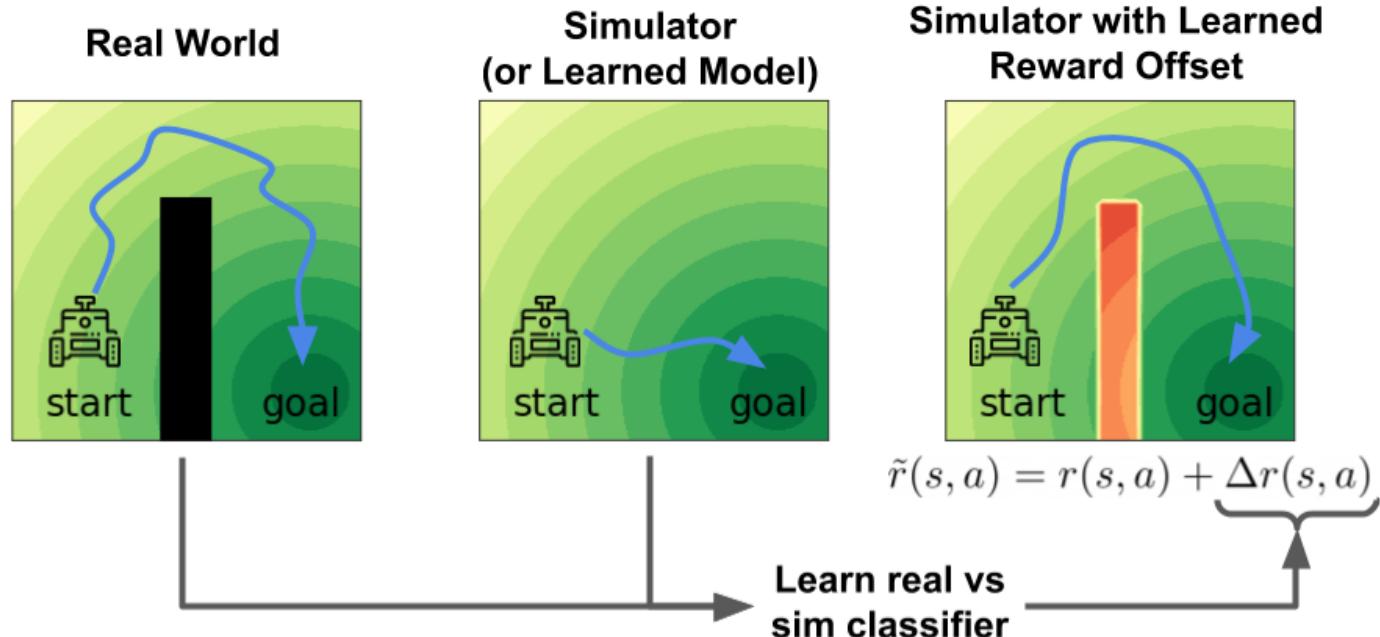
Invariance assumption: everything that is **different** between domains is **irrelevant**

formally:

$p(x)$ is different exists some $z = f(x)$ such that $p(y|z) = p(y|x)$, but $p(z)$ is same

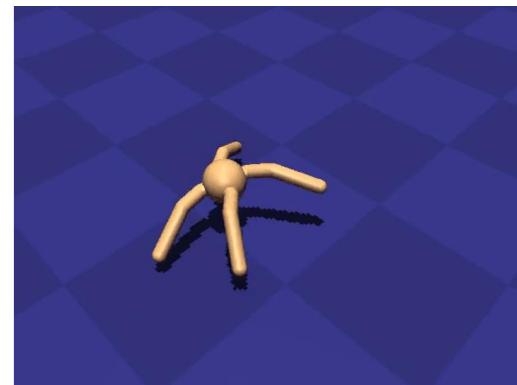
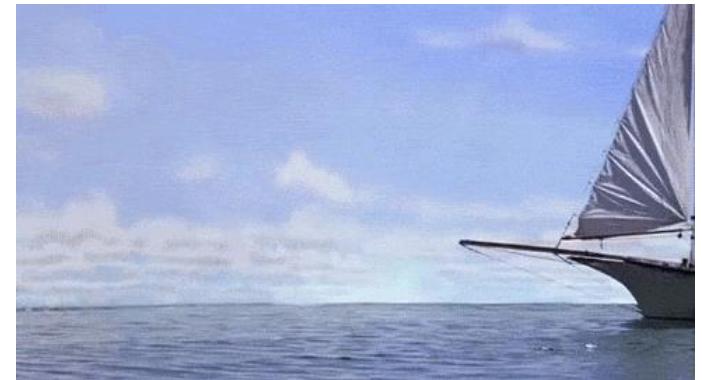
Domain adaptation in RL for dynamics?

Why is **invariance** not enough when the dynamics don't match?



$$\Delta r(s_t, a_t, s_{t+1}) = \log p_{\text{target}}(s_{t+1} | s_t, a_t) - \log p_{\text{source}}(s_{t+1} | s_t, a_t).$$

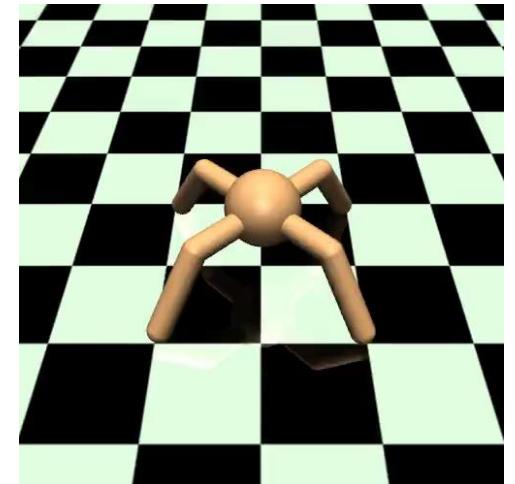
$$\begin{aligned}\Delta r(s_t, a_t, s_{t+1}) = & \log p(\text{target} | s_t, a_t, s_{t+1}) - \log p(\text{target} | s_t, a_t) \\ & - \log p(\text{source} | s_t, a_t, s_{t+1}) + \log p(\text{source} | s_t, a_t)\end{aligned}$$



When might this **not** work?

What if we can also finetune?

1. RL tasks are generally much less diverse
 - Features are less general
 - Policies & value functions become overly specialized
2. Optimal policies in fully observed MDPs are deterministic
 - Loss of exploration at convergence
 - Low-entropy policies adapt very slowly to new settings

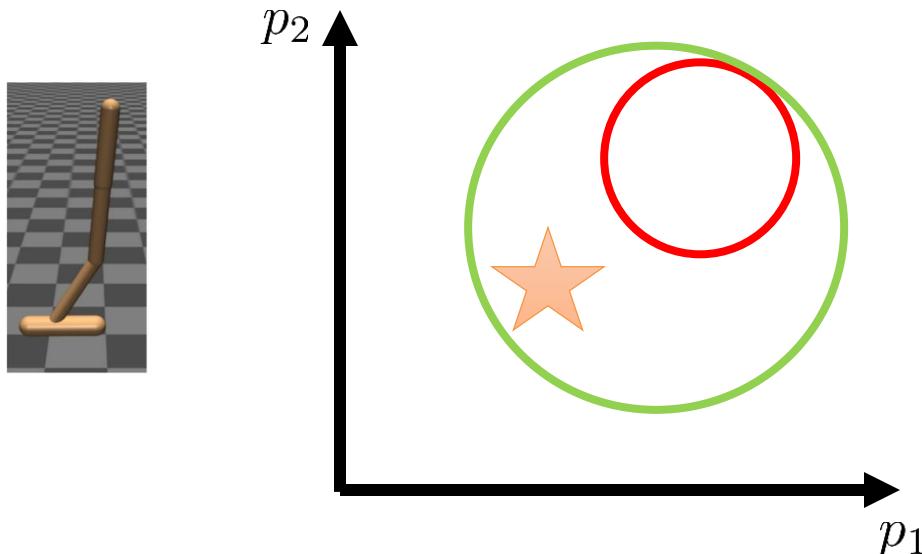


See “exploration 2” lecture on unsupervised skill discovery and “control as inference” lecture on MaxEnt RL methods!

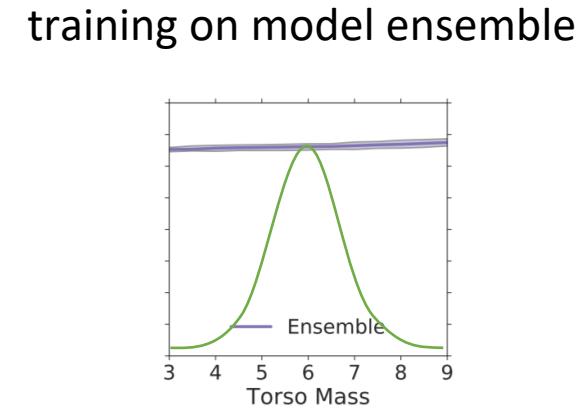
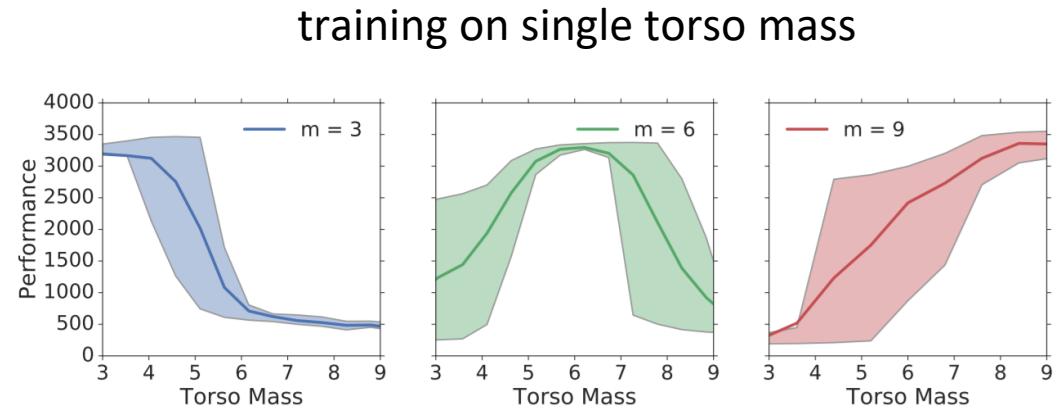
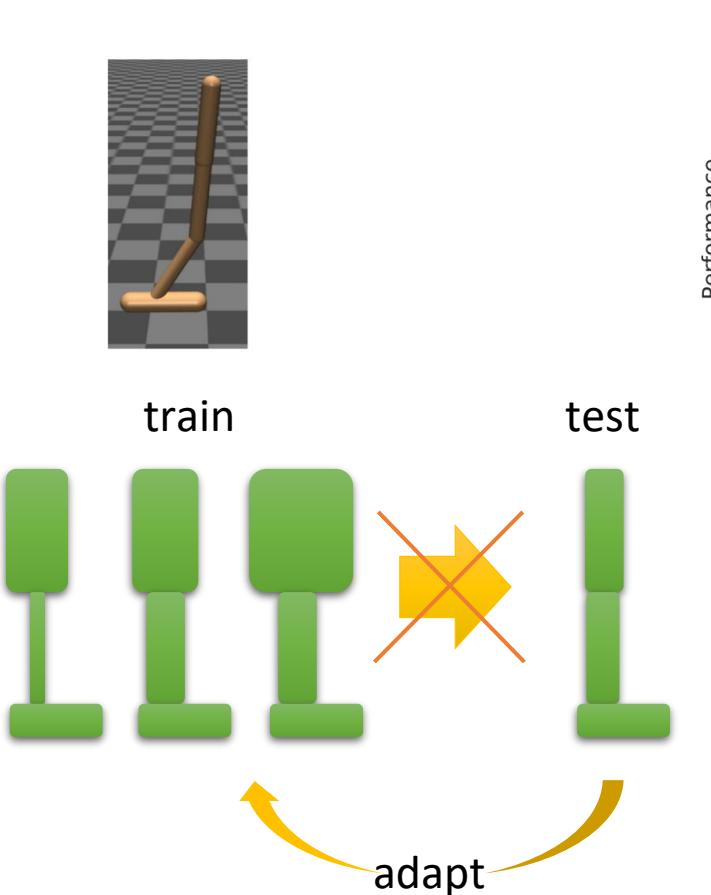
How to maximize forward transfer?

Basic intuition: the more **varied** the training domain is, the more likely we are to generalize in **zero shot** to a slightly different domain.

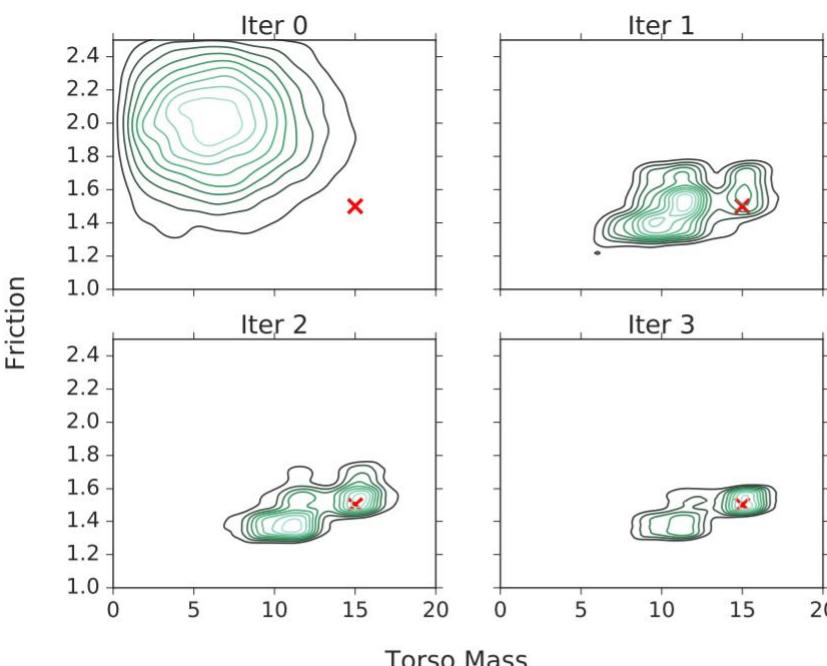
“Randomization” (dynamics/appearance/etc.): widely used for simulation to real world transfer (e.g., in robotics)



EPOpt: randomizing physical parameters



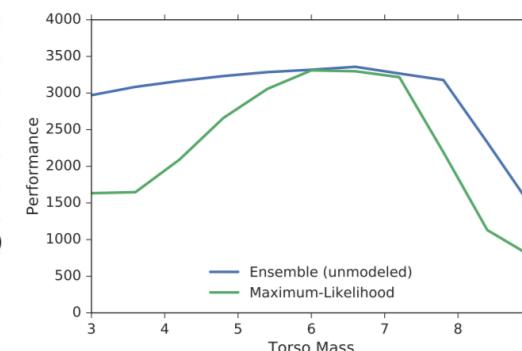
ensemble adaptation



unmodeled effects

Hopper	μ	σ	low	high
mass	6.0	1.5	3.0	9.0
ground friction	2.0	0.25	1.5	2.5
joint damping	2.5	1.0	1.0	4.0
armature	1.0	0.25	0.5	1.5

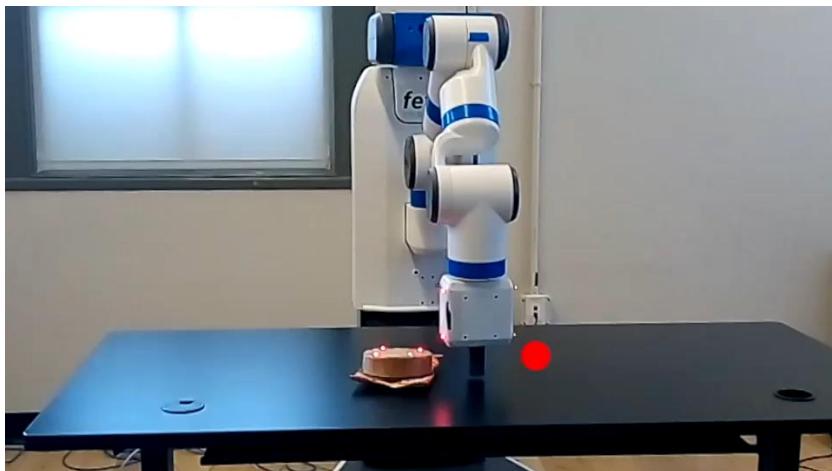
Half-Cheetah	μ	σ	low	high
mass	6.0	1.5	3.0	9.0
ground friction	0.5	0.1	0.3	0.7
joint damping	1.5	0.5	0.5	2.5
armature	0.125	0.04	0.05	0.2



More randomization!



Sadeghi et al., "CAD2RL: Real Single-Image Flight without a Single Real Image." 2016



Xue Bin Peng et al., "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization." 2018



Lee et al., "Learning Quadrupedal Locomotion over Challenging Terrain." 2020

Some suggested readings

Domain adaptation:

Tzeng, Hoffman, Zhang, Saenko, Darrell. **Deep Domain Confusion: Maximizing for Domain Invariance**. 2014.

Ganin, Ustinova, Ajakan, Germain, Larochelle, Laviolette, Marchand, Lempitsky. **Domain-Adversarial Training of Neural Networks**. 2015.

Tzeng*, Devin*, et al., **Adapting Visuomotor Representations with Weak Pairwise Constraints**. 2016.

Eysenbach et al., **Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers**. 2020.

Finetuning:

Finetuning via MaxEnt RL: Haarnoja*, Tang*, et al. (2017). **Reinforcement Learning with Deep Energy-Based Policies**.

Andreas et al. **Modular multitask reinforcement learning with policy sketches**. 2017.

Florensa et al. **Stochastic neural networks for hierarchical reinforcement learning**. 2017.

Kumar et al. **One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL**. 2020

Simulation to real world transfer:

Rajeswaran, et al. (2017). **EPOpt: Learning Robust Neural Network Policies Using Model Ensembles**.

Yu et al. (2017). **Preparing for the Unknown: Learning a Universal Policy with Online System Identification**.

Sadeghi & Levine. (2017). **CAD2RL: Real Single Image Flight without a Single Real Image**.

Tobin et al. (2017). **Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World**.

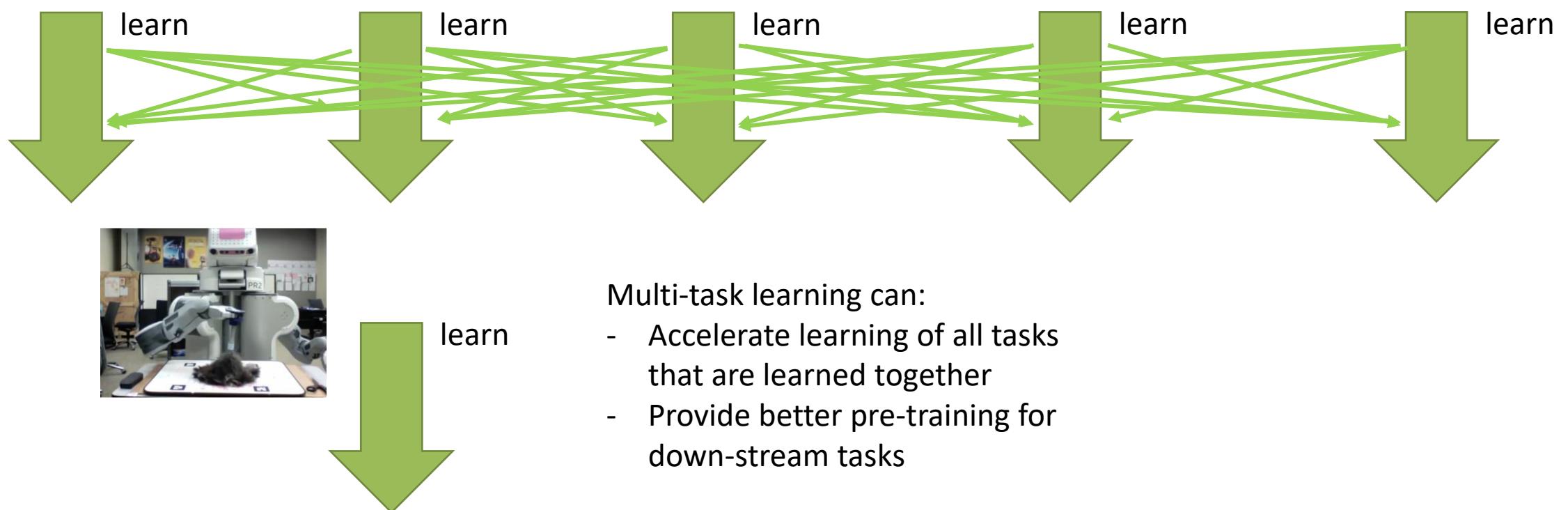
Tan et al. (2018). **Sim-to-Real: Learning Agile Locomotion For Quadruped Robots**.

...and many many others!

How can we frame transfer learning problems?

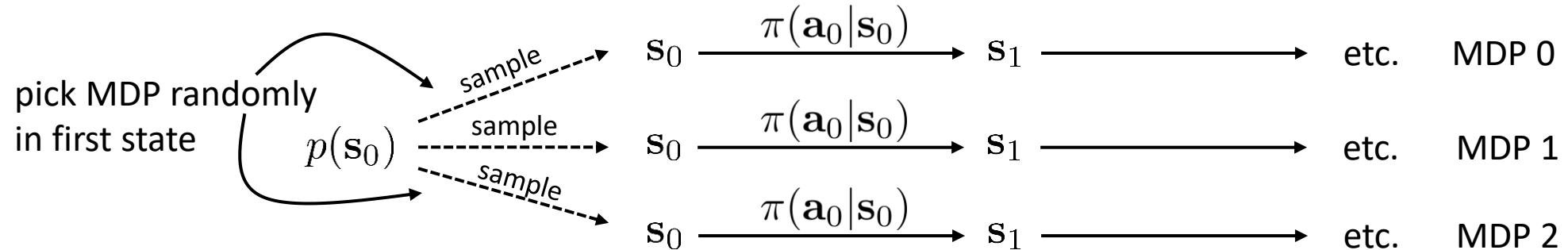
1. Forward transfer: learn policies that transfer effectively
 - a) Train on source task, then run on target task (or finetune)
 - b) Relies on the tasks being quite similar!
2. Multi-task transfer: train on many tasks, transfer to a new task
 - a) Sharing representations and layers across tasks in multi-task learning
 - b) New task needs to be similar to the *distribution* of training tasks
3. Meta-learning: learn to *learn* on many tasks
 - a) Accounts for the fact that we'll be adapting to a new task during training!

Can we learn **faster** by learning multiple tasks?



