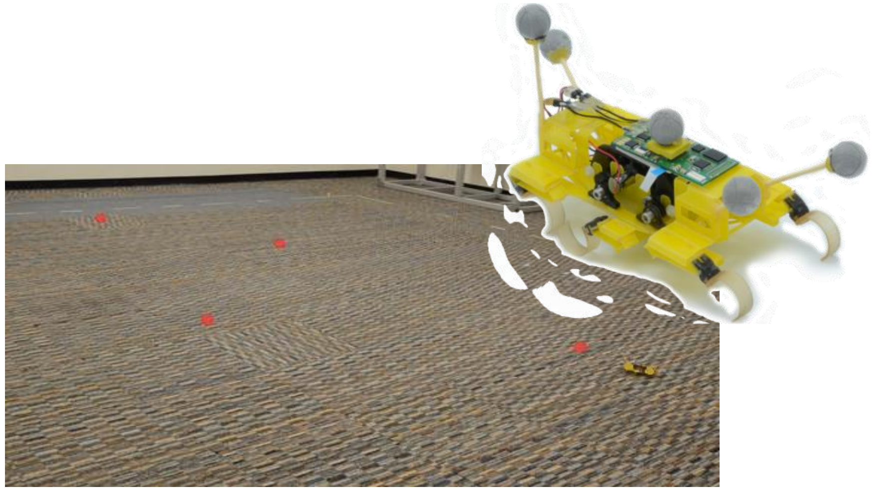
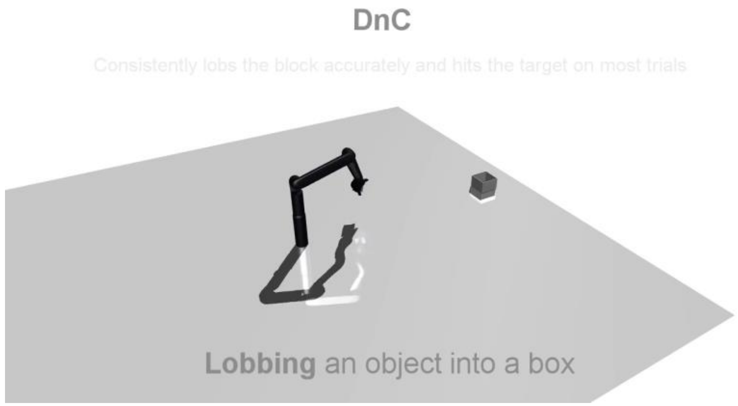
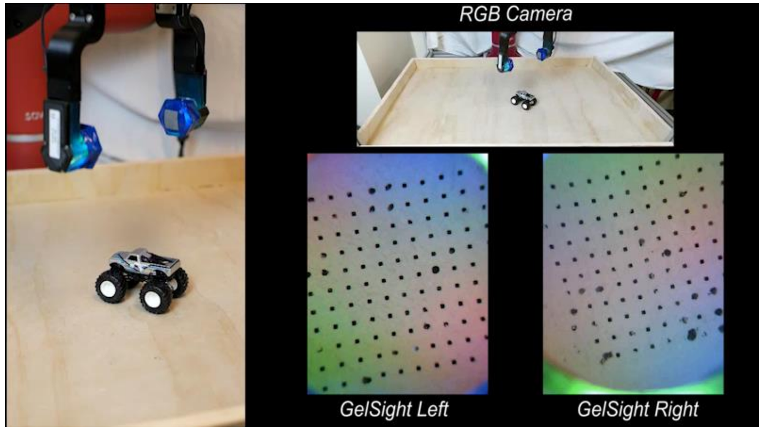
Deep Reinforcement Learning