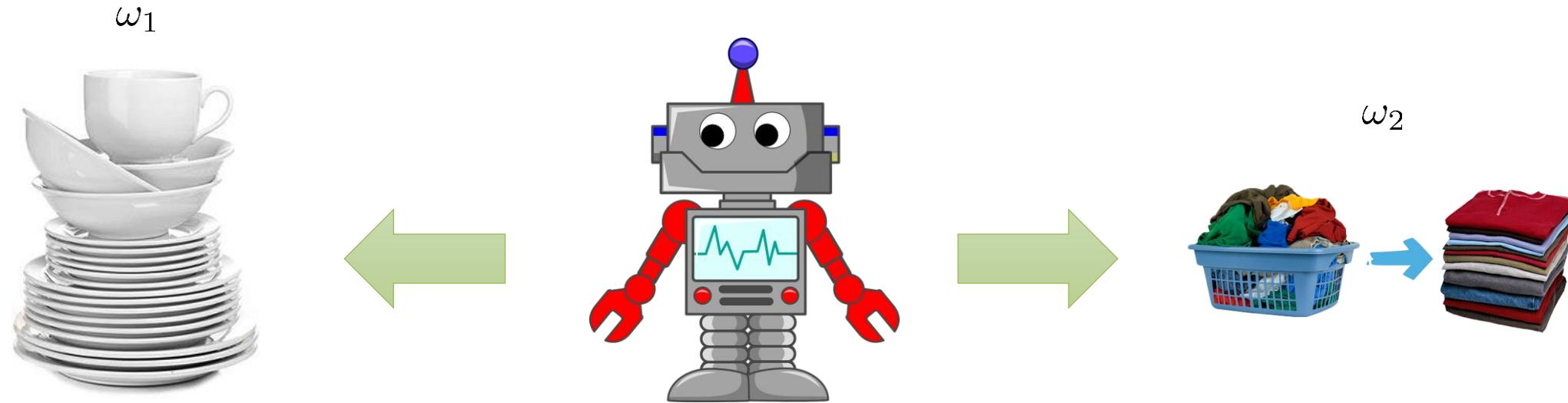
Can we solve multiple tasks at once?

Multi-task RL corresponds to single-task RL in a **joint MDP**



How does the model know what to do?

- What if the policy can do *multiple* things in the *same* environment?



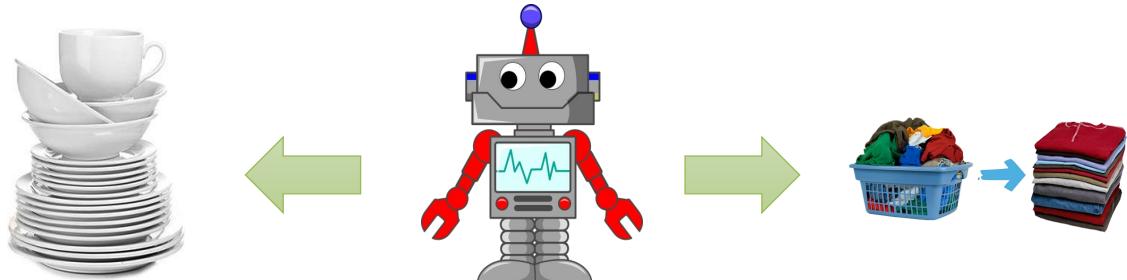
Contextual policies

standard policy: $\pi_\theta(\mathbf{a}|\mathbf{s})$

contextual policy: $\pi_\theta(\mathbf{a}|\mathbf{s}, \omega)$

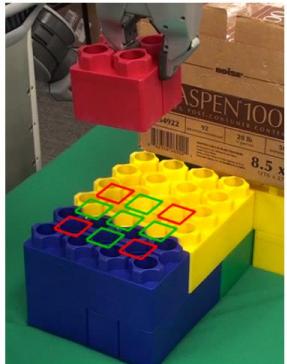


e.g., do dishes or laundry

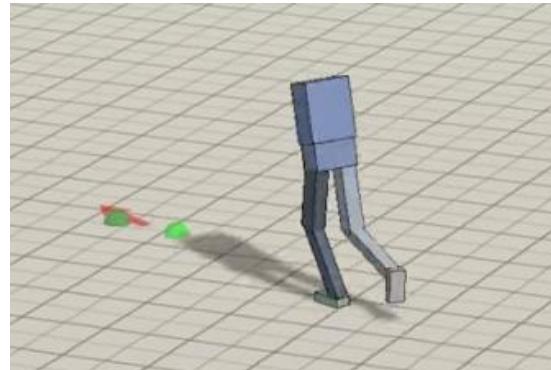


formally, simply defines augmented state space: $\tilde{\mathbf{s}} = \begin{bmatrix} \mathbf{s} \\ \omega \end{bmatrix}$

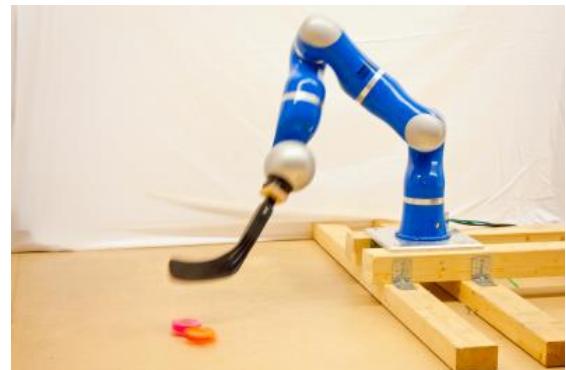
$$\tilde{\mathcal{S}} = \mathcal{S} \times \Omega$$



ω : stack location



ω : walking direction



ω : where to hit puck

Goal-conditioned policies

$$\pi_\theta(a|s, g)$$



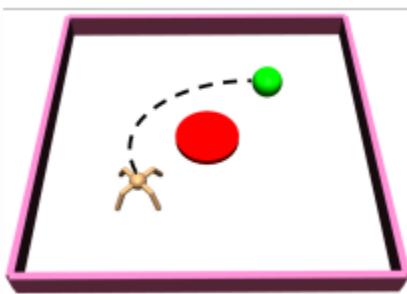
another state

$$r(s, a, g) = \delta(s = g)$$

$$r(s, a, g) = \delta(\|s - g\| \leq \varepsilon)$$

- Convenient: no need to manually define rewards for each task
- Can transfer in **zero shot** to a new task if it's another goal!

- Often hard to train in practice (see references)
- Not all tasks are goal reaching tasks!



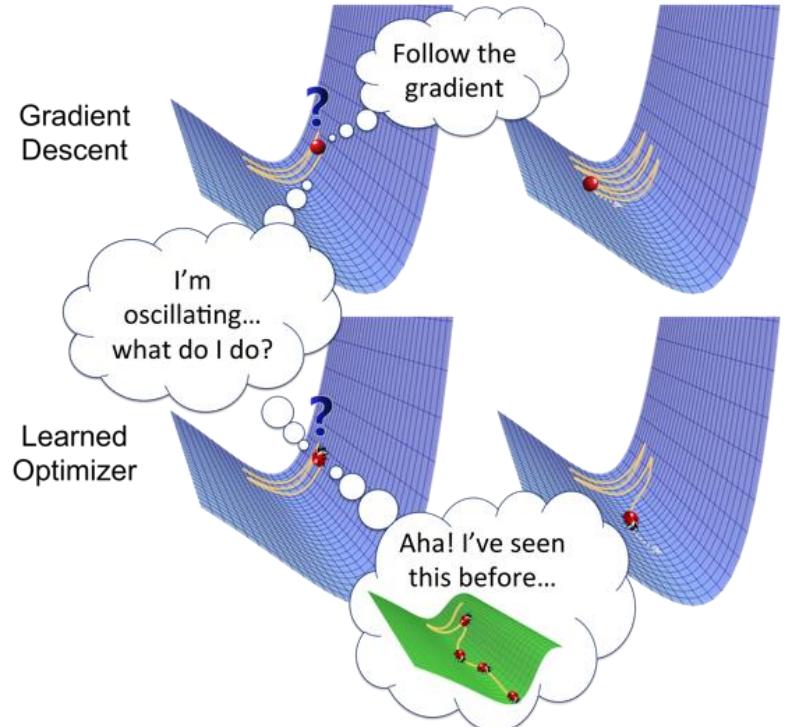
A few relevant papers:

- Kaelbling. **Learning to achieve goals.**
- Schaul et al. **Universal value function approximators.**
- Andrychowicz et al. **Hindsight experience replay.**
- Eysenbach et al. **C-learning: Learning to achieve goals via recursive classification.**

Meta-Learning

What is meta-learning?

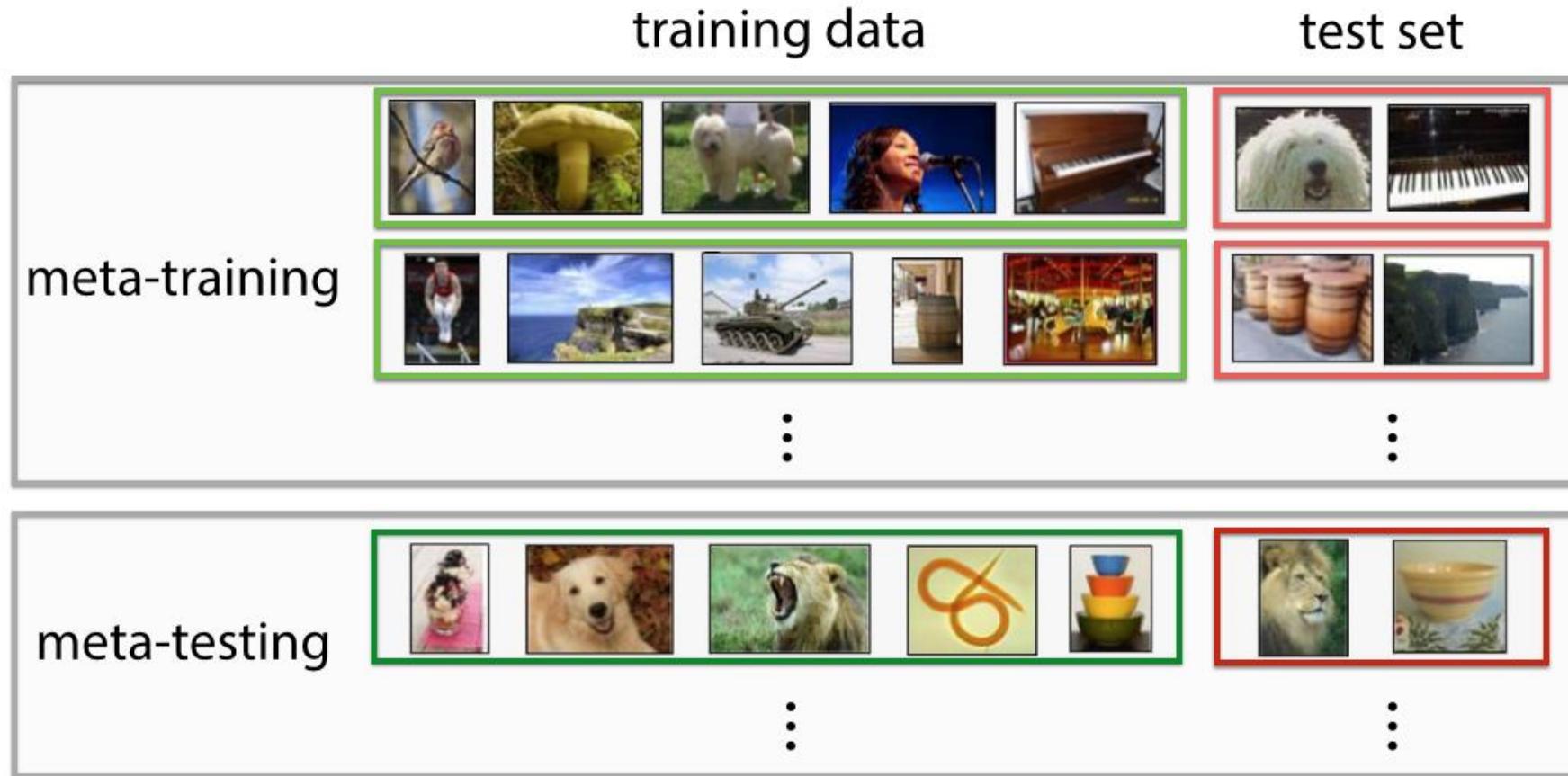
- If you've learned 100 tasks already, can you figure out how to *learn* more efficiently?
 - Now having multiple tasks is a huge advantage!
- Meta-learning = *learning to learn*
- In practice, very closely related to multi-task learning
- Many formulations
 - Learning an optimizer
 - Learning an RNN that ingests experience
 - Learning a representation



Why is meta-learning a good idea?

- Deep reinforcement learning, especially model-free, requires a huge number of samples
- If we can *meta-learn* a faster reinforcement learner, we can learn new tasks efficiently!
- What can a *meta-learned* learner do differently?
 - Explore more intelligently
 - Avoid trying actions that are known to be useless
 - Acquire the right features more quickly

Meta-learning with supervised learning

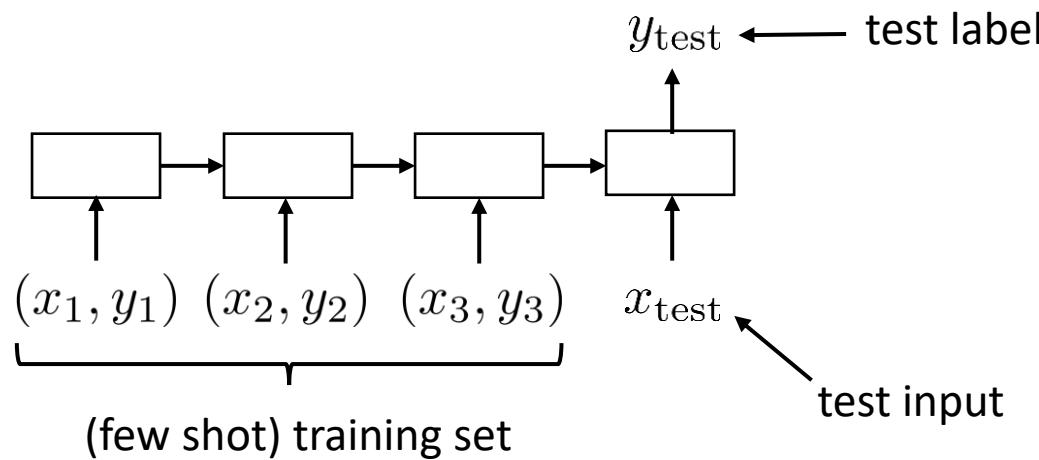


Meta-learning with supervised learning



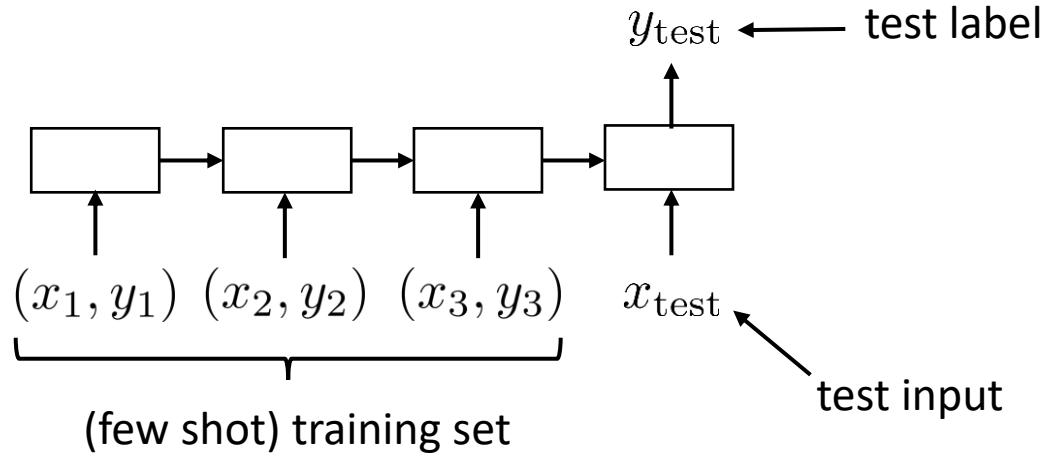
supervised learning: $f(x) \rightarrow y$
↑
input (e.g., image) output (e.g., label)

supervised meta-learning: $f(\mathcal{D}^{\text{tr}}, x) \rightarrow y$
↑
training set



- How to read in training set?
 - Many options, RNNs can work
 - More on this later

What is being “learned”?



supervised meta-learning: $f(\mathcal{D}^{\text{tr}}, x) \rightarrow y$

“Generic” learning:

$$\begin{aligned}\theta^* &= \arg \min_{\theta} \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}) \\ &= f_{\text{learn}}(\mathcal{D}^{\text{tr}})\end{aligned}$$

“Generic” meta-learning:

$$\theta^* = \arg \min_{\theta} \sum_{i=1}^n \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_{\theta}(\mathcal{D}_i^{\text{tr}})$

What is being “learned”?

“Generic” learning:

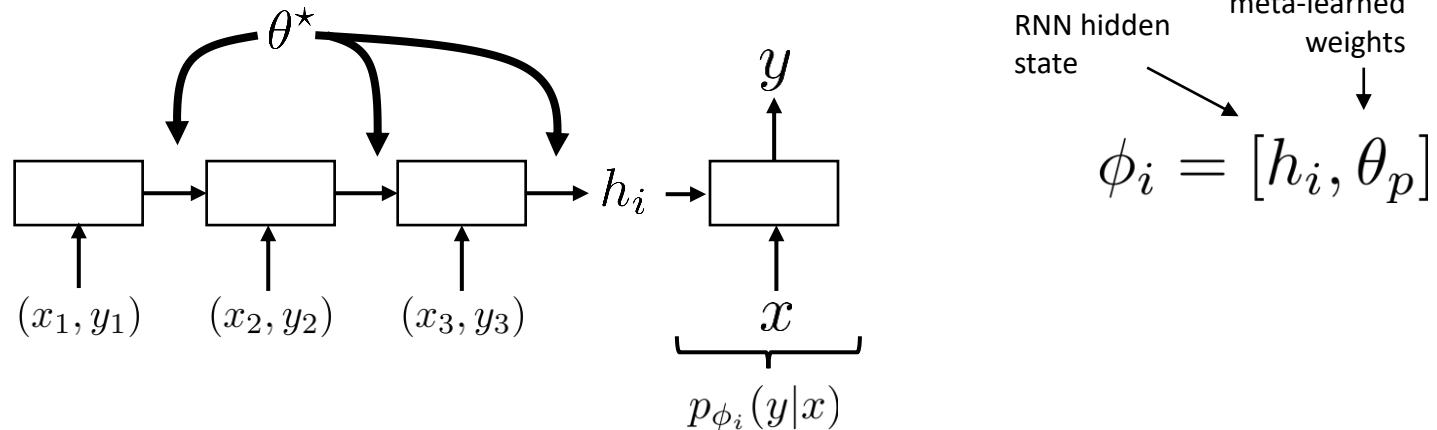
$$\theta^* = \arg \min_{\theta} \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

$$= f_{\text{learn}}(\mathcal{D}^{\text{tr}})$$

“Generic” meta-learning:

$$\theta^* = \arg \min_{\theta} \sum_{i=1}^n \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_{\theta}(\mathcal{D}_i^{\text{tr}})$$



Meta Reinforcement Learning

The meta reinforcement learning problem

“Generic” learning:

$$\begin{aligned}\theta^* &= \arg \min_{\theta} \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}) \\ &= f_{\text{learn}}(\mathcal{D}^{\text{tr}})\end{aligned}$$

Reinforcement learning:

$$\begin{aligned}\theta^* &= \arg \max_{\theta} E_{\pi_{\theta}(\tau)}[R(\tau)] \\ &= f_{\text{RL}}(\mathcal{M}) \quad \mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}\end{aligned}$$

MDP

“Generic” meta-learning:

$$\begin{aligned}\theta^* &= \arg \min_{\theta} \sum_{i=1}^n \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}}) \\ \text{where } \phi_i &= f_{\theta}(\mathcal{D}_i^{\text{tr}})\end{aligned}$$

Meta-reinforcement learning:

$$\begin{aligned}\theta^* &= \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)] \\ \text{where } \phi_i &= f_{\theta}(\mathcal{M}_i)\end{aligned}$$

MDP for task i

The meta reinforcement learning problem

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_{\theta}(\mathcal{M}_i)$

assumption: $\mathcal{M}_i \sim p(\mathcal{M})$

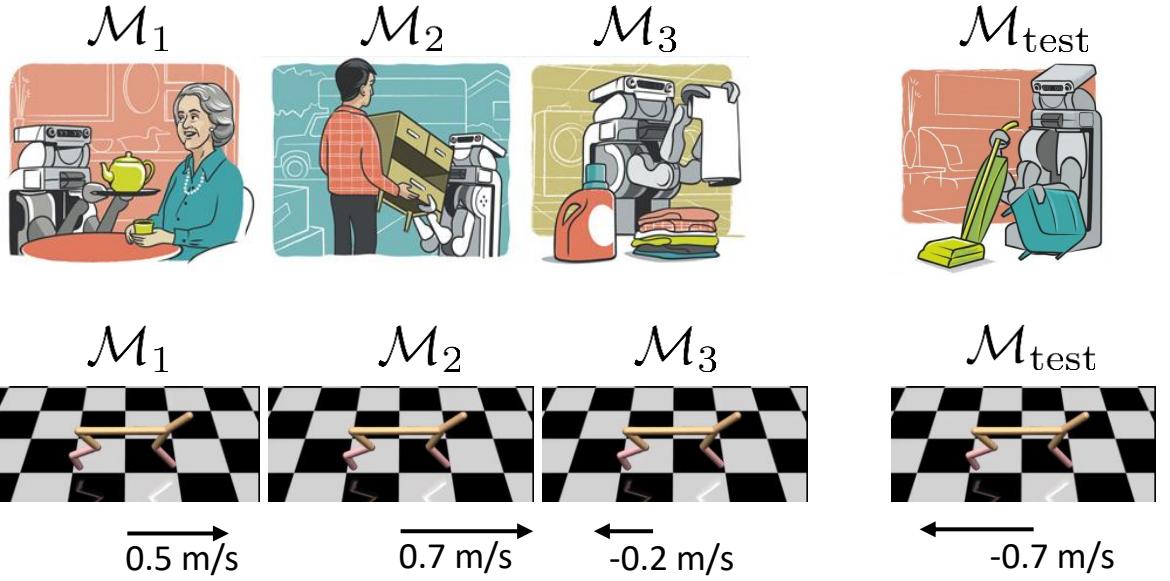
meta test-time:

sample $\mathcal{M}_{\text{test}} \sim p(\mathcal{M})$, get $\phi_i = f_{\theta}(\mathcal{M}_{\text{test}})$

$\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$

↑
meta-training MDPs

Some examples:



Contextual policies and meta-learning

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_{\theta}(\mathcal{M}_i)$



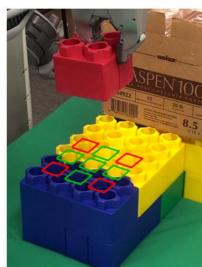
$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\theta}}[R(\tau)]$$

$$\pi_{\theta}(a_t | s_t, \underbrace{s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}}_{\text{context}})$$

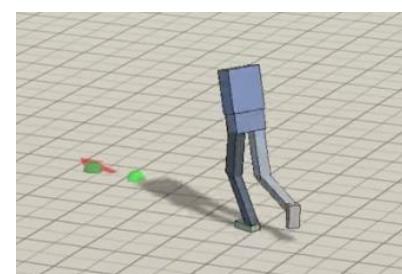
context used to infer whatever we need to solve \mathcal{M}_i
i.e., z_t or ϕ_i (which are really the same thing)

in meta-RL, the *context* is inferred from experience from \mathcal{M}_i

in multi-task RL, the context is typically given



ϕ : stack location



ϕ : walking direction



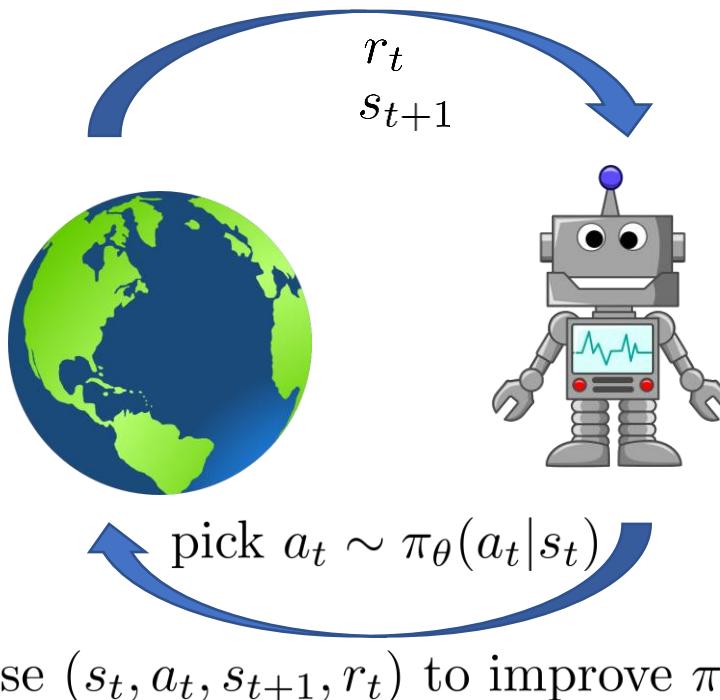
ϕ : where to hit puck



Meta-RL with recurrent policies

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_{\theta}(\mathcal{M}_i)$



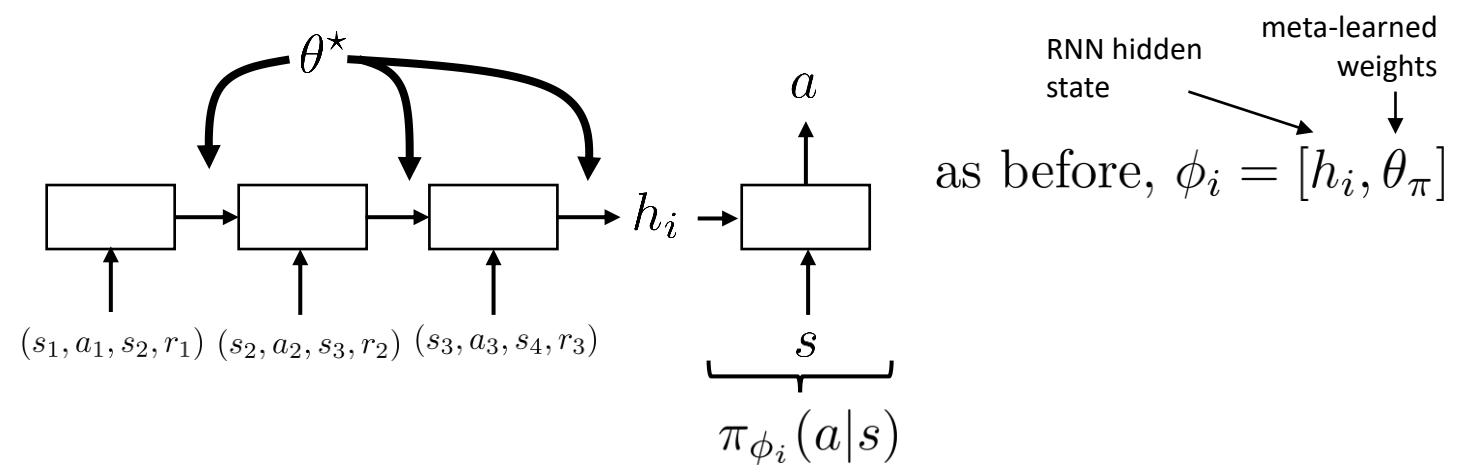
main question: how to implement $f_{\theta}(\mathcal{M}_i)$?

what should $f_{\theta}(\mathcal{M}_i)$ do?

1. improve policy with experience from \mathcal{M}_i

$\{(s_1, a_1, s_2, r_1), \dots, (s_T, a_T, s_{T+1}, r_T)\}$

2. (new in RL): choose how to interact, i.e. choose a_t
meta-RL must also *choose* how to *explore*!



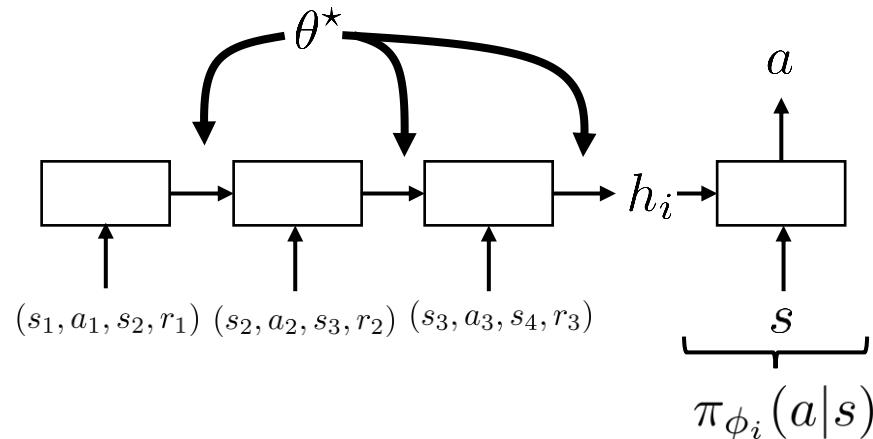
Meta-RL with recurrent policies

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

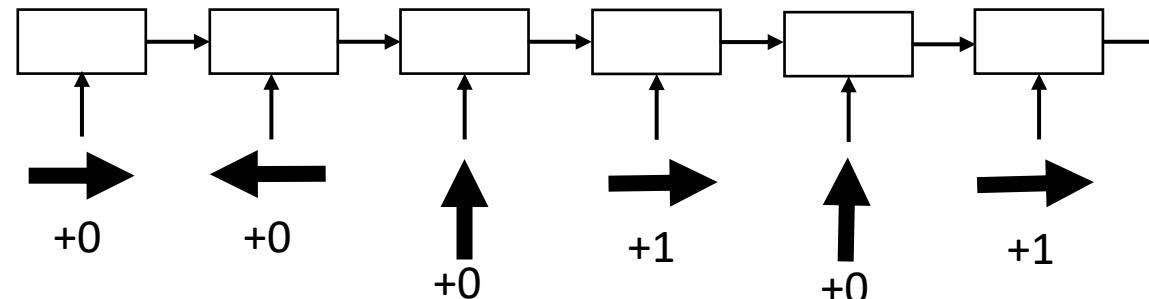
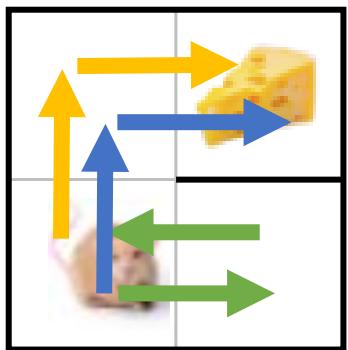
where $\phi_i = f_{\theta}(\mathcal{M}_i)$

so... we just train an RNN policy?

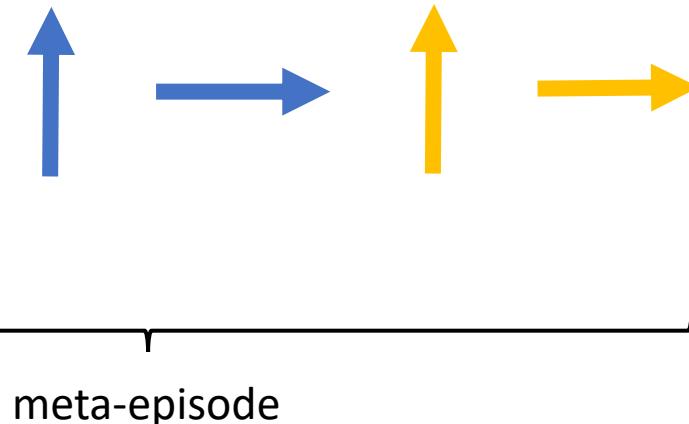
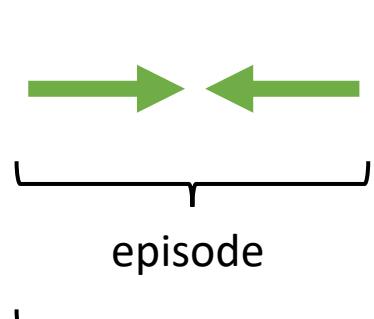
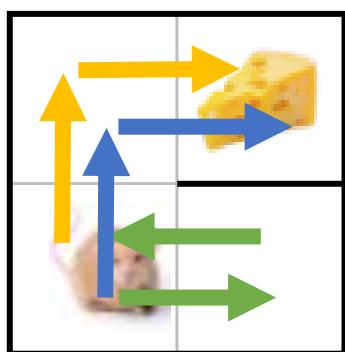
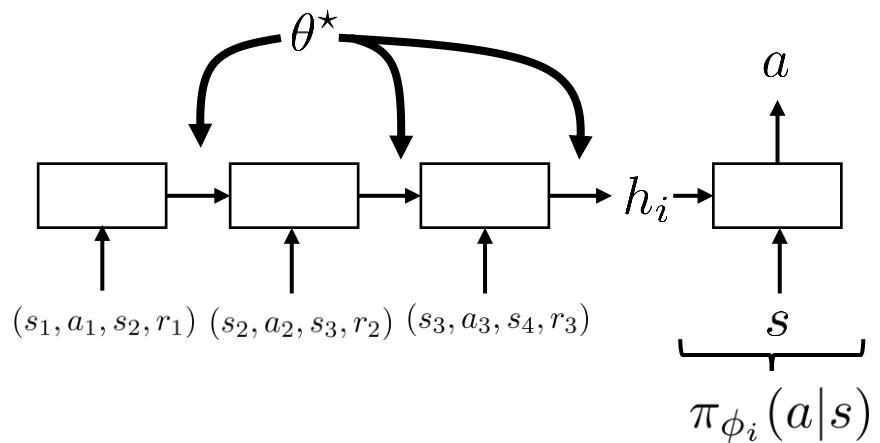
yes!



crucially, RNN hidden state is **not reset between episodes!**



Why recurrent policies *learn to explore*



1. improve policy with experience from \mathcal{M}_i
 $\{(s_1, a_1, s_2, r_1), \dots, (s_T, a_T, s_{T+1}, r_T)\}$
2. (new in RL): choose how to interact, i.e. choose a_t
meta-RL must also *choose* how to *explore*!

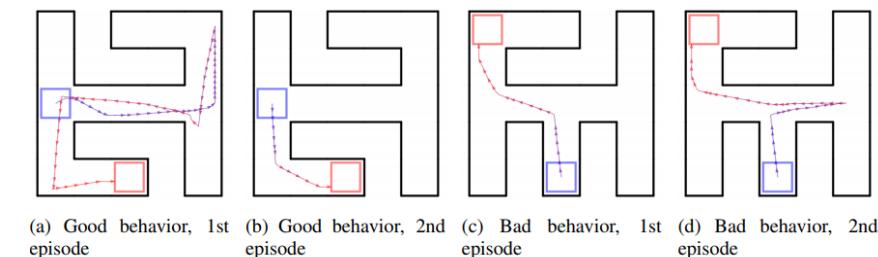
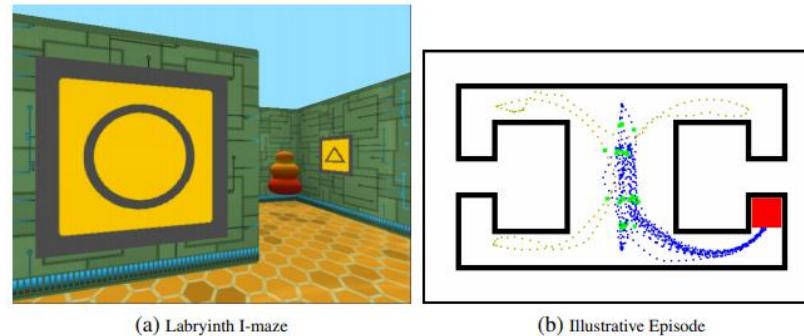
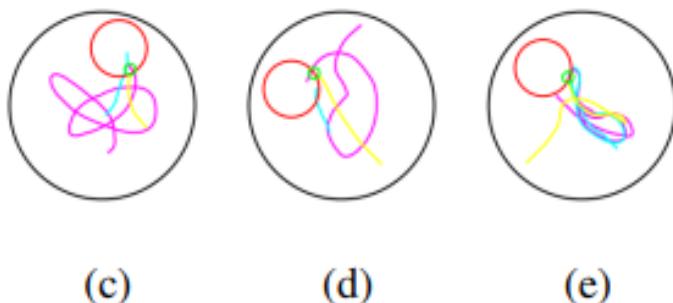
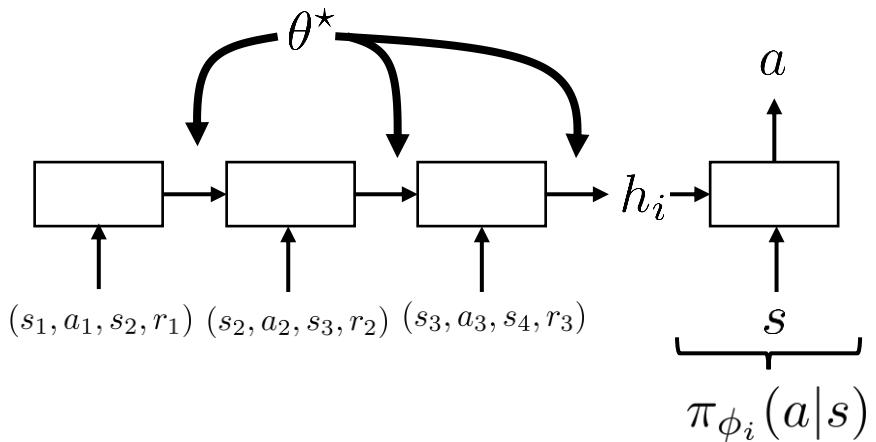
$$\theta^* = \arg \max_{\theta} E_{\pi_{\theta}} \left[\sum_{t=0}^T r(s_t, a_t) \right]$$

optimizing total reward over the entire **meta**-episode with RNN policy **automatically** learns to explore!

Meta-RL with recurrent policies

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_{\theta}(\mathcal{M}_i)$

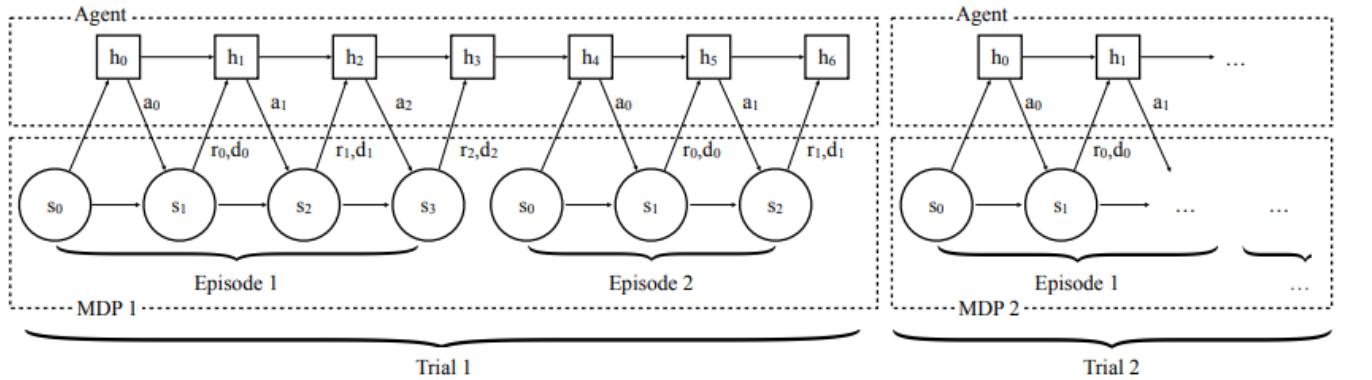


Heess, Hunt, Lillicrap, Silver. **Memory-based control with recurrent neural networks.** 2015.

Wang, Kurth-Nelson, Tirumala, Soyer, Leibo, Munos, Blundell, Kumaran, Botvinick. **Learning to Reinforcement Learning.** 2016.

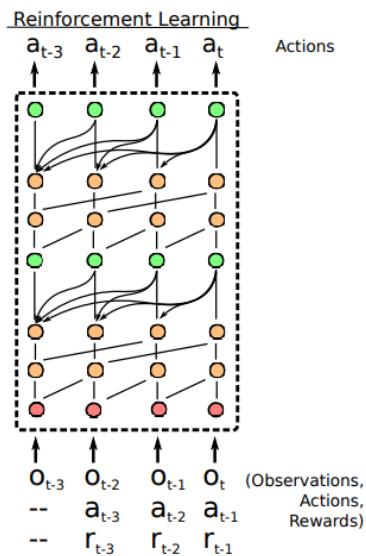
Duan, Schulman, Chen, Bartlett, Sutskever, Abbeel. **RL2: Fast Reinforcement Learning via Slow Reinforcement Learning.** 2016.

Architectures for meta-RL



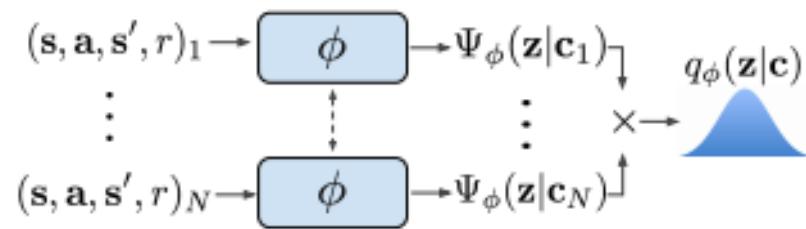
standard RNN (LSTM) architecture

Duan, Schulman, Chen, Bartlett, Sutskever, Abbeel. **RL2: Fast Reinforcement Learning via Slow Reinforcement Learning.** 2016.



attention + temporal convolution

Mishra, Rohaninejad, Chen, Abbeel. **A Simple Neural Attentive Meta-Learner.**

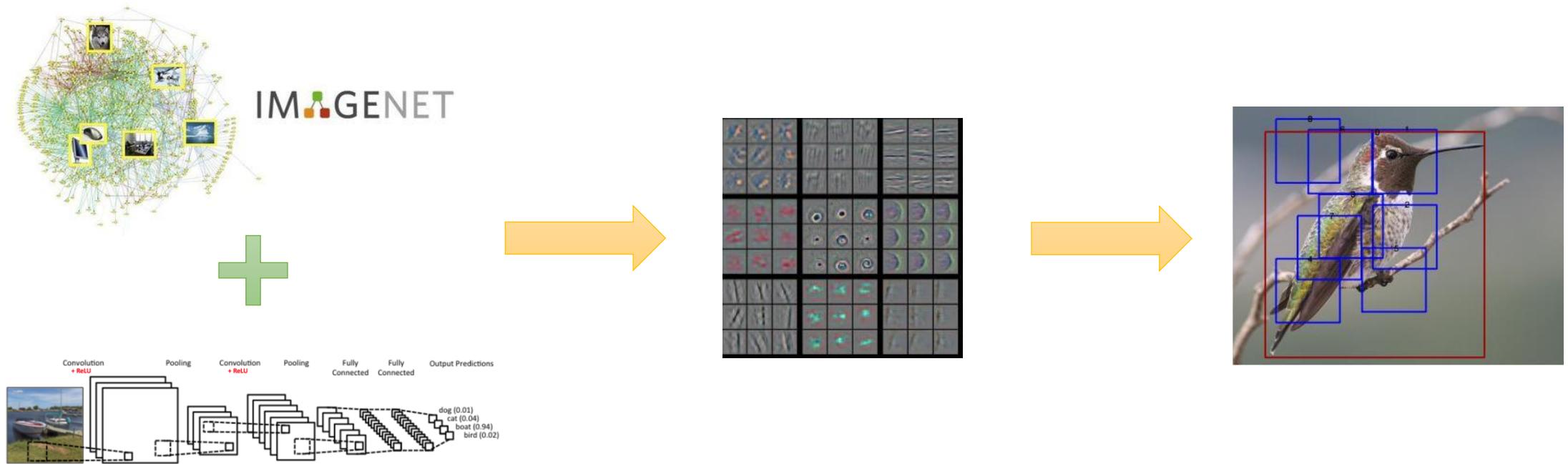


parallel permutation-invariant context encoder

Rakelly*, Zhou*, Quillen, Finn, Levine. **Efficient Off-Policy Meta-Reinforcement learning via Probabilistic Context Variables.**

Gradient-Based Meta-Learning

Back to representations...



is pretraining a *type* of meta-learning?
better features = faster learning of new task!