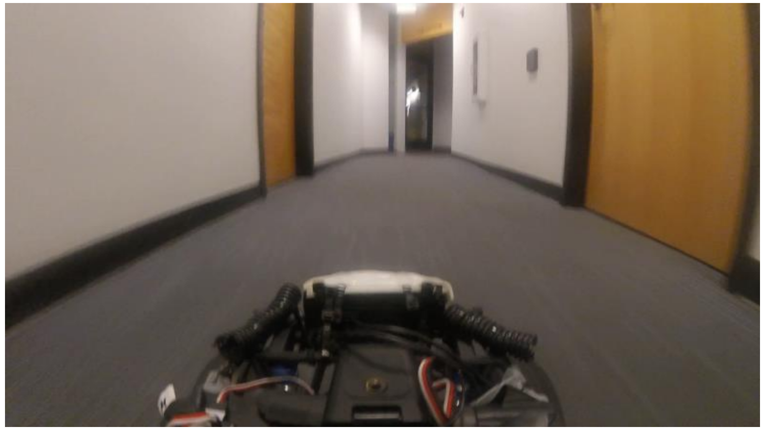
Lecture 1

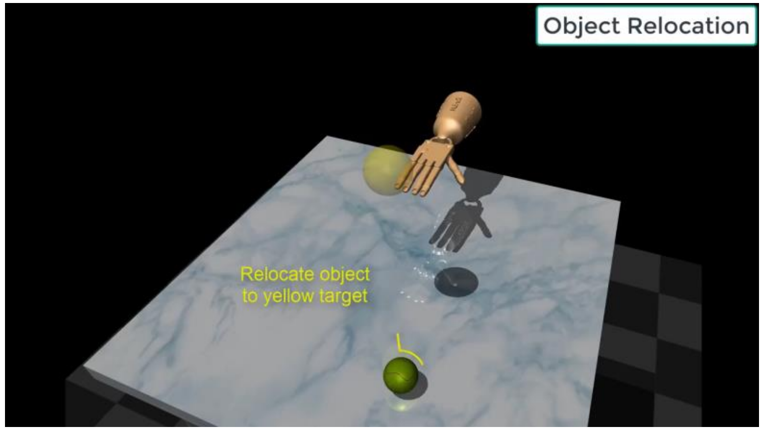
Sergey Levine

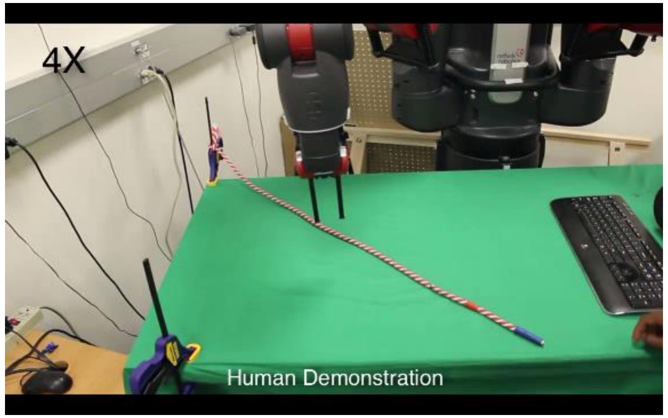






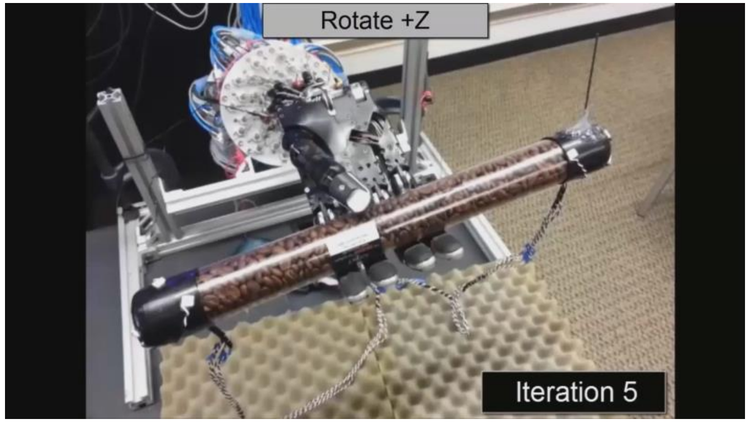