Meta-RL as an optimization problem

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

1. improve policy with experience from \mathcal{M}_i
 $\{(s_1, a_1, s_2, r_1), \dots, (s_T, a_T, s_{T+1}, r_T)\}$

where $\phi_i = f_{\theta}(\mathcal{M}_i)$

what if $f_{\theta}(\mathcal{M}_i)$ is *itself* an RL algorithm?

standard RL:

$$f_{\theta}(\mathcal{M}_i) = \theta + \alpha \underbrace{\nabla_{\theta} J_i(\theta)}$$

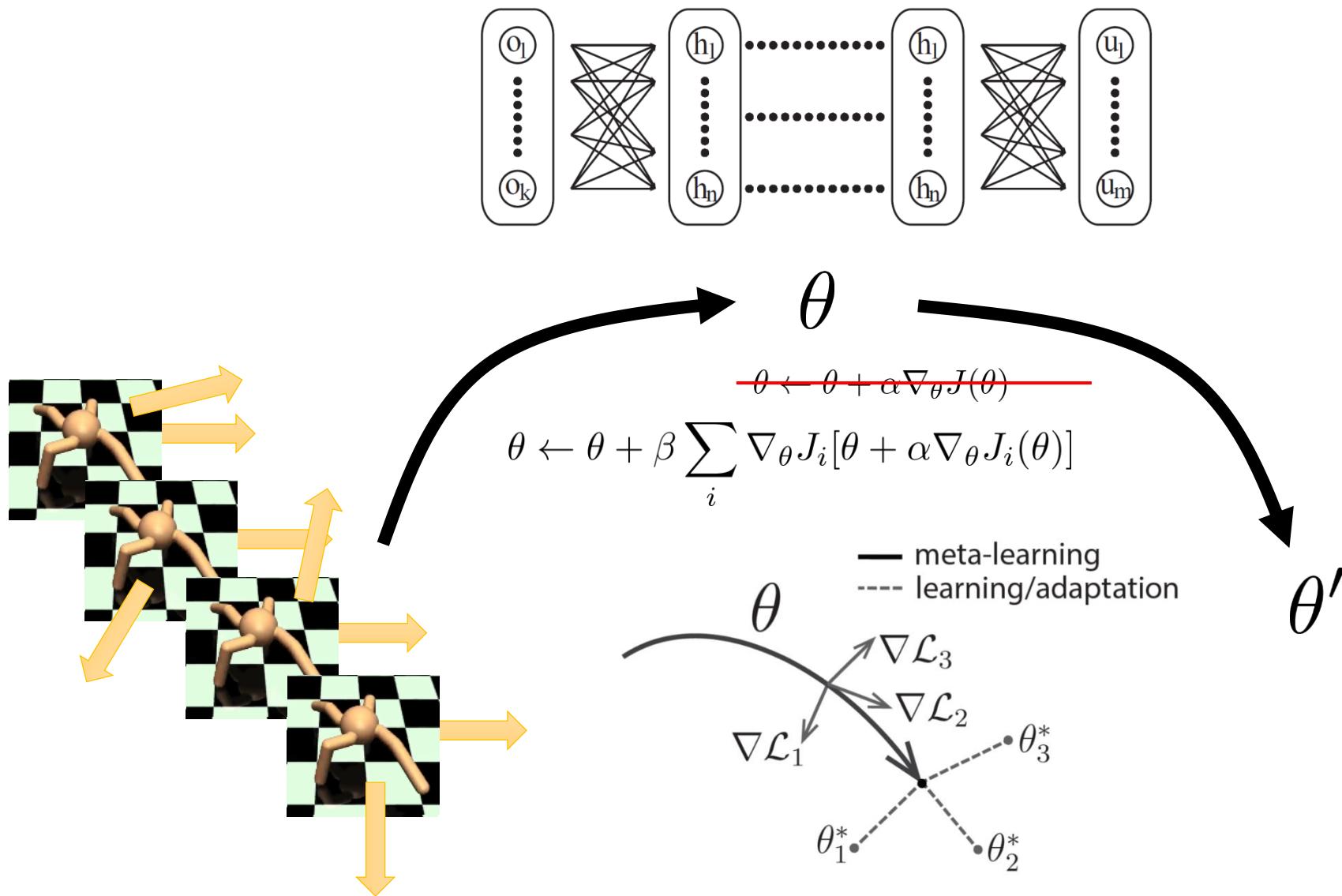
requires interacting with \mathcal{M}_i
to estimate $\nabla_{\theta} E_{\pi_{\theta}}[R(\tau)]$

$$\theta^* = \arg \max_{\theta} \underbrace{E_{\pi_{\theta}}[R(\tau)]}_{J(\theta)}$$

$$\theta^{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta^k} J(\theta^k)$$

model-agnostic meta-learning (MAML)

MAML for RL in pictures



What did we just do??

supervised learning: $f(x) \rightarrow y$

supervised meta-learning: $f(\mathcal{D}^{\text{tr}}, x) \rightarrow y$

model-agnostic meta-learning: $f_{\text{MAML}}(\mathcal{D}^{\text{tr}}, x) \rightarrow y$

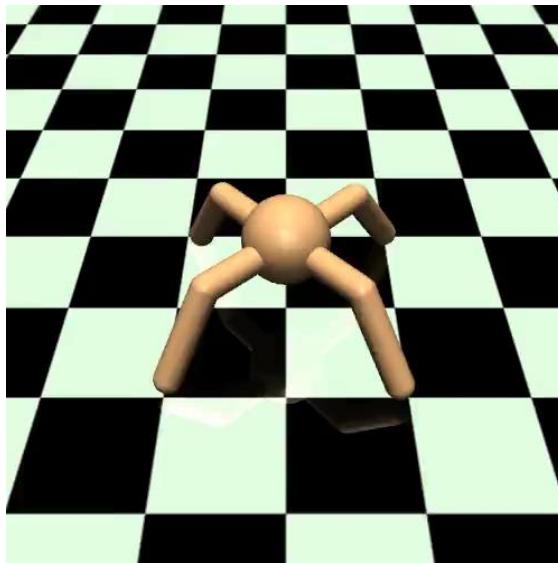
$$f_{\text{MAML}}(\mathcal{D}^{\text{tr}}, x) = f_{\theta'}(x)$$

$$\theta' = \theta - \alpha \sum_{(x,y) \in \mathcal{D}^{\text{tr}}} \nabla_{\theta} \mathcal{L}(f_{\theta}(x), y)$$

Just another computation graph...
Can implement with any autodiff
package (e.g., TensorFlow)
But has favorable inductive bias...

MAML for RL in videos

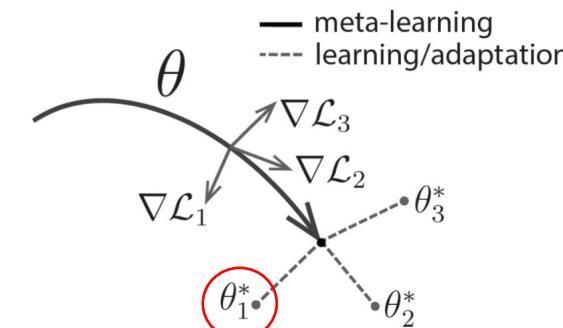
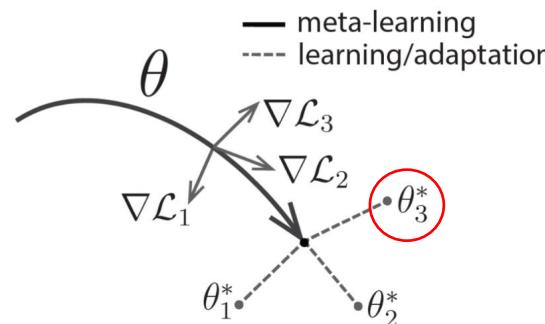
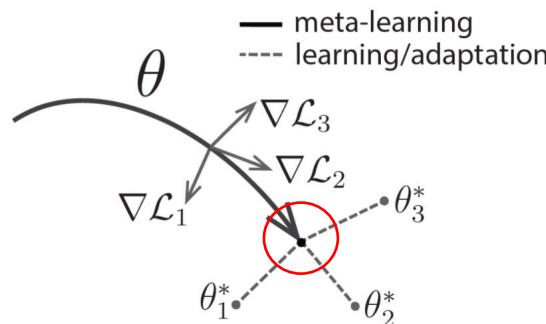
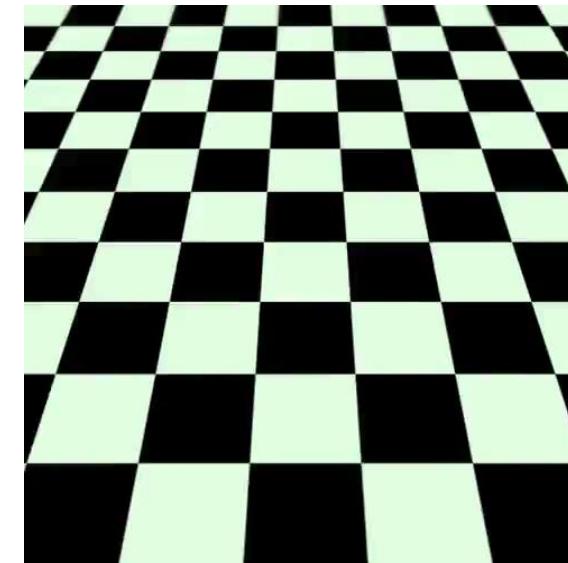
after MAML training



after 1 gradient step
(forward reward)



after 1 gradient step
(backward reward)



More on MAML/gradient-based meta-learning for RL

MAML meta-policy gradient estimators:

- Finn, Abbeel, Levine. **Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.**
- Foerster, Farquhar, Al-Shedivat, Rocktaschel, Xing, Whiteson. **DiCE: The Infinitely Differentiable Monte Carlo Estimator.**
- Rothfuss, Lee, Clavera, Asfour, Abbeel. **ProMP: Proximal Meta-Policy Search.**

Improving exploration:

- Gupta, Mendonca, Liu, Abbeel, Levine. **Meta-Reinforcement Learning of Structured Exploration Strategies.**
- Stadie*, Yang*, Houthooft, Chen, Duan, Wu, Abbeel, Sutskever. **Some Considerations on Learning to Explore via Meta-Reinforcement Learning.**

Hybrid algorithms (not necessarily gradient-based):

- Houthooft, Chen, Isola, Stadie, Wolski, Ho, Abbeel. **Evolved Policy Gradients.**
- Fernando, Sygnowski, Osindero, Wang, Schaul, Teplyashin, Sprechmann, Pirtzel, Rusu. **Meta-Learning by the Baldwin Effect.**

Meta-RL as a POMDP

Meta-RL as... partially observed RL?

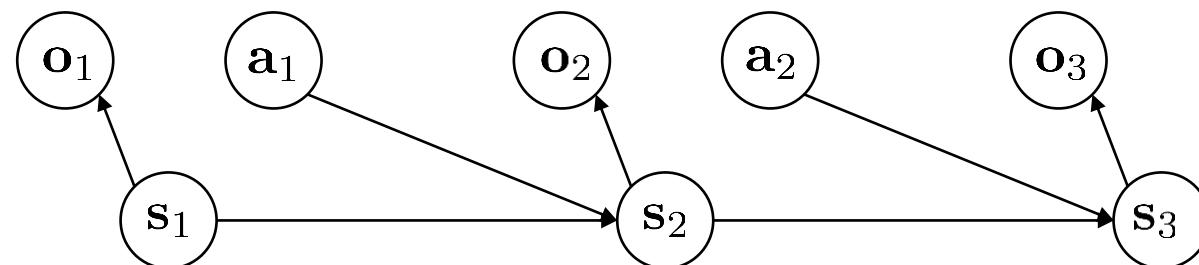
$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{E}, r\}$$

\mathcal{O} – observation space

observations $o \in \mathcal{O}$ (discrete or continuous)

\mathcal{E} – emission probability $p(o_t|s_t)$

policy must act on observations o_t !



$\pi_\theta(a|o)$

typically requires *either*:

explicit state estimation, i.e. to estimate $p(s_t|o_{1:t})$

policies with memory

Meta-RL as... partially observed RL?

$$\pi_{\theta}(a|\overbrace{s, z}^{\tilde{s}})$$

encapsulates information policy
needs to solve current task

learning a task = inferring z

from *context* $(s_1, a_1, s_2, r_1), (s_2, a_2, s_3, r_2), \dots$

this is just a POMDP!

before: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}$

now: $\tilde{\mathcal{M}} = \{\tilde{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{O}}, \tilde{\mathcal{P}}, \mathcal{E}, r\}$

$$\tilde{\mathcal{S}} = \mathcal{S} \times \mathcal{Z} \quad \tilde{s} = (s, z)$$

$$\tilde{\mathcal{O}} = \mathcal{S} \quad \tilde{o} = s$$

key idea: solving the POMDP $\tilde{\mathcal{M}}$ is equivalent to meta-learning!

Meta-RL as... partially observed RL?

$$\pi_\theta(a|s, z)$$

encapsulates information policy
needs to solve current task

learning a task = inferring z
from *context* $(s_1, a_1, s_2, r_1), (s_2, a_2, s_3, r_2), \dots$

exploring via posterior sampling with latent context

- 
1. sample $z \sim \hat{p}(z_t|s_{1:t}, a_{1:t}, r_{1:t})$ ← some approximate posterior
(e.g., variational)
 2. act according to $\pi_\theta(a|s, z)$ to collect more data
← act as though z was correct!

this is just a POMDP!

typically requires *either*:

explicit state estimation, i.e. to estimate $p(s_t|o_{1:t})$

policies with memory

need to estimate $p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$

this is *not* optimal!
why?

but it's pretty good,
both in theory and in
practice!

Variational inference for meta-RL

policy: $\pi_\theta(a_t|s_t, z_t)$

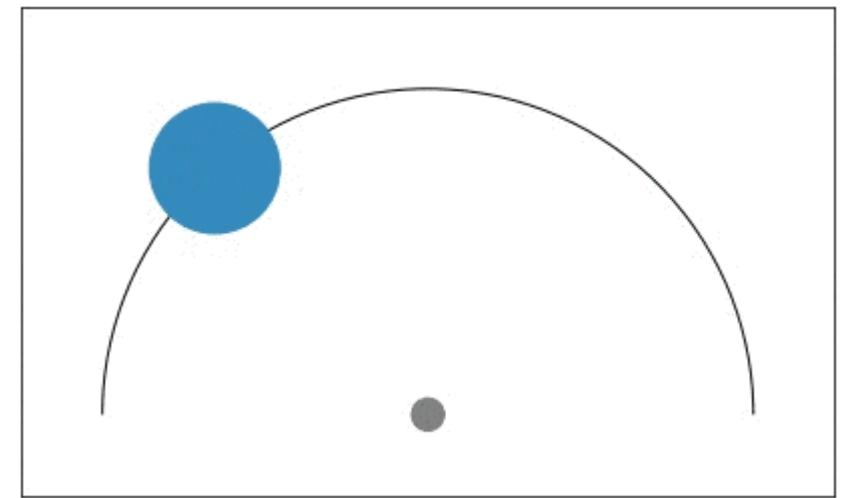
inference network: $q_\phi(z_t|s_1, a_1, r_1, \dots, s_t, a_t, r_t)$

$$(\theta, \phi) = \arg \max_{\theta, \phi} \frac{1}{N} \sum_{i=1}^n E_{z \sim q_\phi, \tau \sim \pi_\theta} [R_i(\tau) - D_{\text{KL}}(q(z|\dots)\|p(z))]$$

maximize *post-update* reward
(same as standard meta-RL)

stay close to prior

$$z_t \sim q_\phi(z_t|s_1, a_1, r_1, \dots, s_t, a_t, r_t)$$



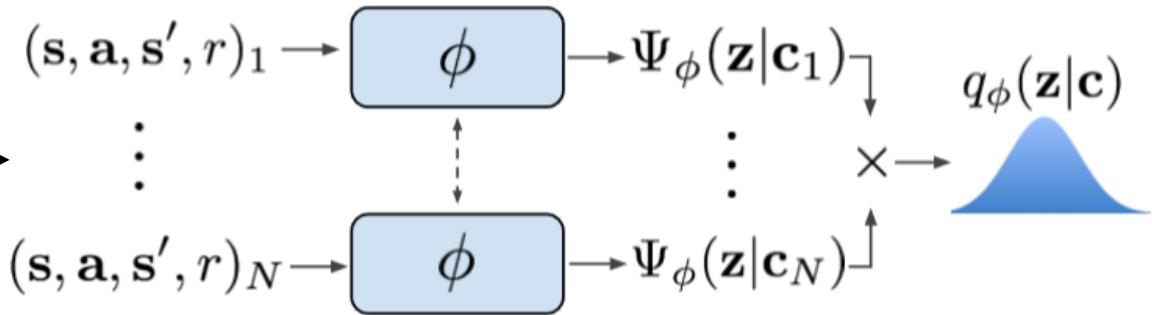
conceptually *very* similar to RNN meta-RL, but with stochastic z

stochastic z enables exploration via *posterior sampling*

Specific instantiation: PEARL

policy: $\pi_\theta(a_t|s_t, z_t)$

inference network: $q_\phi(z_t|s_1, a_1, r_1, \dots, s_t, a_t, r_t) \longrightarrow$



$$(\theta, \phi) = \arg \max_{\theta, \phi} \frac{1}{N} \sum_{i=1}^n E_{z \sim q_\phi, \tau \sim \pi_\theta} [R_i(\tau) - D_{\text{KL}}(q(z|\dots) \| p(z))]$$

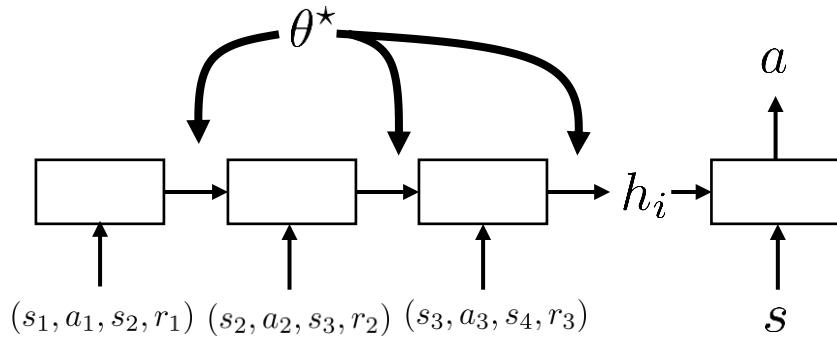
↑
perform maximization using soft actor-critic (SAC),
state-of-the-art off-policy RL algorithm

References on meta-RL, inference, and POMDPs

- Rakelly*, Zhou*, Quillen, Finn, Levine. **Efficient Off-Policy Meta-Reinforcement learning via Probabilistic Context Variables.** ICML 2019.
- Zintgraf, Igl, Shiarlis, Mahajan, Hofmann, Whiteson. **Variational Task Embeddings for Fast Adaptation in Deep Reinforcement Learning.**
- Humplik, Galashov, Hasenclever, Ortega, Teh, Heess. **Meta reinforcement learning as task inference.**

The three perspectives on meta-RL

Perspective 1: just RNN it



Perspective 2: bi-level optimization

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta)$$

MAML for RL

Perspective 3: it's an inference problem!

$$\pi_\theta(a|s, z) \quad z_t \sim p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$$

↑
everything needed to solve task

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

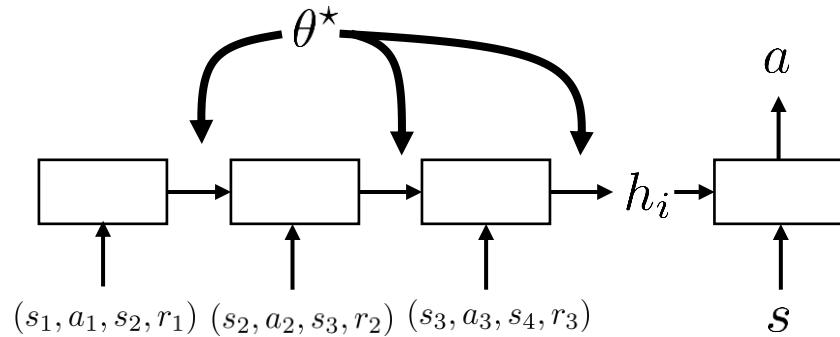
$$\text{where } \phi_i = f_\theta(\mathcal{M}_i)$$

what should $f_\theta(\mathcal{M}_i)$ do?

1. improve policy with experience from \mathcal{M}_i
 $\{(s_1, a_1, s_2, r_1), \dots, (s_T, a_T, s_{T+1}, r_T)\}$
2. (new in RL): choose how to interact, i.e. choose a_t
meta-RL must also *choose* how to *explore*!

The three perspectives on meta-RL

Perspective 1: just RNN it



Perspective 2: bi-level optimization

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta)$$

MAML for RL

Perspective 3: it's an inference problem!

$$\pi_\theta(a|s, z) \quad z_t \sim p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$$

everything needed to solve task

- + conceptually simple
- + relatively easy to apply
- vulnerable to *meta-overfitting*
- challenging to optimize in practice

- + good extrapolation (“consistent”)
- + conceptually elegant
- complex, requires many samples

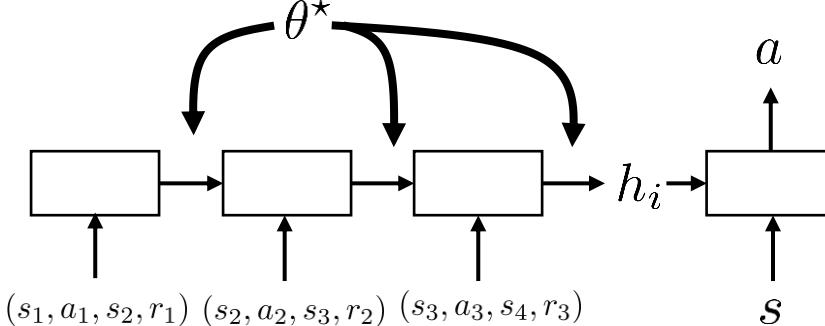
- + simple, effective exploration via posterior sampling
- + elegant reduction to solving a special POMDP
- vulnerable to *meta-overfitting*
- challenging to optimize in practice

But they're not that different!

just perspective 1,
but with stochastic
hidden variables!

i.e., $\phi = \mathbf{z}$

Perspective 1: just RNN it



Perspective 2: bi-level optimization

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta)$$

MAML for RL

Perspective 3: it's an inference problem!

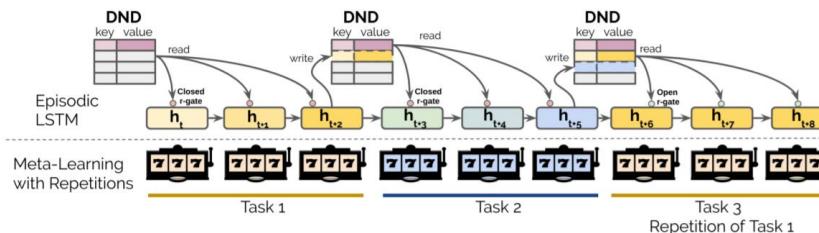
$$\pi_\theta(a|s, z) \quad z_t \sim p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$$

everything needed to solve task

just a particular
architecture choice
for these

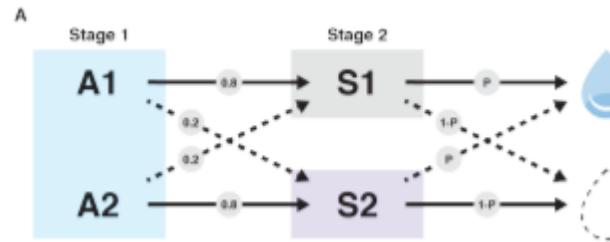
Meta-RL and emergent phenomena

meta-RL gives rise to
episodic learning



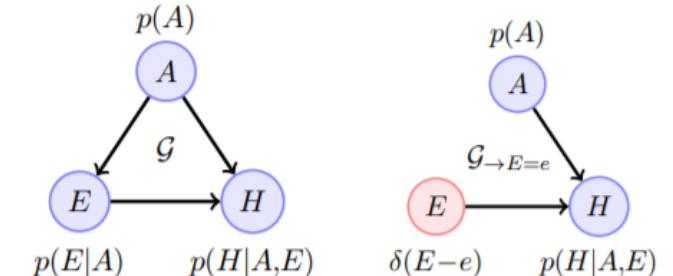
Ritter, Wang, Kurth-Nelson, Jayakumar, Blundell, Pascanu, Botvinick. **Been There, Done That: Meta-Learning with Episodic Recall.**

model-free meta-RL gives rise to
model-based adaptation



Wang, Kurth-Nelson, Kumaran, Tirumala, Soyer, Leibo, Hassabis, Botvinick. **Prefrontal Cortex as a Meta-Reinforcement Learning System.**

meta-RL gives rise to
causal reasoning (!)



Dasgupta, Wang, Chiappa, Mitrovic, Ortega, Raposo, Hughes, Battaglia, Botvinick, Kurth-Nelson. **Causal Reasoning from Meta-Reinforcement Learning.**

Humans and animals *seemingly* learn behaviors in a variety of ways:

- Highly efficient but (apparently) model-free RL
- Episodic recall
- Model-based RL
- Causal inference
- etc.

Perhaps each of these is a separate “algorithm” in the brain

But maybe these are all emergent phenomena resulting from meta-RL?