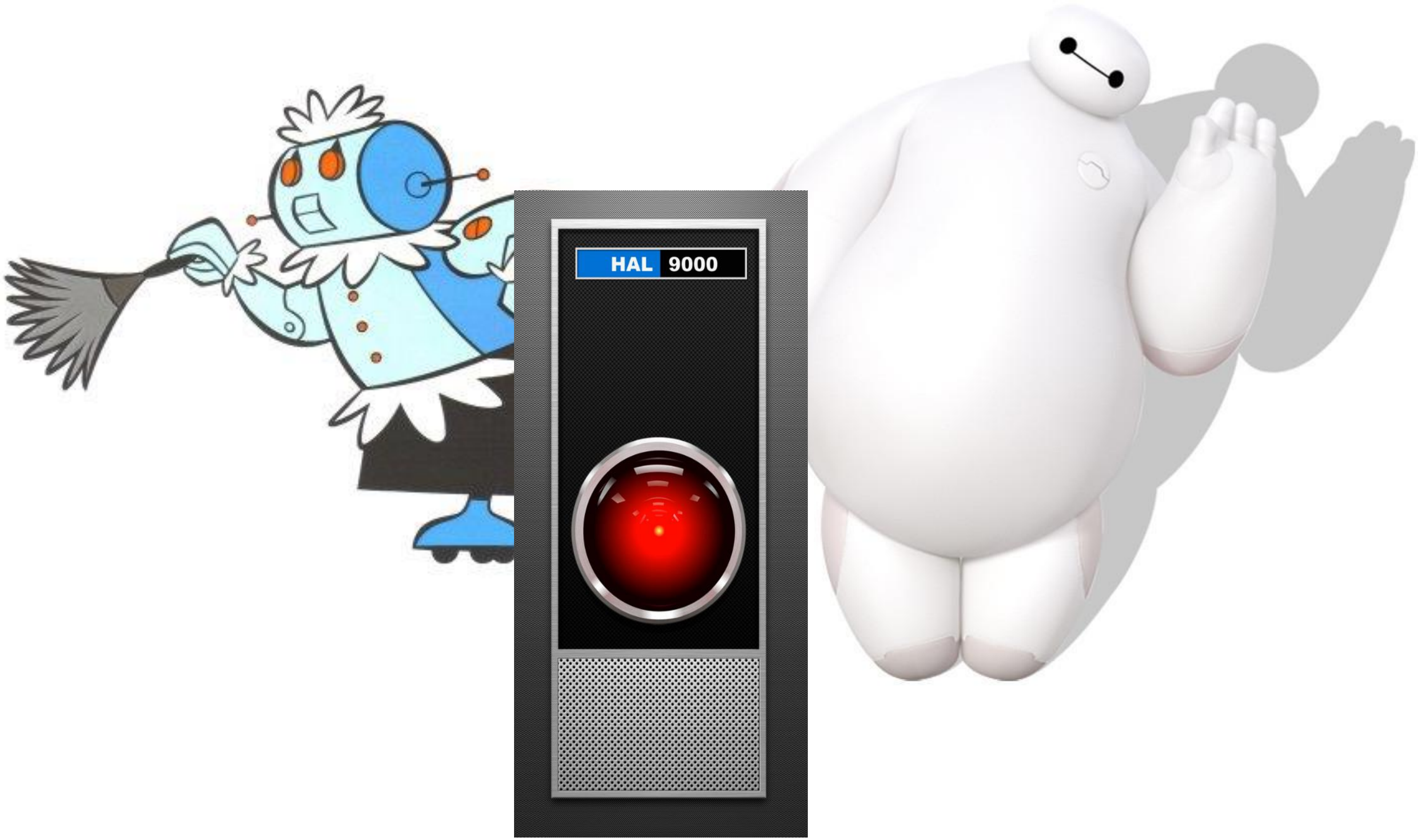


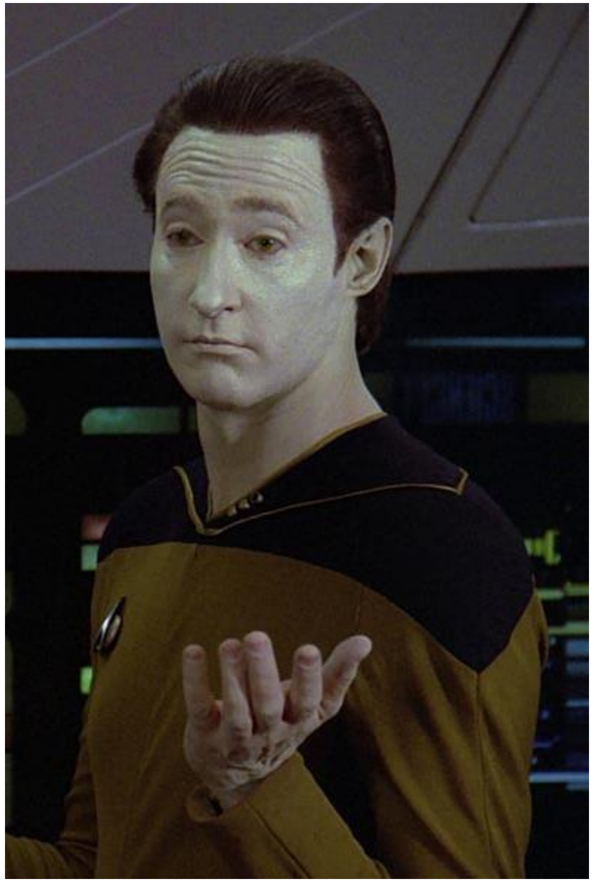
Real-World Experiments

Not accounting for uncertainty
(higher-speed collisions)



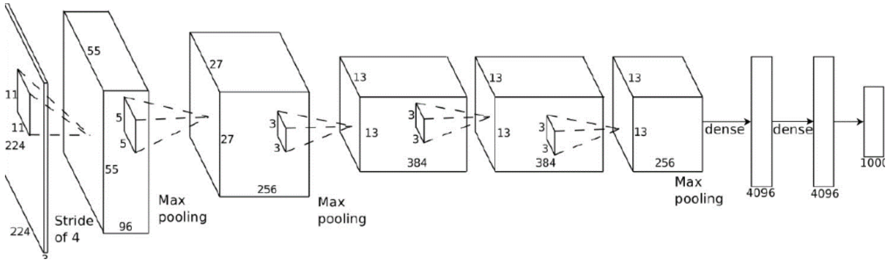