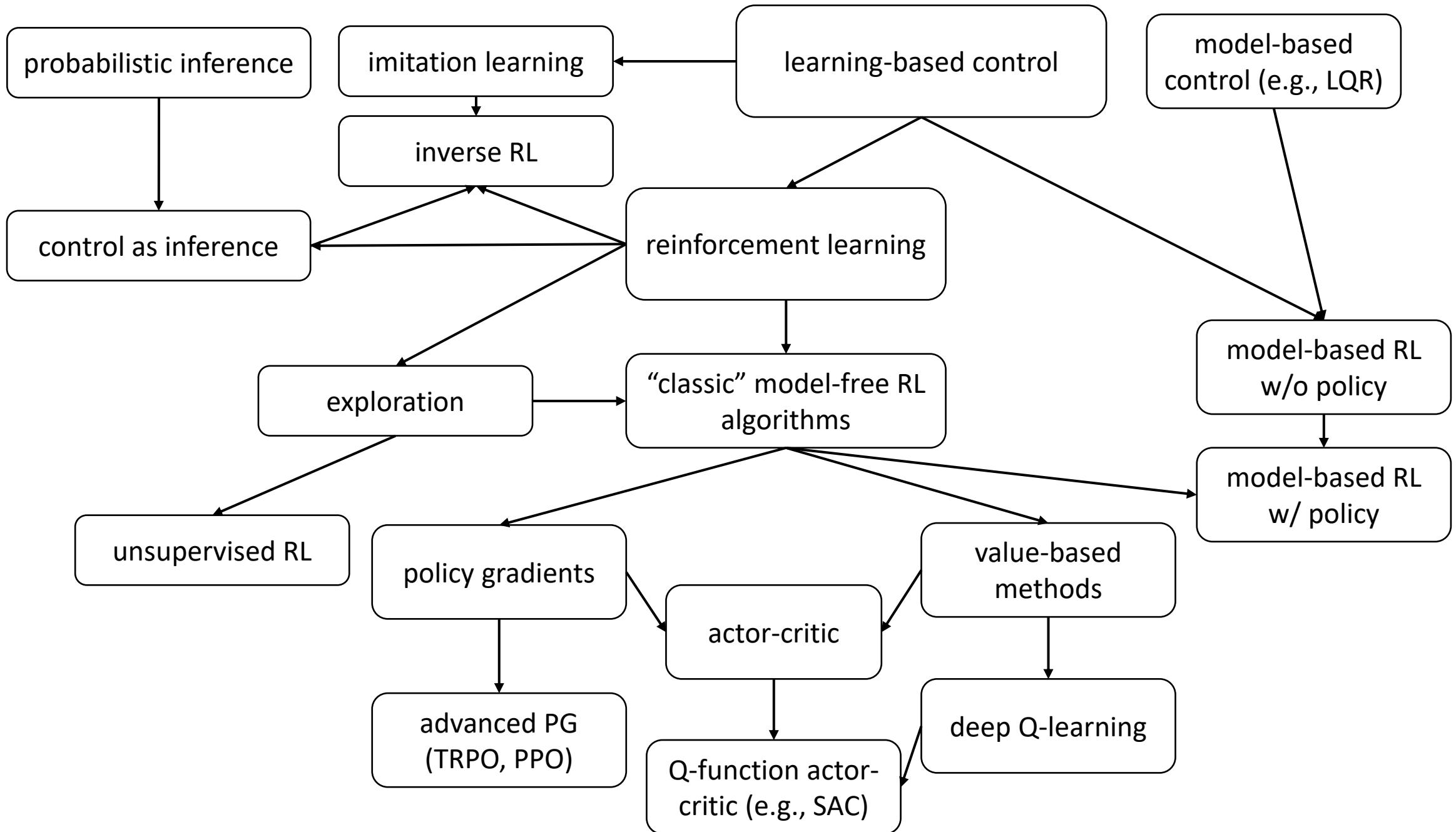
Challenges and Open Problems

CS 285

Instructor: Sergey Levine
UC Berkeley



A Brief Review



Challenges in Deep Reinforcement Learning

What's the problem?

Challenges with **core algorithms**:

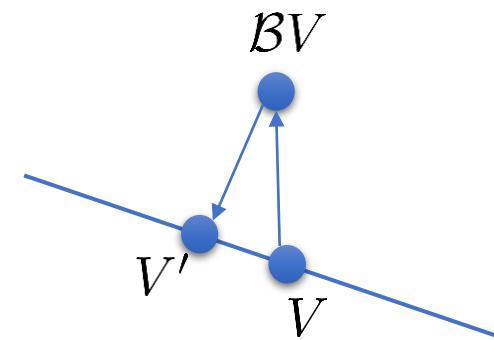
- Stability: does your algorithm converge?
- Efficiency: how long does it take to converge? (how many samples)
- Generalization: after it converges, does it generalize?

Challenges with **assumptions**:

- Is this even the right problem formulation?
- What is the source of *supervision*?

Stability and hyperparameter tuning

- Devising stable RL algorithms is very hard
- Q-learning/value function estimation
 - Fitted Q/fitted value methods with deep network function estimators are typically not contractions, hence no guarantee of convergence
 - Lots of parameters for stability: target network delay, replay buffer size, clipping, sensitivity to learning rates, etc.
- Policy gradient/likelihood ratio/REINFORCE
 - Very high variance gradient estimator
 - Lots of samples, complex baselines, etc.
 - Parameters: batch size, learning rate, design of baseline
- Model-based RL algorithms
 - Model class and fitting method
 - Optimizing policy w.r.t. model non-trivial due to backpropagation through time
 - More subtle issue: policy tends to *exploit* the model



gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, SAC, etc.)

10x

model-based deep RL
(e.g. PETS, guided policy search)

10x

model-based “shallow” RL
(e.g. PILCO)

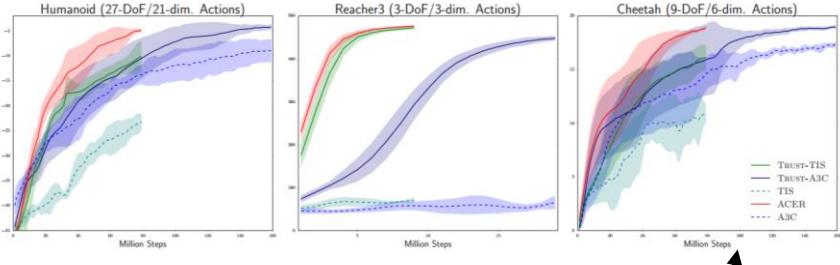
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans¹ Jonathan Ho¹ Xi Chen¹ Ilya Sutskever¹

half-cheetah (slightly different version)

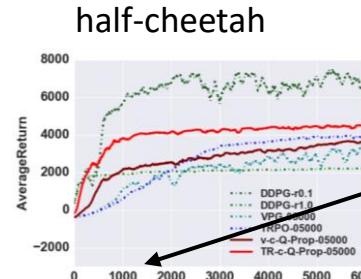


TRPO+GAE (Schulman et al. '16)



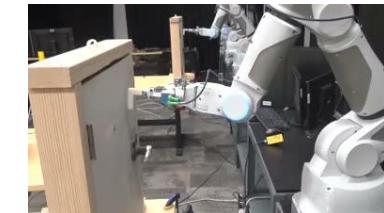
Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

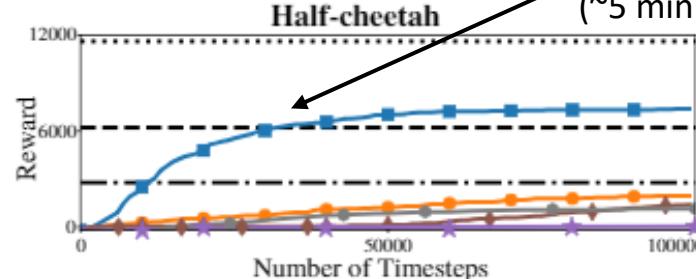


Gu et al. '16

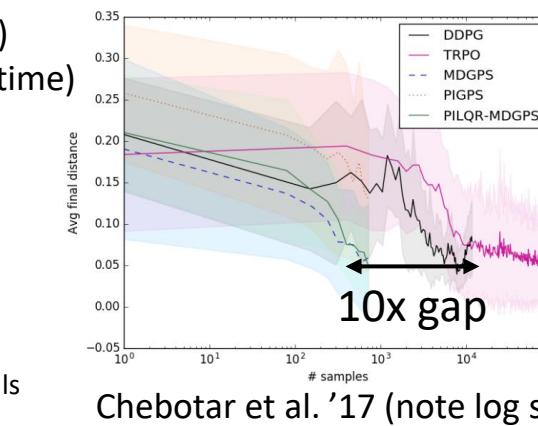
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)



1,000,000 steps
(1,000 episodes)
(~3 hours real time)



Chua et al. '18: Deep Reinforcement Learning in a Handful of Trials



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

The challenge with sample complexity

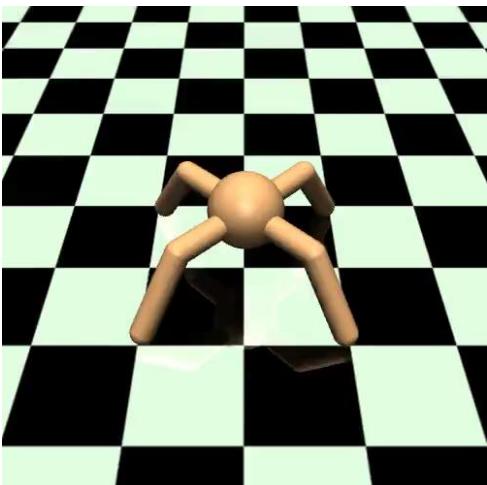
- Need to wait for a long time for your homework to finish running
- Real-world learning becomes difficult or impractical
- Precludes the use of expensive, high-fidelity simulators
- Limits applicability to real-world problems



Scaling up deep RL & generalization



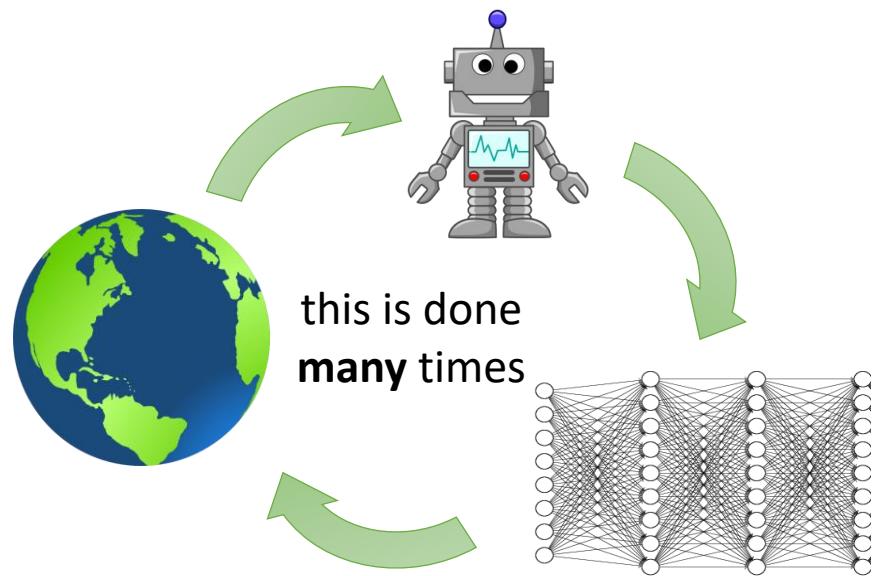
- Large-scale
- Emphasizes diversity
- Evaluated on generalization



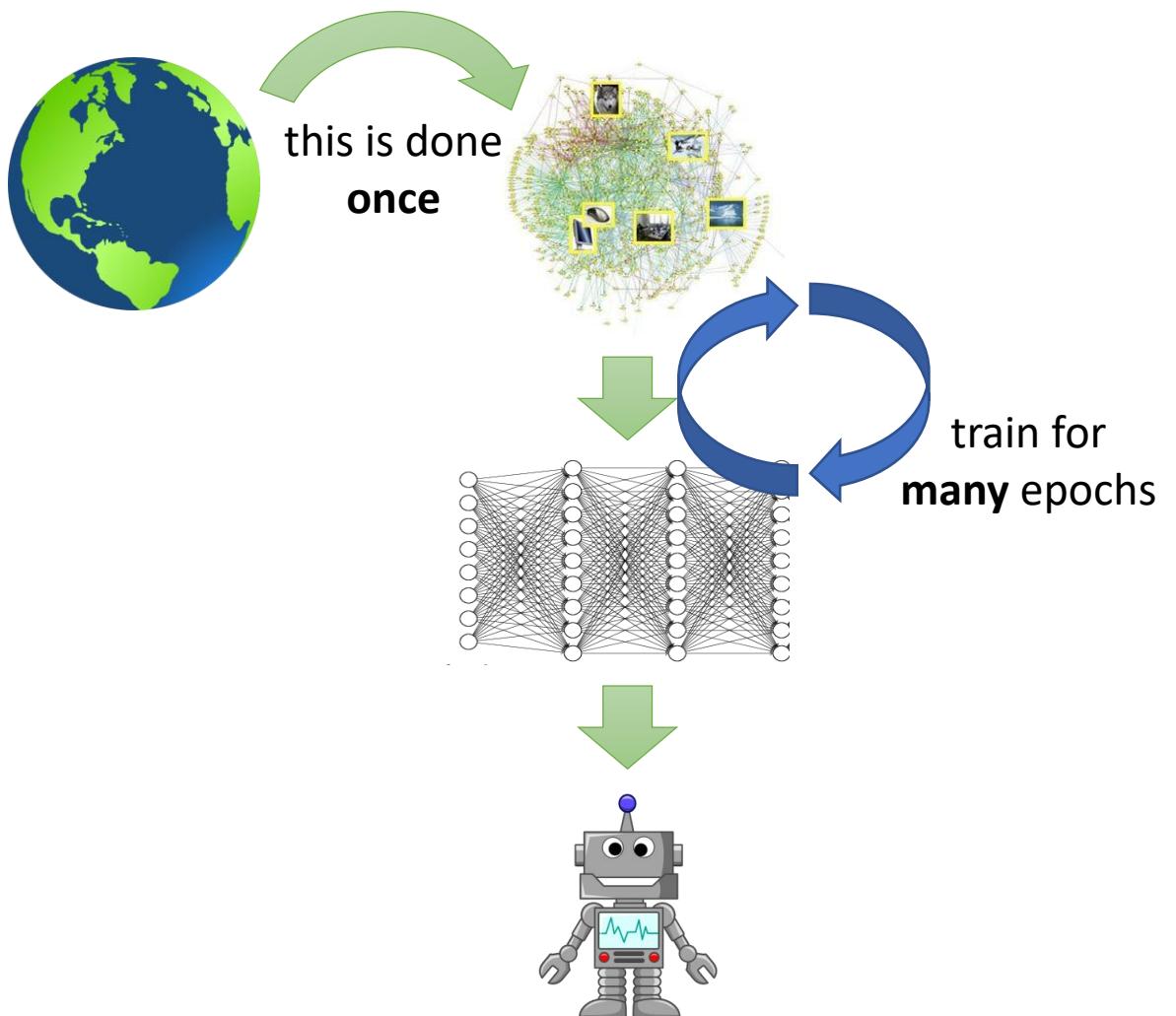
- Small-scale
- Emphasizes mastery
- Evaluated on performance
- Where is the generalization?

RL has a **big** problem

reinforcement learning

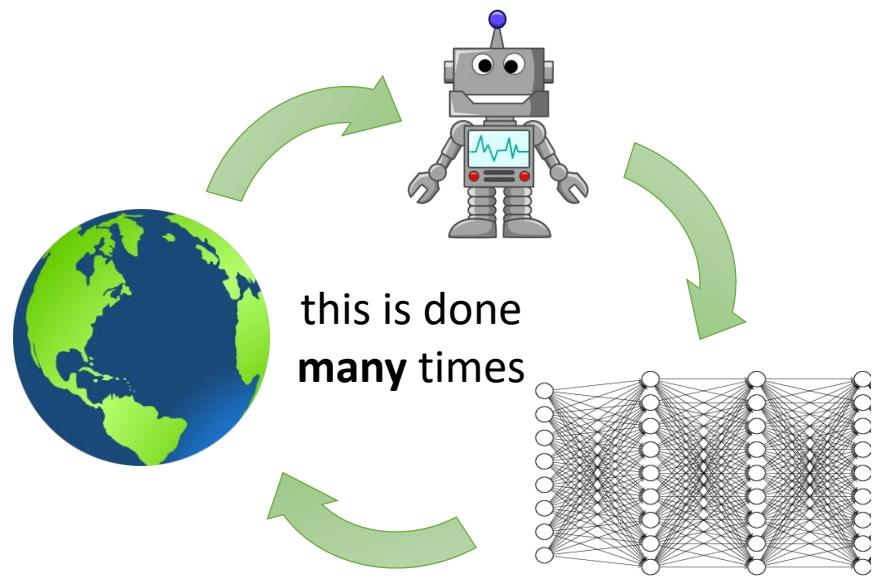


supervised machine learning

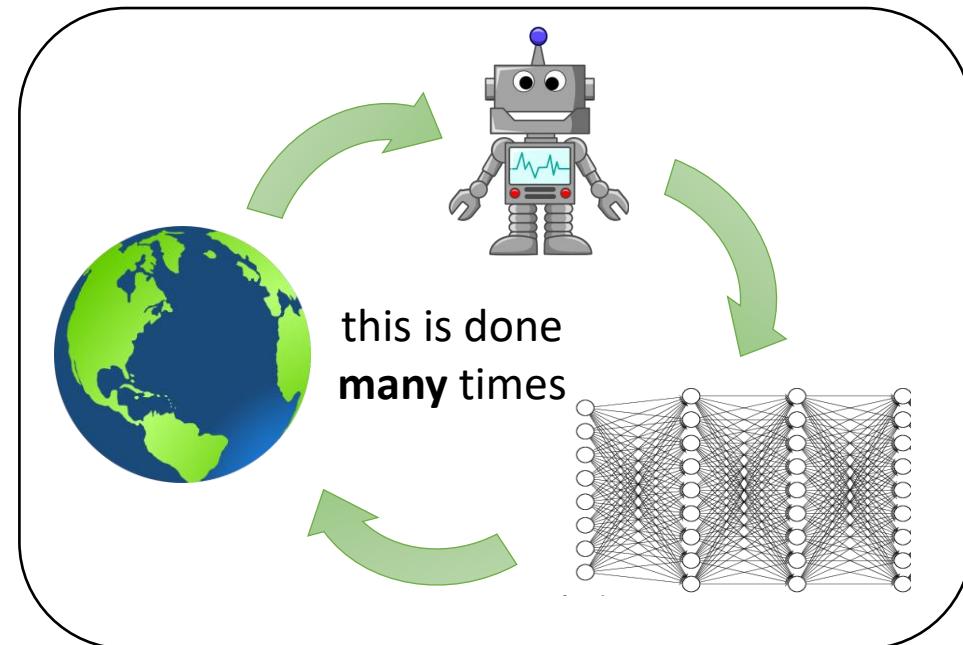


RL has a **big** problem

reinforcement learning

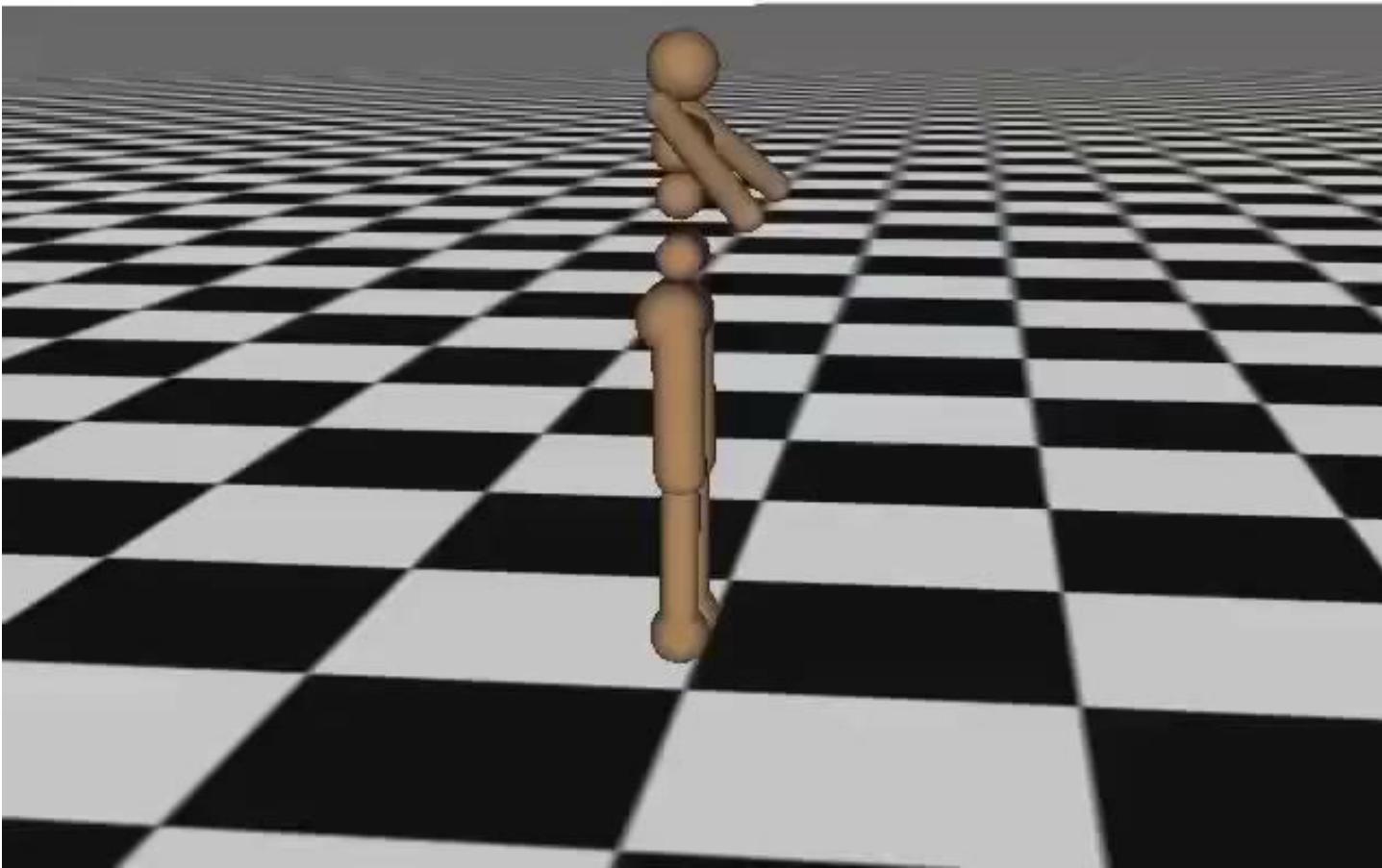


actual reinforcement learning



How bad is it?