How do we build intelligent machines?

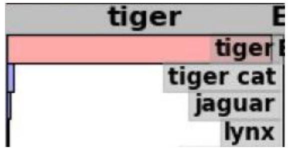


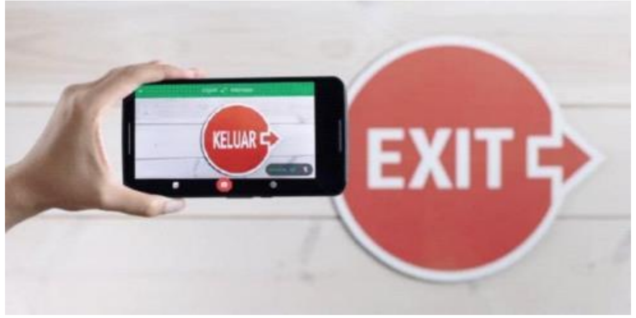


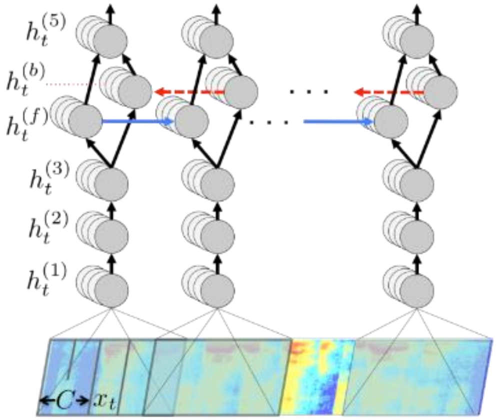
Intelligent machines must be able to adapt

Deep learning helps us handle *unstructured environments*



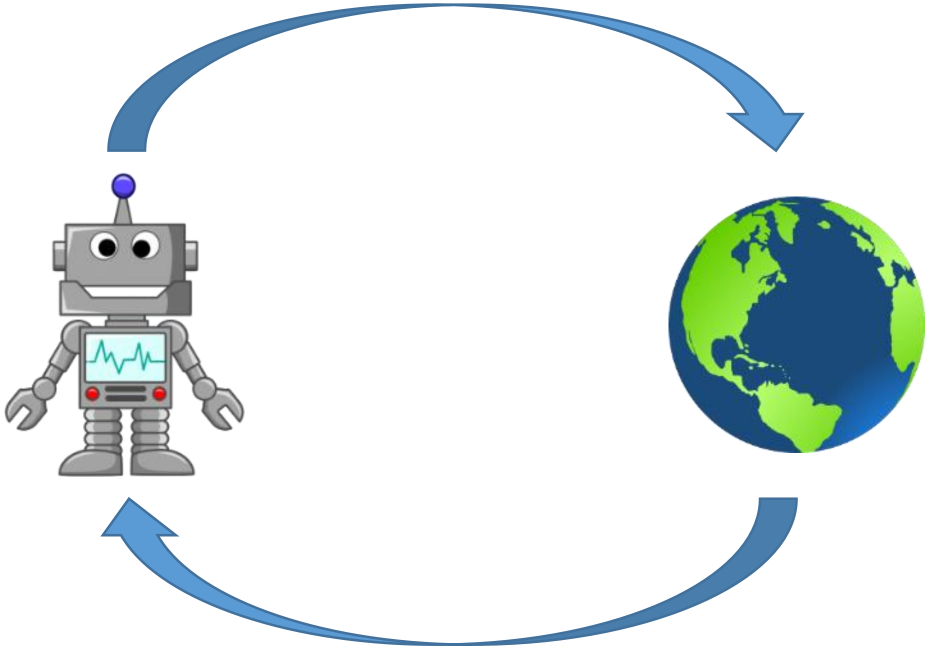






Reinforcement learning provides a formalism for *behavior*

decisions (actions)



consequences
observations
rewards

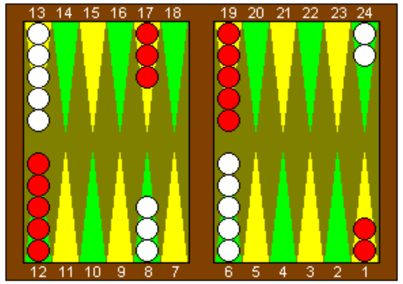
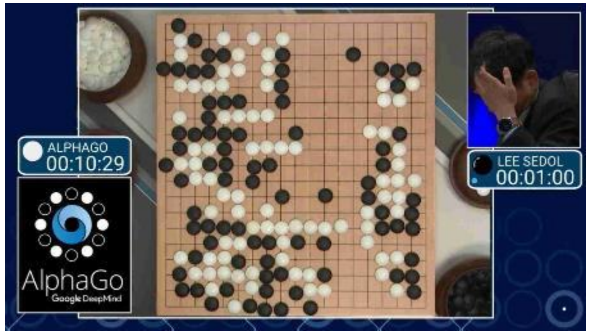
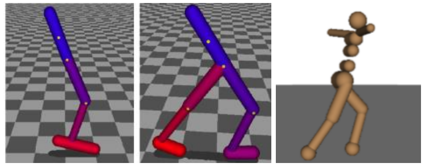
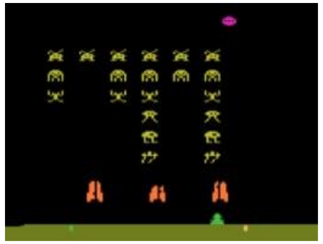


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference,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