Iteration 0



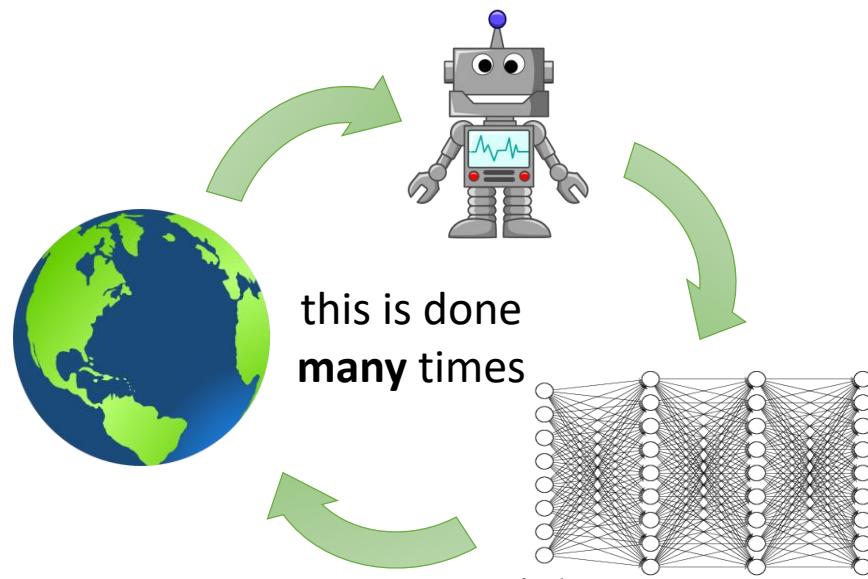
- This is quite cool
- It takes 6 days of real time (if it was real time)
- ...to run on an infinite flat plane



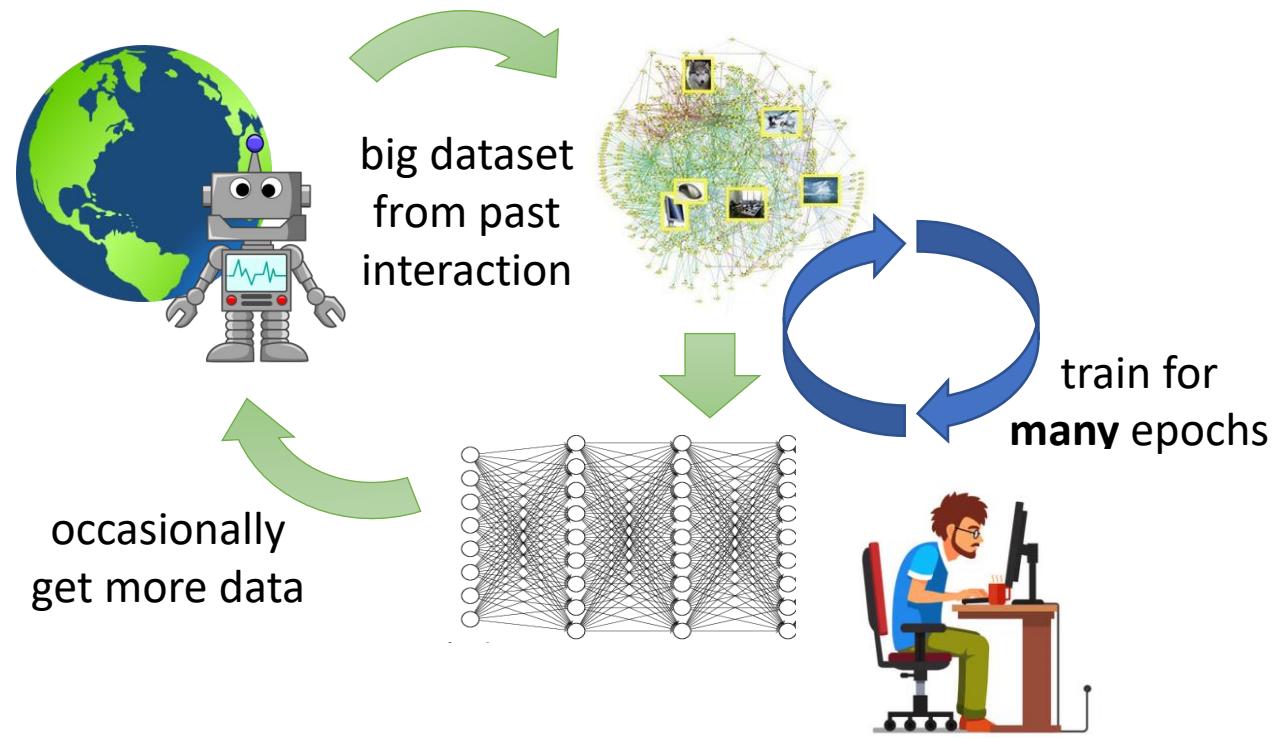
The real world is not so simple!

Off-policy RL?

reinforcement learning



off-policy reinforcement learning



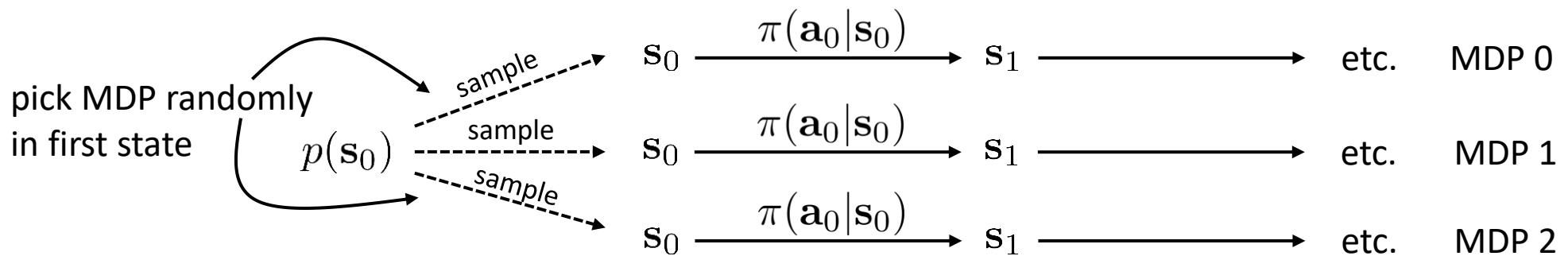
Single task or multi-task?



this is where generalization can come from...

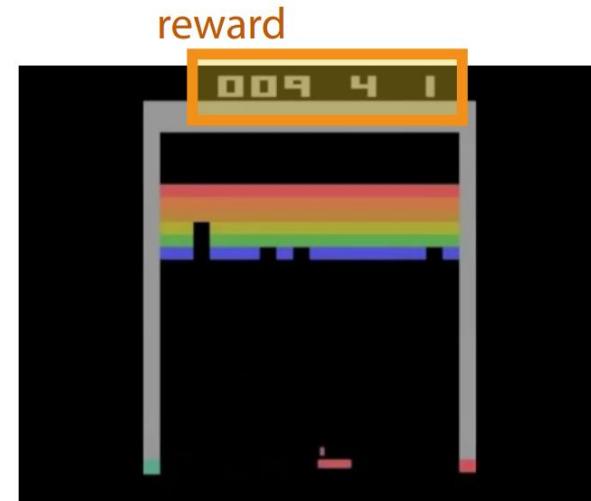
maybe doesn't require any new assumption, but might merit additional treatment

The real world is not so simple!



Where does the supervision come from?

- If you want to learn from many different tasks, you need to get those tasks somewhere!
- Learn objectives/rewards from demonstration (inverse reinforcement learning)
- Generate objectives automatically?



what is the **reward**?

Other sources of supervision

- Demonstrations

- Muelling, K et al. (2013). Learning to Select and Generalize Striking Movements in Robot Table Tennis

Should supervision tell us **what** to do or **how** to do it?



- Language

- Andreas et al. (2018). Learning with latent language

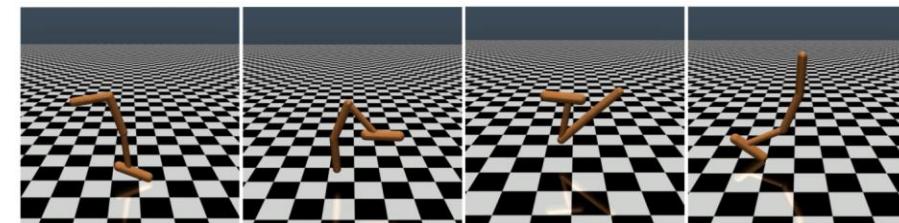
Human description:
move to the star

Inferred description:
reach the star cell



- Human preferences

- Christiano et al. (2017). Deep reinforcement learning from human preferences



Rethinking the Problem Formulation

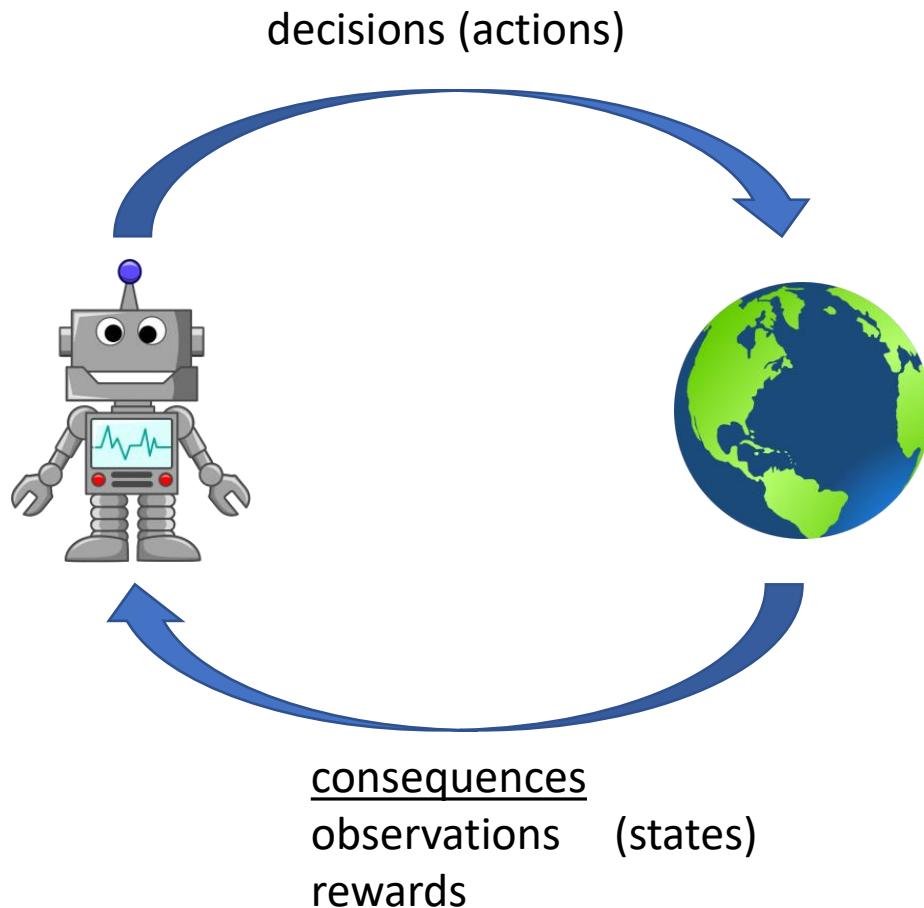
- How should we define a *control* problem?
 - What is the data?
 - What is the goal?
 - What is the supervision?
 - may not be the same as the goal...
- Think about the assumptions that fit your problem setting!
- Don't assume that the basic RL problem is set in stone

Some perspectives...

Reinforcement Learning as an Engineering Tool
Reinforcement Learning and the Real World
Reinforcement Learning as “Universal” Learning

Reinforcement Learning as an Engineering Tool

What we think RL is...



Engineering a control system



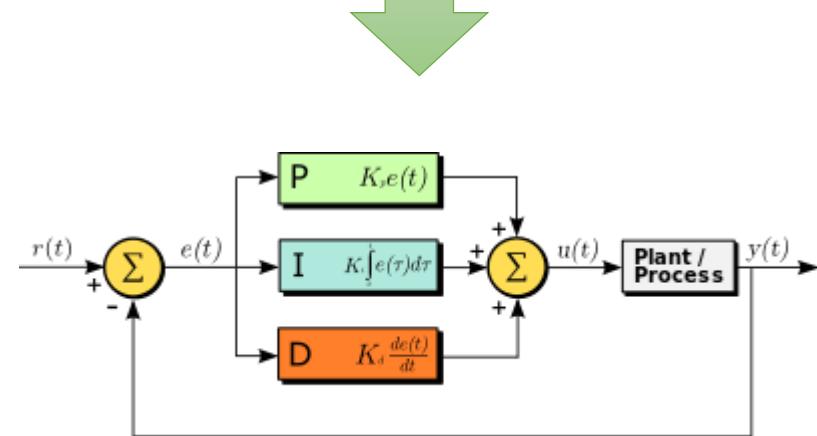
$$\mathbf{r} = \mathbf{r}(t) = r\hat{\mathbf{e}}_r$$

$$\mathbf{v} = v\hat{\mathbf{e}}_r + r \frac{d\theta}{dt}\hat{\mathbf{e}}_\theta + r \frac{d\varphi}{dt} \sin \theta \hat{\mathbf{e}}_\varphi$$

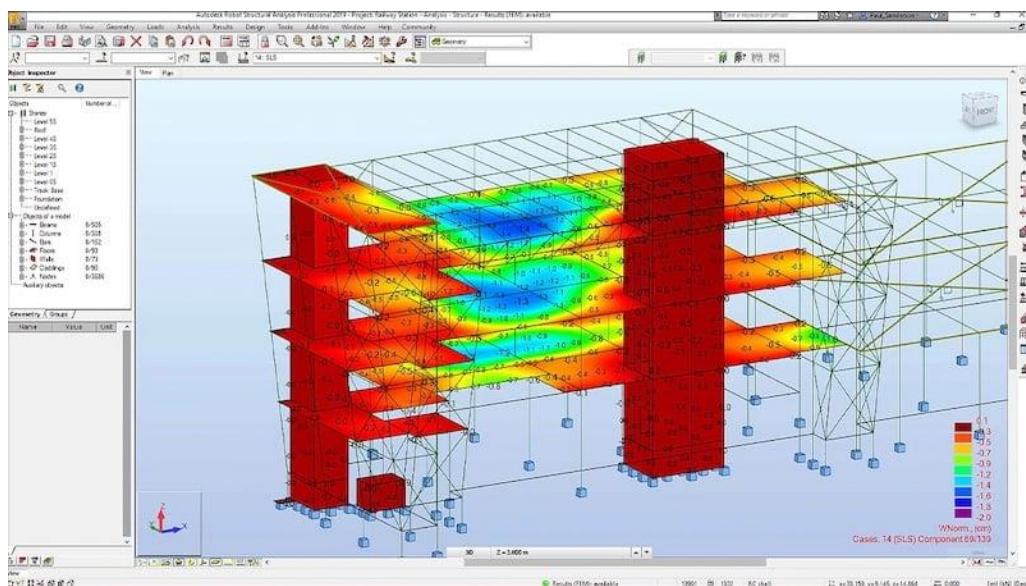
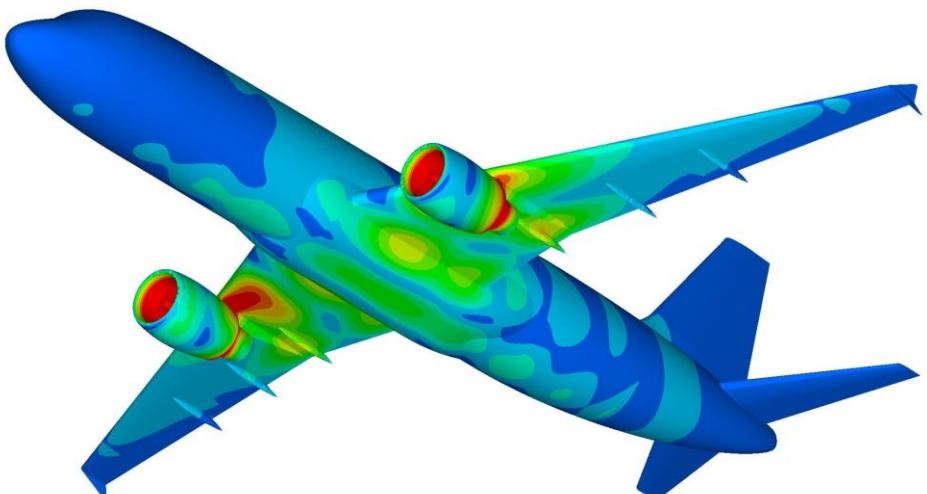
$$\mathbf{a} = \left(a - r \left(\frac{d\theta}{dt} \right)^2 - r \left(\frac{d\varphi}{dt} \right)^2 \sin^2 \theta \right) \hat{\mathbf{e}}_r$$

$$+ \left(r \frac{d^2\theta}{dt^2} + 2v \frac{d\theta}{dt} - r \left(\frac{d\varphi}{dt} \right)^2 \sin \theta \cos \theta \right) \hat{\mathbf{e}}_\theta$$

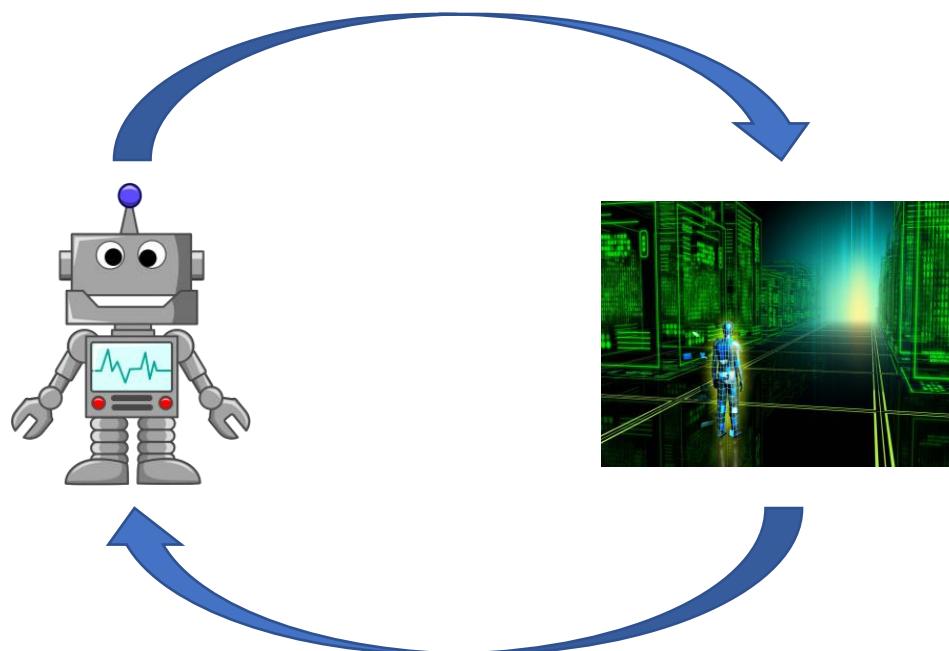
$$+ \left(r \frac{d^2\varphi}{dt^2} \sin \theta + 2v \frac{d\varphi}{dt} \sin \theta + 2r \frac{d\theta}{dt} \frac{d\varphi}{dt} \cos \theta \right) \hat{\mathbf{e}}_\varphi$$



Characterization and simulation...



RL: anything you can *simulate* you can *control*



- Provides a powerful engineering tool
- Now *that* different from conventional engineering approach!
 - Before: characterize, simulate, control
 - Now: characterize, simulate, run RL
- Main role: powerful *inversion* engine
- Main weakness: still need to simulate!

Reinforcement Learning and the Real World

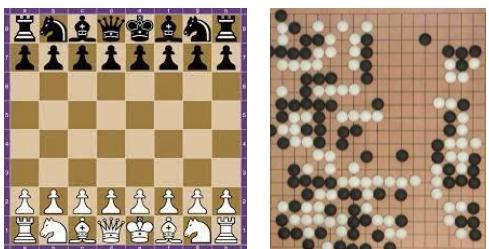


Moravec's paradox

Moravec's paradox seems like a statement about AI

but it is actually a statement about the physical universe

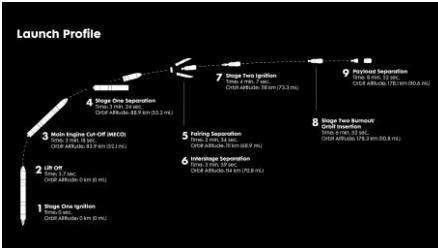
“easy” universes



“hard” universes



Why?



We are all prodigious olympians in perceptual and motor areas, so good that we make the difficult look easy. Abstract thought, though, is a new trick, perhaps less than 100 thousand years old. We have not yet mastered it. It is not all that intrinsically difficult; it just seems so when we do it.

- Hans Moravec

The main lesson of thirty-five years of AI research is that the hard problems are easy and the easy problems are hard. The mental abilities of a four-year-old that we take for granted – recognizing a face, lifting a pencil, walking across a room, answering a question – in fact solve some of the hardest engineering problems ever conceived.

- Steven Pinker



What does this have to do with RL?



How do we engineer a system that can deal with the unexpected?

- Minimal external supervision about what to do
- Unexpected situations that require adaptation
- Must discover solutions autonomously
- Must “stay alive” long enough to discover them!

- Humans are extremely good at this
- Current AI systems are extremely bad at this
- RL *in principle* can do this, and nothing else can

So what's the problem?



**RL *should* be really good in the
“hard” universes!**

➤ RL *in principle* can do this, and nothing else can

But we rarely study this kind of setting in RL research!

“easy” universes

success = high reward
 (“optimal control”)

closed world, rules
are known

lots of simulation

Main question: can RL
algorithms **optimize**
really well

“hard” universes

success = “survival”
 (“good enough control”)

open world, everything
must come from data

no simulation (because
rules are unknown)

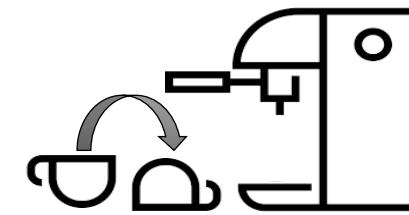
Main question: can RL
generalize and adapt

Some questions that come up in the real world

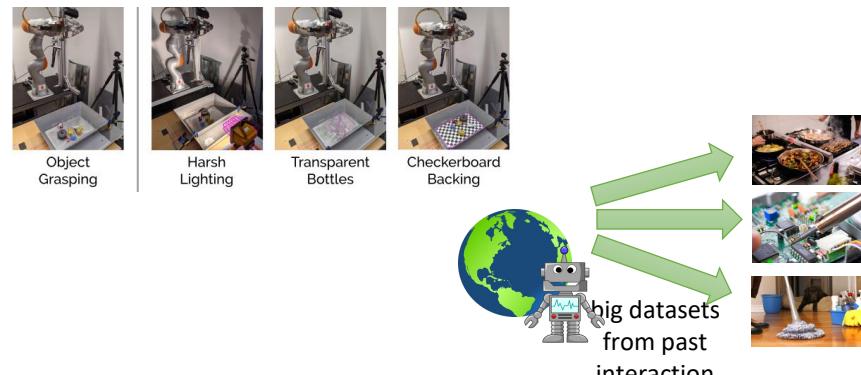
How do we tell RL agents **what we want them to do?**



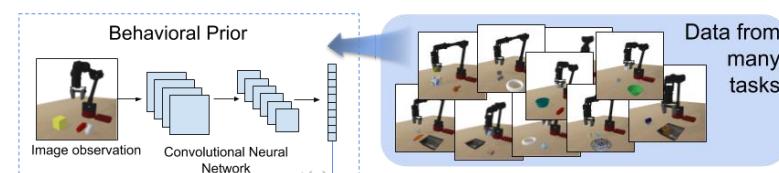
How can we learn **fully autonomously** in continual environments?