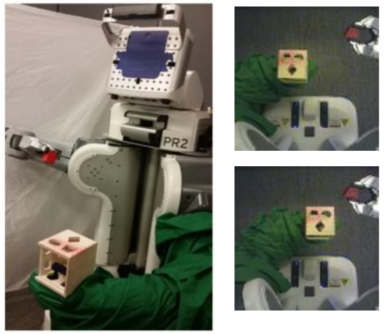




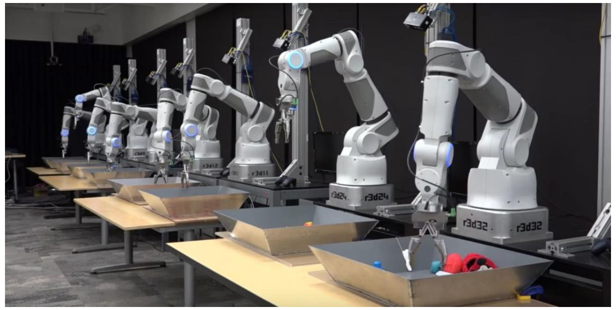
Schulman et al. '14 & '15



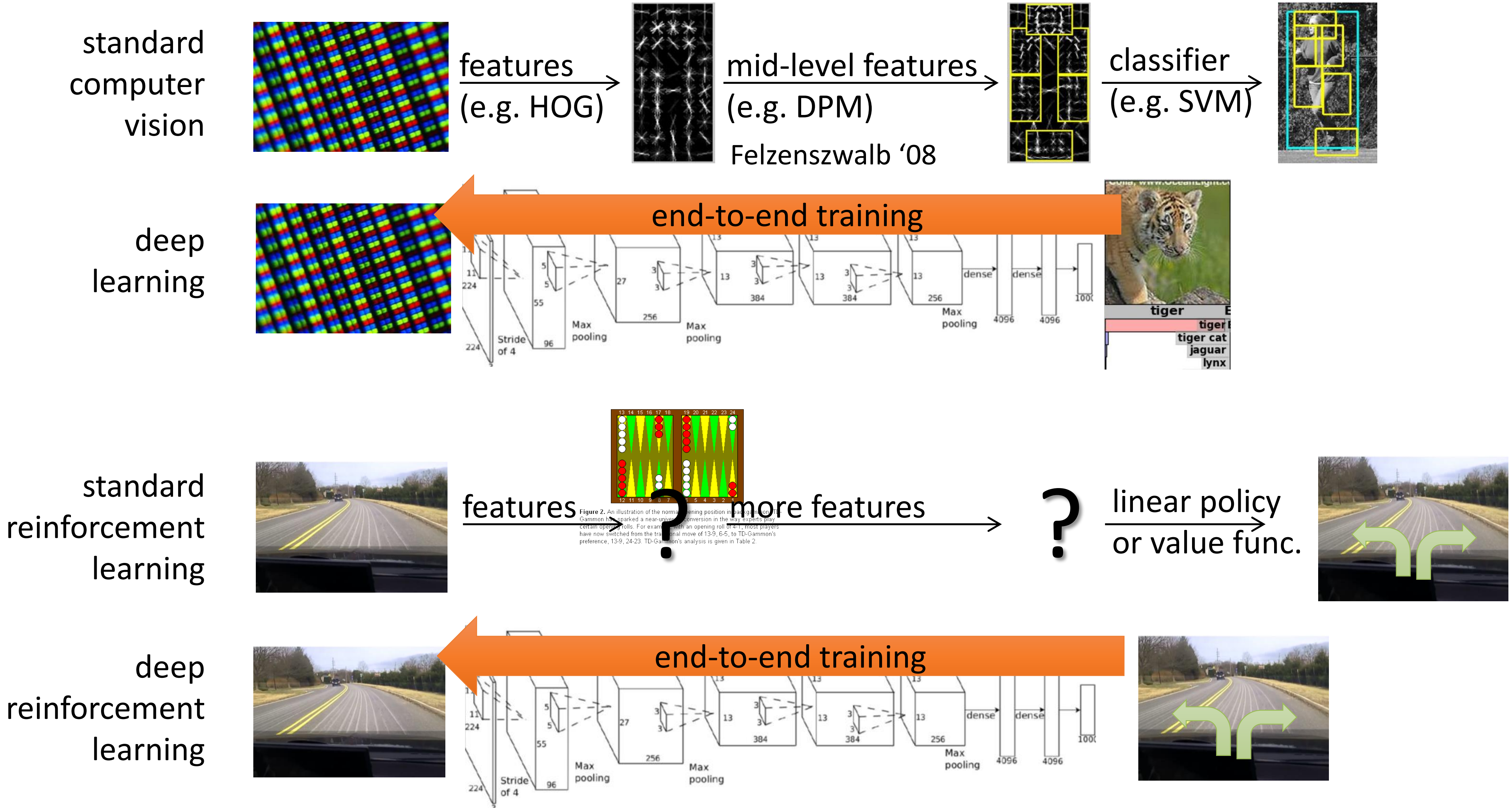
Mnih et al. '13



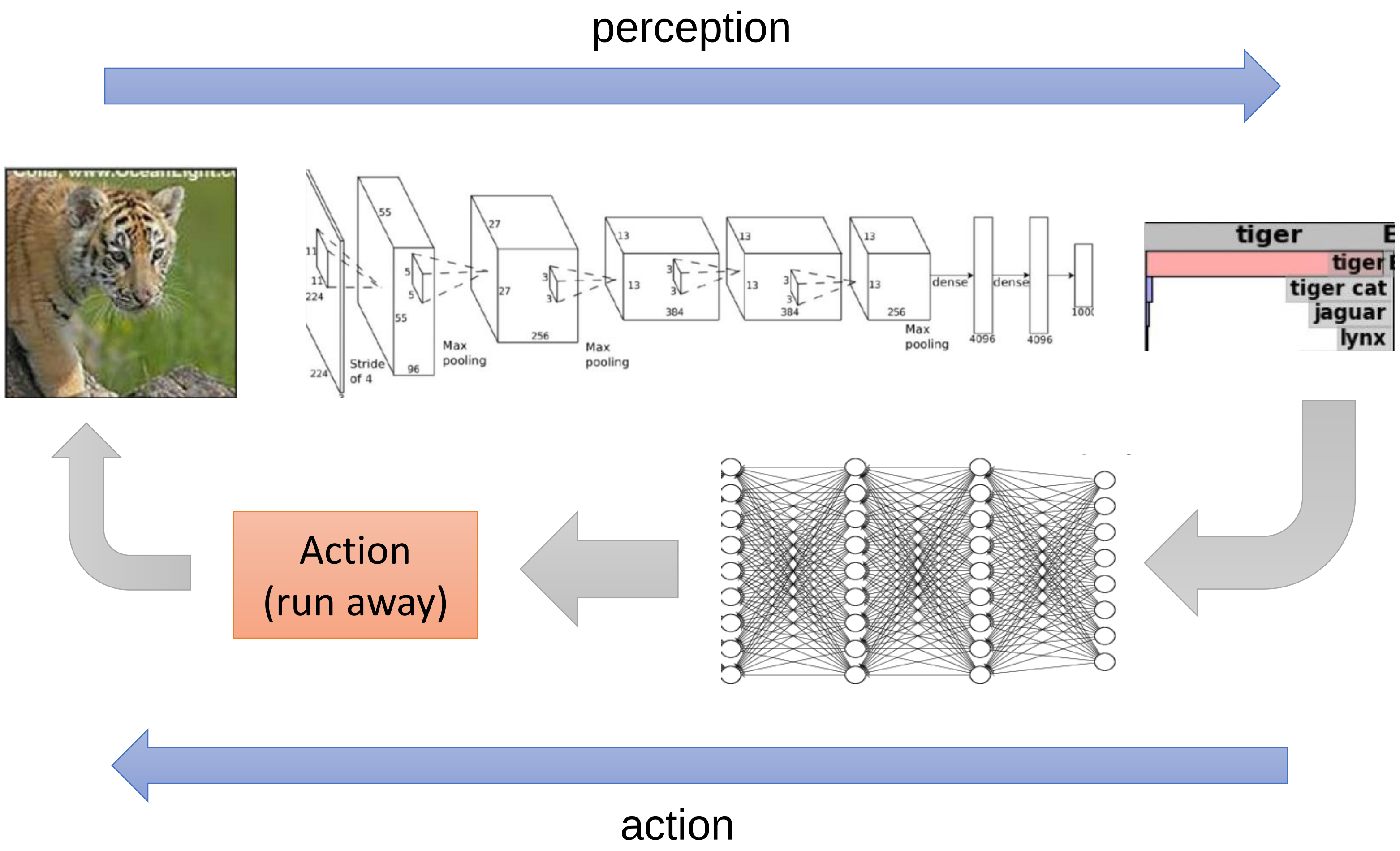
Levine*, Finn*, et al. '16