How do remain **robust** as the environment changes around us?



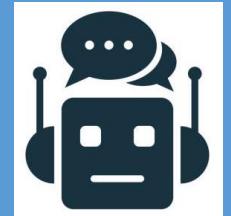
What is the right way to **generalize** using **experience & prior data**?



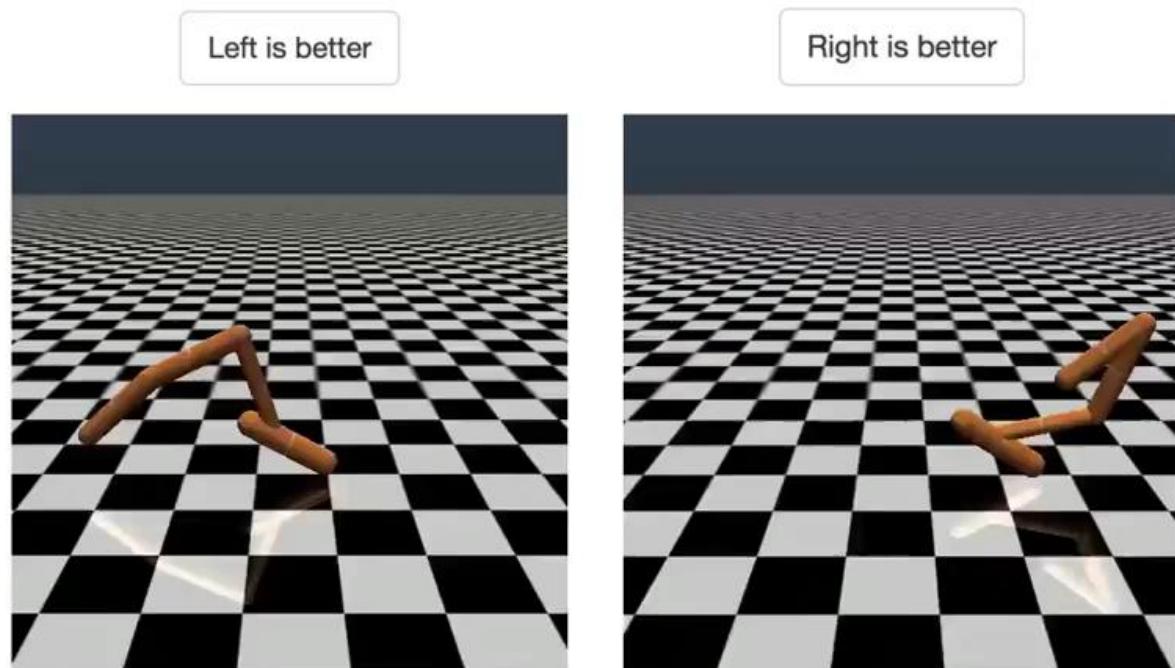
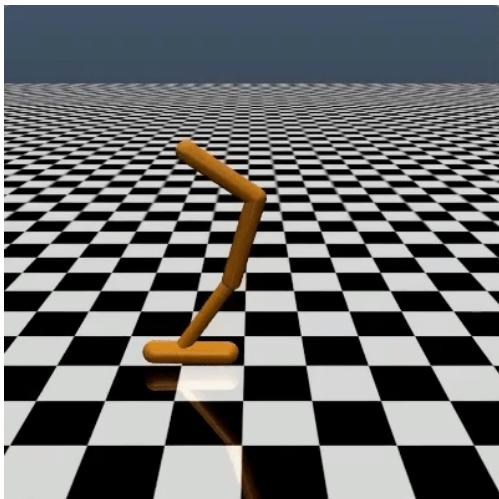
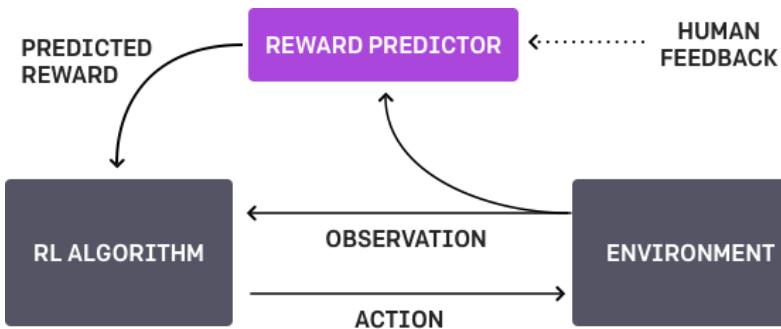
What is the right way to **bootstrap exploration** with **prior experience**?

This is not about robots

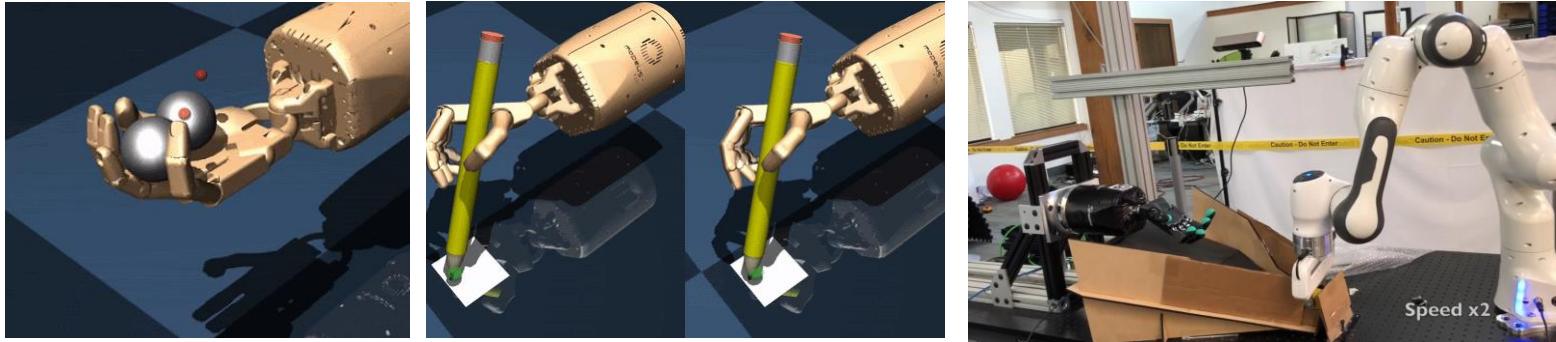
Robots are the most natural for us to think about, because they are embodied like we are



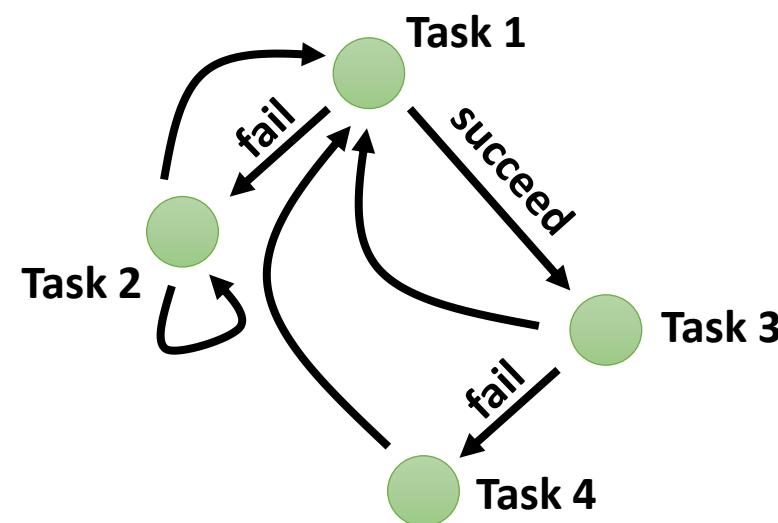
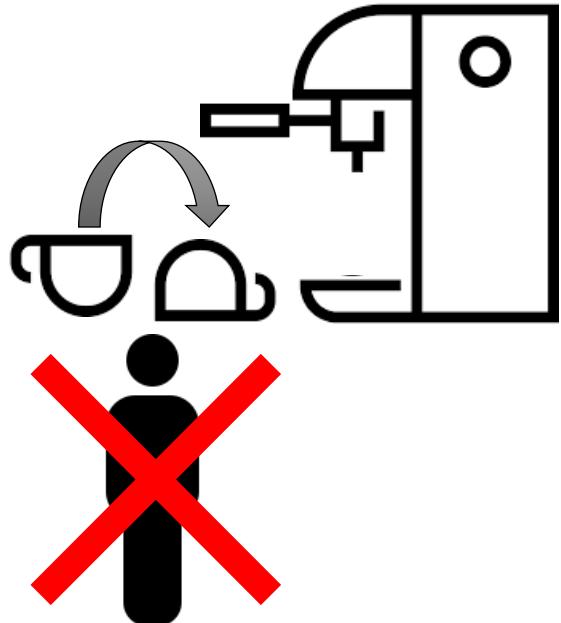
Other ways to communicate objectives?



How can we learn fully autonomously?



Nagabandi, Konolige, Levine, Kumar. Deep Dynamics Models for Learning Dexterous Manipulation. CoRL 2019.



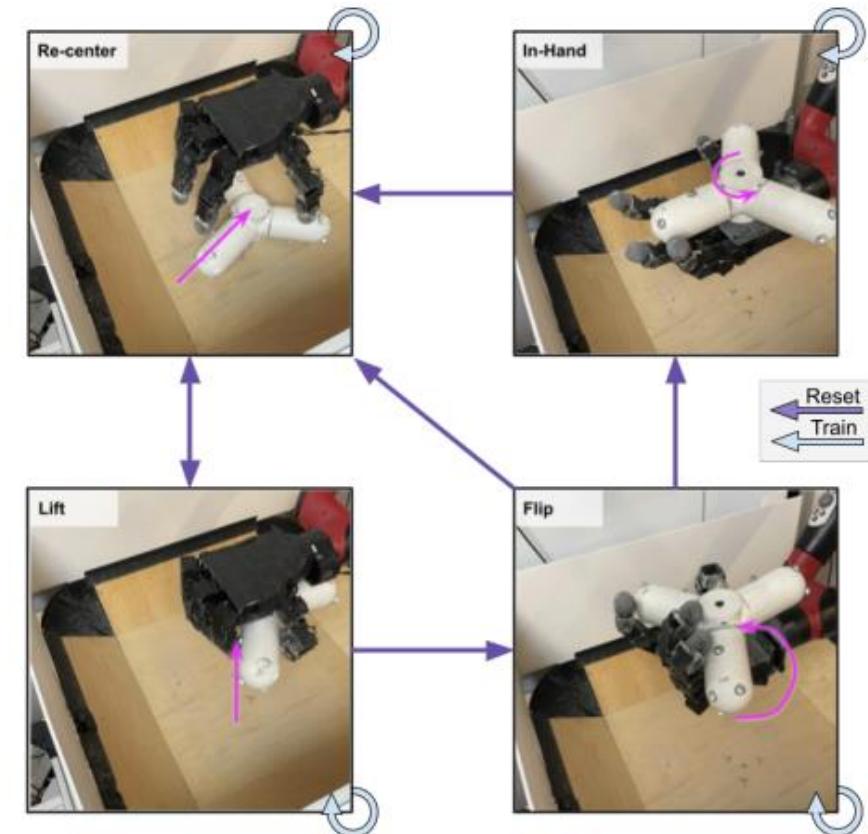
Task 1: put cup in coffee machine

Task 2: pick up cup

Task 3: replace cup

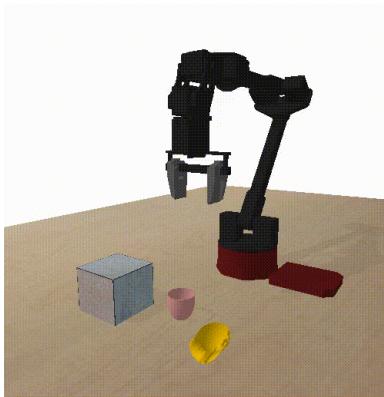
Task 4: clean up spill from cup...

How can we learn fully autonomously?

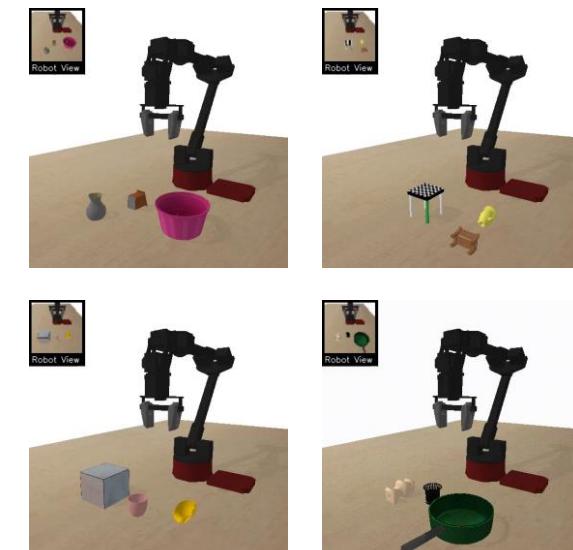
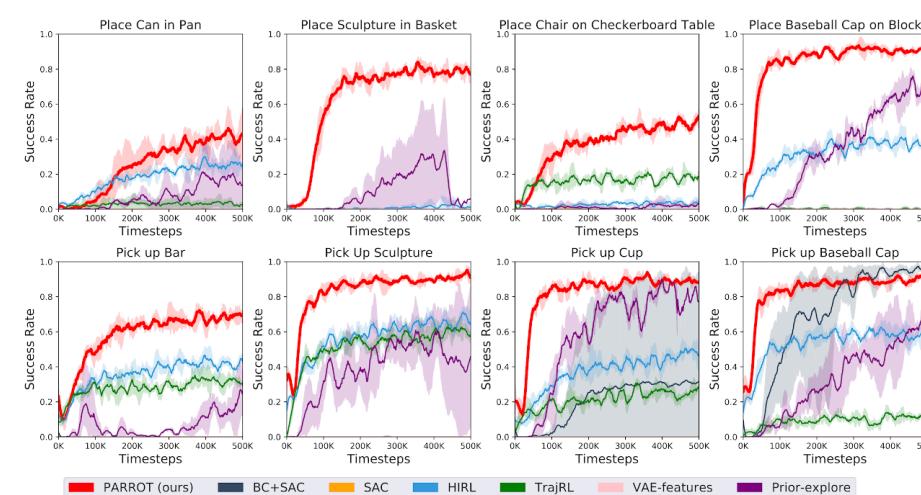
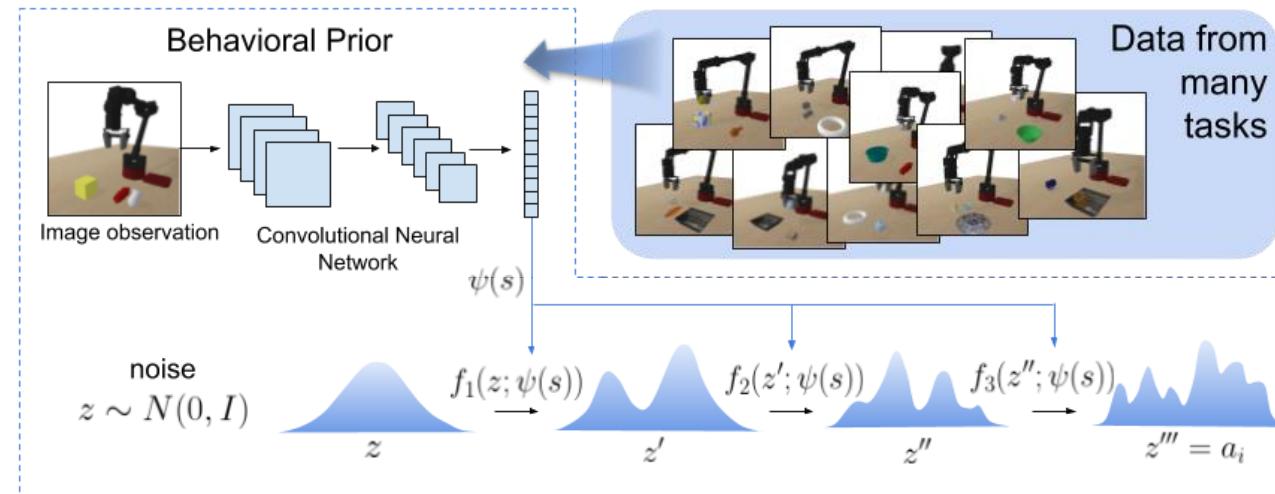
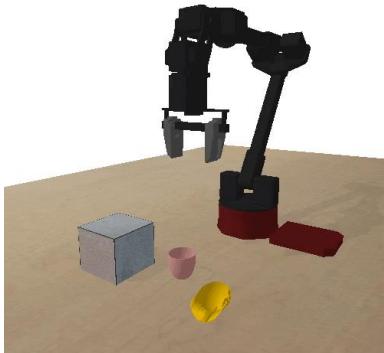


How to bootstrap exploration from experience?

exploring from scratch



exploring from behavioral prior



This all seems really hard, what's the point?



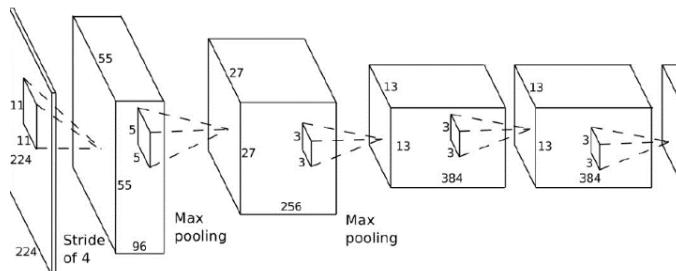
Why is this interesting?

- It's exciting to see what solutions intelligent agents come up with
 - Most exciting if they come up with something we don't expect
 - This requires the world they inhabit to admit novel solutions
 - This means that world must be complicated enough!
-
- To see interesting emergent behavior, we must train our systems in environments that actually require interesting emergent behavior!
 - RL in the real world may be difficult, but it is also rewarding

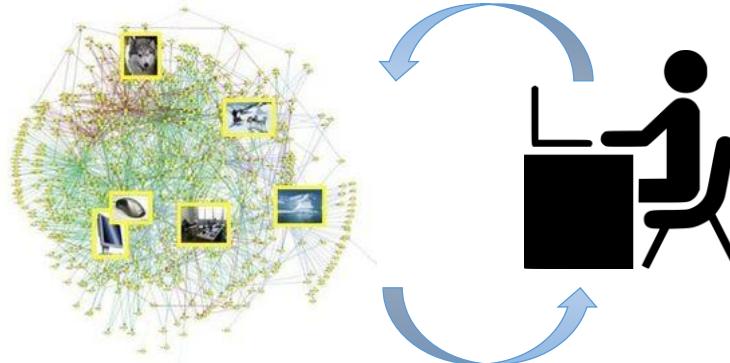
Reinforcement Learning as “Universal” Learning

Large-scale machine learning

Why does deep learning work?

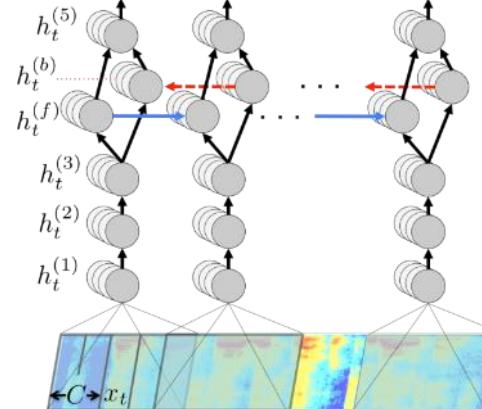
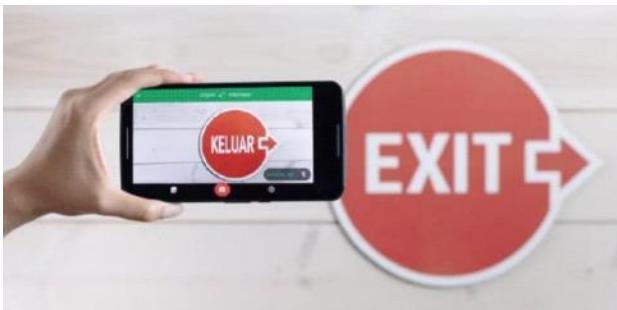


big models

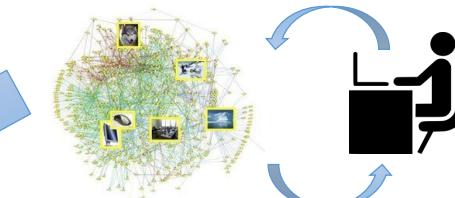
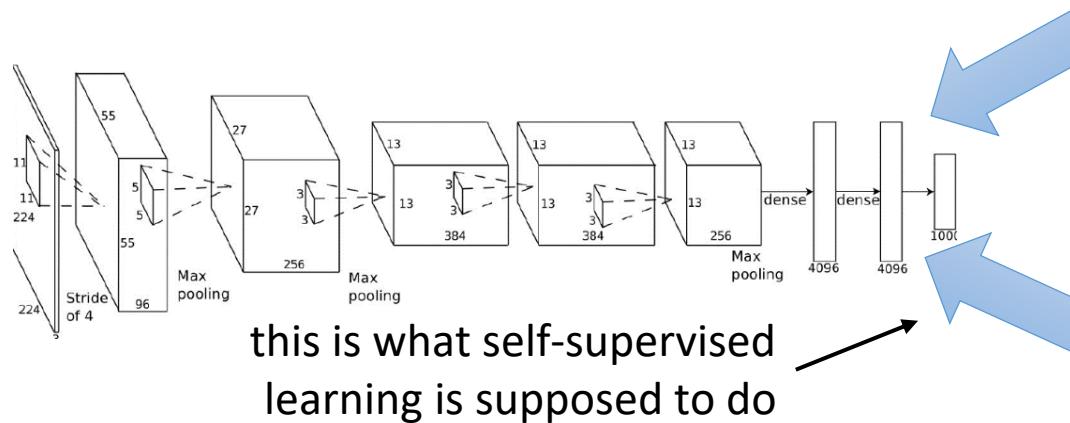


big datasets

labeled



Reducing the supervision burden



small labeled dataset



giant unlabeled garbage dataset (aka the Internet)

But then where does the knowledge come from?

“Classic” unsupervised learning: $p_\theta(\mathbf{x})$

(this is what, for example, large language models do)



Aside: perhaps this is why “prompting” large language models is such an art!

Stepping back a bit...

Why do we need **machine learning** anyway?

A postulate:

We need machine learning for one reason and one reason only – that's to produce **adaptable and complex decisions**.



Decision: how do I move my joints?



Decision: how do I steer the car?



What is the decision? The image label?

Usually not!

What happens with that label **afterwards**?

Is it used to tag a user's photo?

Detect an endangered animal in a camera trap?

Aside: why do we need **brains** anyway?

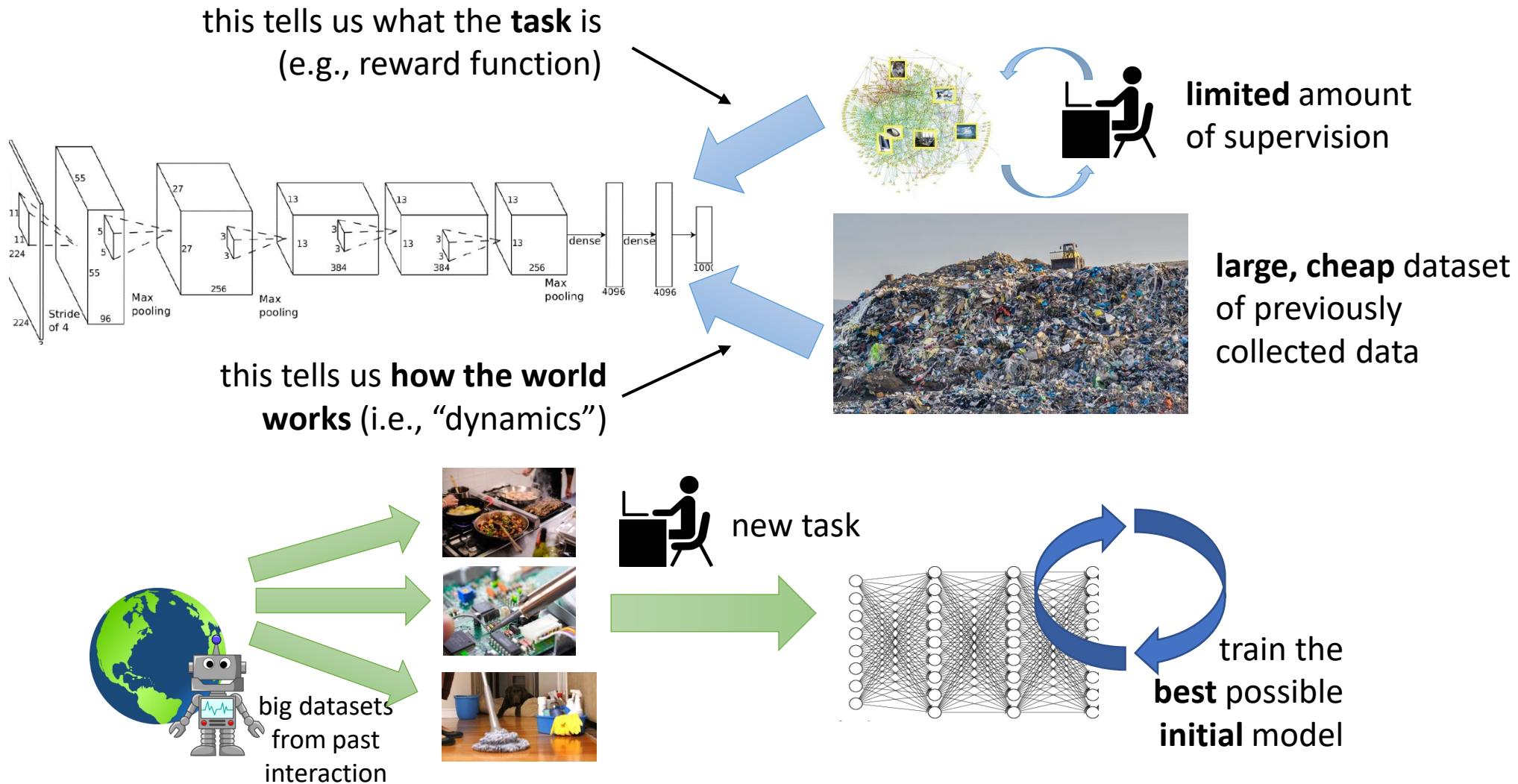


Daniel Wolpert
(knows quite a lot about brains)

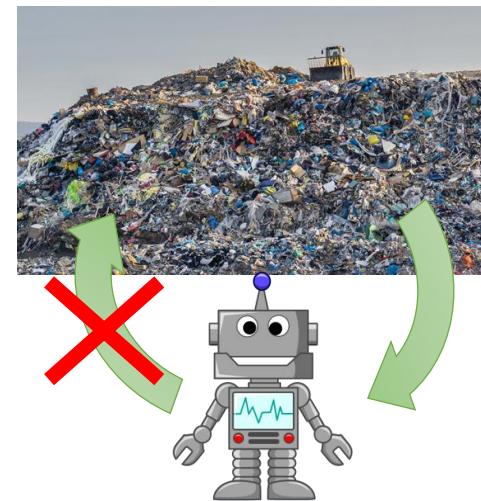
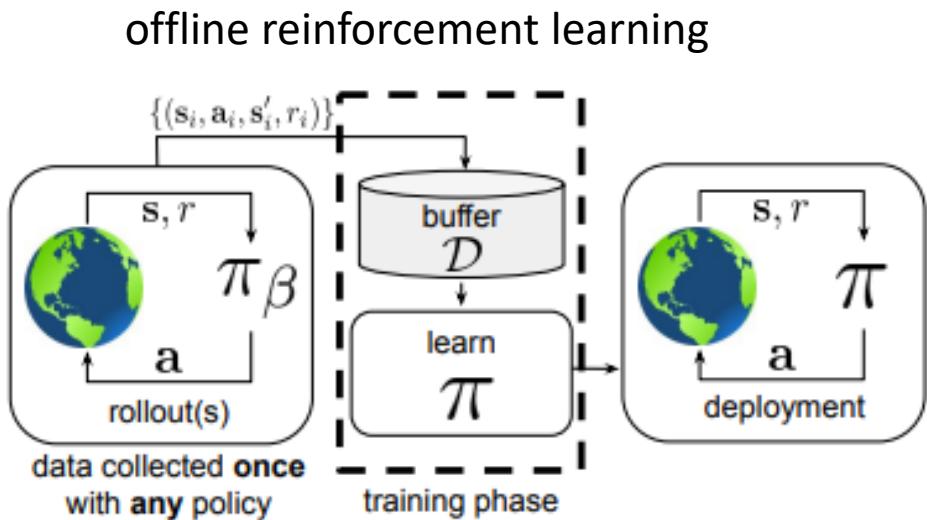


"We have a brain for one reason and one reason only -- that's to produce **adaptable and complex movements**. Movement is the only way we have affecting the world around us... I believe that to understand movement is to understand the whole brain."

Reinforcement learning as a way to use “cheap” (previously collected) data



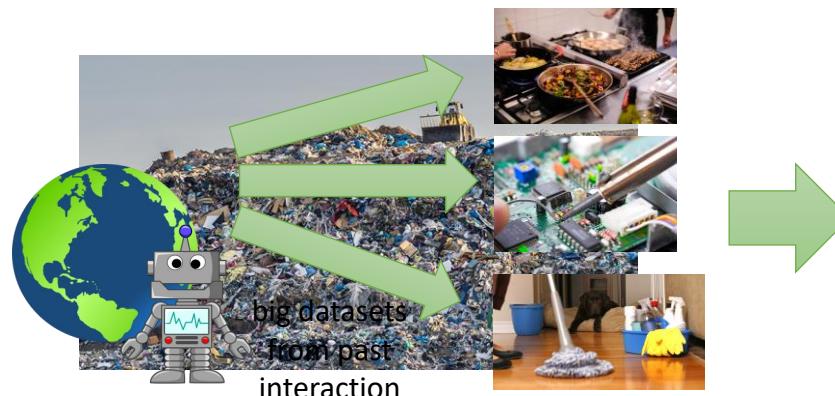
The RL + data problem



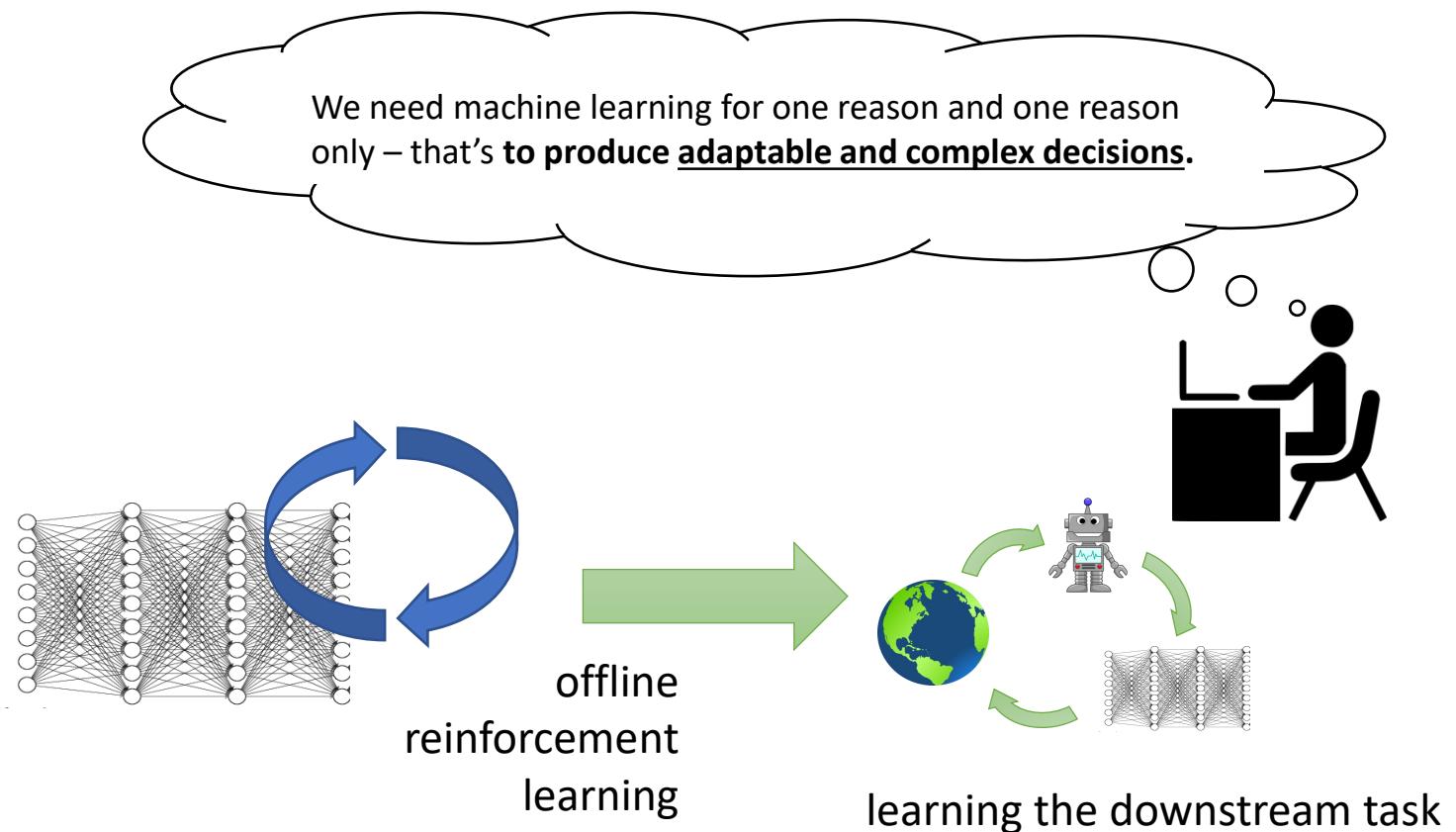
with naïve RL, this is a **costly** interactive process if done in the real world!

but self-supervised learning is about using **cheap** data that we already have lying around (in the garbage heap)!

The recipe



large dataset of diverse
(but possibly low-quality)
behavior



there are a few different
choices here:

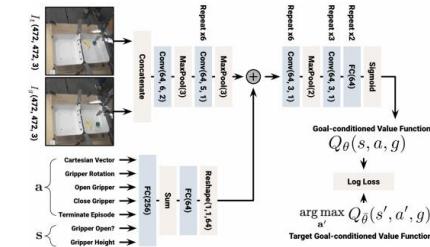
- human-defined skills
- goal-conditioned RL
- self-supervised skill discovery

Can we learn from offline data without well-defined tasks?

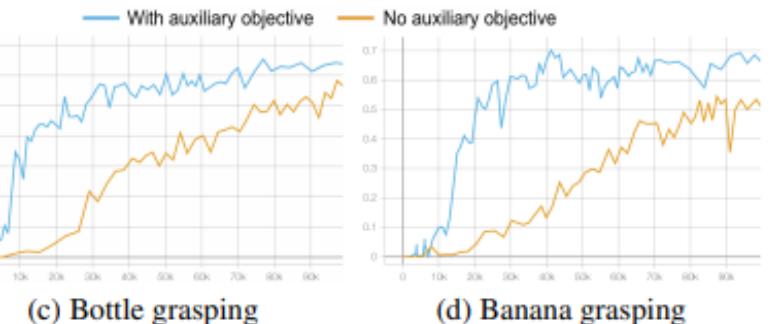


- No reward function at all, task is defined entirely using a **goal image**
- Uses a conservative offline RL method designed for goal-reaching, based on CQL
- Works very well as an **unsupervised pretraining objective**

1. Train **goal-conditioned Q**-function with offline RL



2. Finetune with a **task reward** and limited data



Can offline RL train large language models?

What we want:



Can you teach me about behavior cloning?



Of course! I'd be happy to explain behavior cloning. To start, could you tell me if you've ever come across the terms "artificial intelligence" or "machine learning" before?



Yes, I've heard those terms, but I'm not exactly sure what they mean.



No problem at all! Let's take it step by step. Have you ever used a computer or a smartphone?

What we get:



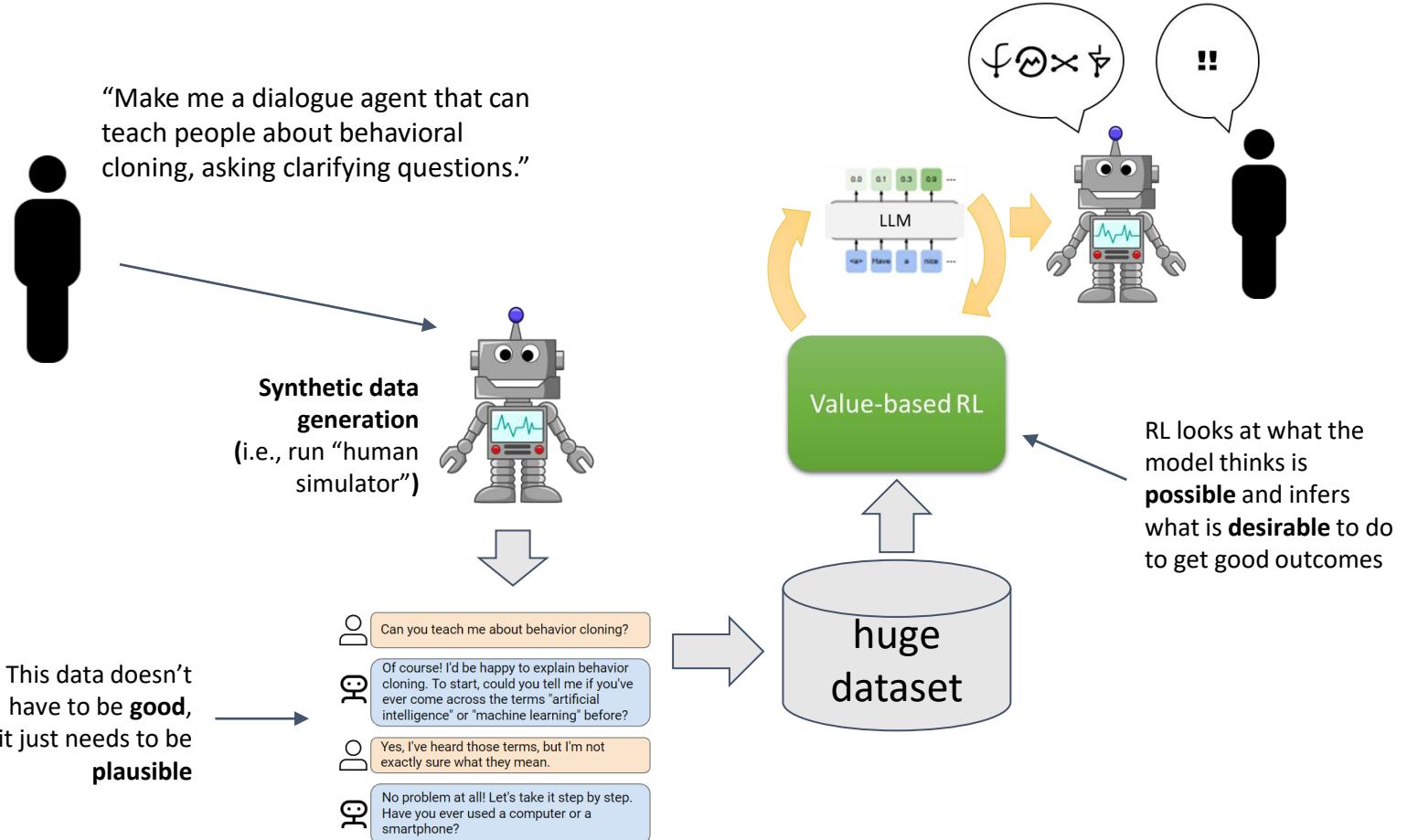
J Can you explain the AI concept of behavior cloning to me? But first, I want you to ask me questions to figure out what my background in AI is, then tailor your explanation to match my level.



Certainly! Let's start by understanding your background in AI to provide an explanation that matches your level of knowledge. Please answer the following questions:

1. Have you ever studied or worked with artificial intelligence or machine learning concepts before, even at a basic level?
2. Are you familiar with terms like "neural networks," "training data," or "supervised learning"?
3. Have you ever heard of or worked with specific AI tools or frameworks like TensorFlow, PyTorch, or scikit-learn?
4. Are you interested in a high-level overview or a more detailed explanation of behavior cloning?
5. Do you have any specific examples or applications in mind where you'd like to see behavior

Model-based offline RL for multi-turn dialogue



RL + LLM is great for tasks that are easy for LLMs to **simulate** but hard for them to **perform optimally**.

With RL, instead of emulating humans, LLMs can learn to achieve desired outcomes using their understanding of how humans behave.

Example: model-based RL for multi-turn dialogue

GPT Agent Dialogue:

H: Hi, I've been hearing about something called "behavior cloning." I don't really understand what it is. Can you explain it to me?

AI: Certainly! Let's start by understanding your background. Have you ever used a computer or worked with artificial intelligence or machine learning before? Are you familiar with terms like "neural networks," "training data," or "optimization?"

H: I have some basic knowledge of artificial intelligence and machine learning. I'm an expert. I am familiar with neural networks and supervised learning.

AI: Great, thank you for that information. Behavior cloning is a type of reinforcement learning, which is a subcategory of machine learning. In reinforcement learning, an AI model learns to make optimal decisions by interacting with its environment and receiving feedback in the form of rewards or punishments. Behavior cloning, on the other hand, is a specific technique that imitates or "clones" the behavior of an already established agent, such as a human expert or a pre-existing system. Does that make sense so far?

H: Yes, that's starting to make sense. What do you mean by "expert"?

AI: An expert, in this context, refers to a human or an advanced AI model that has learned how to solve a particular task or problem. The AI model then uses this knowledge to learn how to perform the task itself. Does that clarify things?

H: Yes, thank you.

IE-RL Agent Dialogue:

H: Hi there! I've heard about this thing called "behavior cloning" in AI, but I don't really understand what it is. Can you explain it to me?

AI: Of course! I'd be happy to explain behavior cloning. To start, could you tell me if you've ever come across the terms "artificial intelligence" or "machine learning" before?

H: Yes, I've heard those terms, but I'm not exactly sure what they mean.

AI: No problem at all! Let's take it step by step. Have you ever used a computer or a smartphone?

H: Yes, I use my computer and phone all the time.

AI: Great! So, imagine that a computer or a smartphone could learn to do something by watching you do it. That's a bit like what behavior cloning is.

H: Oh, interesting. Can you explain more about how that works?

AI: Sure. Behavior cloning is a type of machine learning algorithm where a computer learns to perform a task by imitating a human's behavior. It does this by closely analyzing and mimicking the actions of a human expert, using a large dataset of examples. This approach is often used in tasks that don't have a clear set of rules or can't be easily solved through traditional programming methods. Does that make sense?

H: Yes, I think so. So how does the computer learn to mimic the behavior?

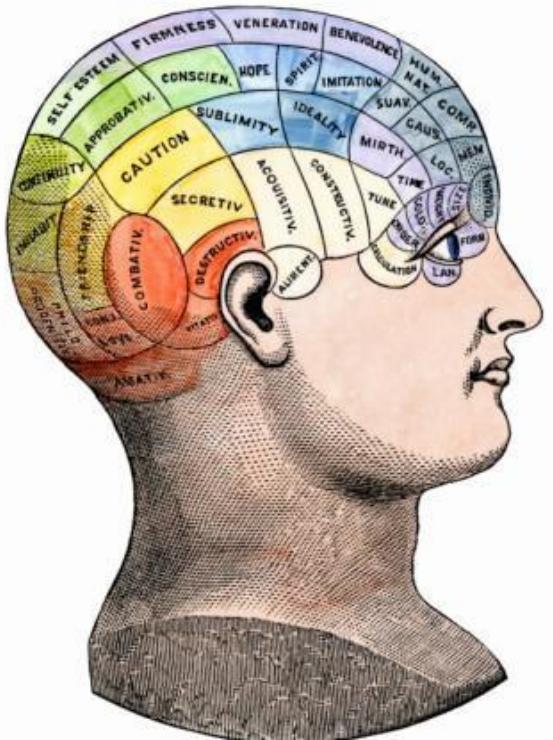
AI: That's a great question. The computer is fed a large amount of data that shows different examples of the task being performed by a human expert. The computer then uses this data to identify patterns and develop a model of the behavior. This model allows the computer to predict the correct actions to take in a given situation. Over time, the computer can continue to refine its model and improve its performance. Does that help clarify things?

H: That does. Thank you!

Back to the Bigger Picture

Learning as the basis of intelligence

- Reinforcement learning = can reason about decision making
- Deep models = allows RL algorithms to learn and represent complex input-output mappings



What is missing?

How Much Information Does the Machine Need to Predict? Y LeCun

- "Pure" Reinforcement Learning (cherry)
 - ▶ The machine predicts a scalar reward given once in a while.
 - ▶ **A few bits for some samples**
- Supervised Learning (icing)
 - ▶ The machine predicts a category or a few numbers for each input
 - ▶ Predicting human-supplied data
 - ▶ **10→10,000 bits per sample**
- Unsupervised/Predictive Learning (cake)
 - ▶ The machine predicts any part of its input for any observed part.
 - ▶ Predicts future frames in videos
 - ▶ **Millions of bits per sample**
- (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)



Where does the *signal* come from?

- Yann LeCun's cake
 - Unsupervised or self-supervised learning
 - Model learning (predict the future)
 - Generative modeling of the world
 - Lots to do even before you accomplish your goal!
- Imitation & understanding other agents
 - We are social animals, and we have culture – for a reason!
- The giant value backup
 - All it takes is one +1
- All of the above

How should we answer these questions?

- Pick the right problems!
 - Ask: does this have a **chance** of solving an important problem?
 - Optimism in the face of uncertainty is a good exploration strategy!
- Don't be afraid to change the problem statement
 - Many of these challenges won't be met by iterating on existing benchmarks!
- Applications matter
 - Sometimes applying methods to realistic and challenging real-world domains can teach us a lot about the important things that are missing
 - RL has a long history of disregarding this fact
- Think big and start small