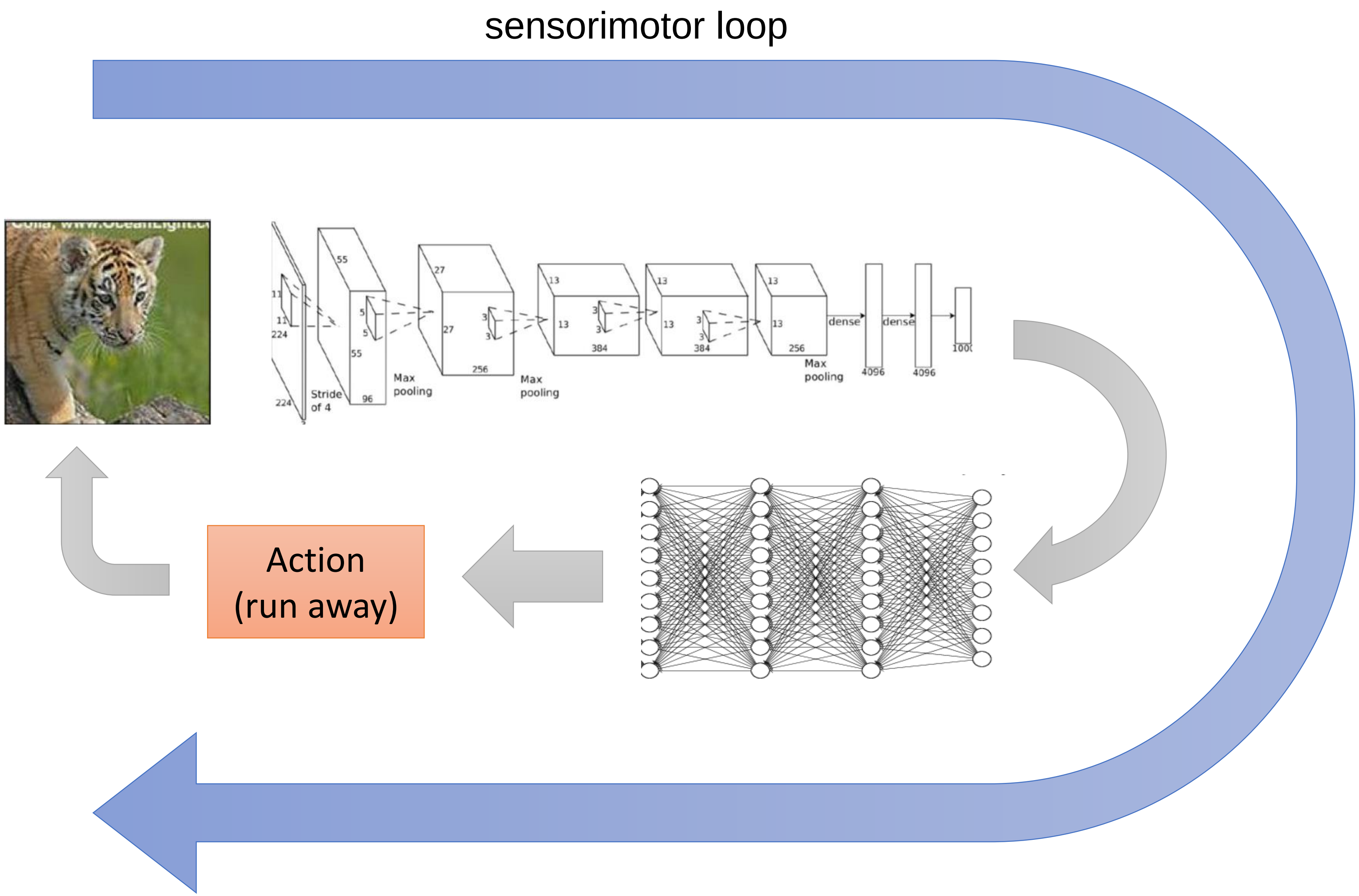


What is deep RL, and why should we care?

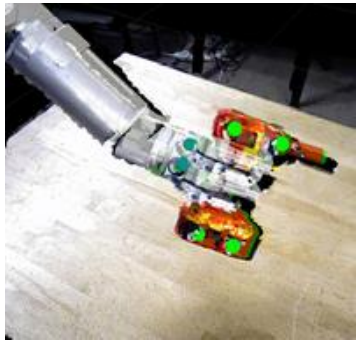


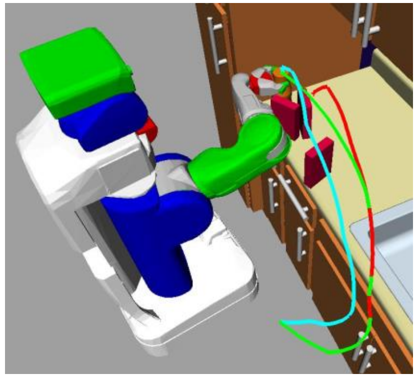
What does end-to-end learning mean for sequential decision making?


perception
tiger
tiger cat
jaguar
lynx
Stride of 4
Max pooling
Max pooling
Max pooling
dense
dense
Action
(run away)
action


sensorimotor loop
Stride of 4
Max pooling
Max pooling
Max pooling
dense
dense
Action
(run away)

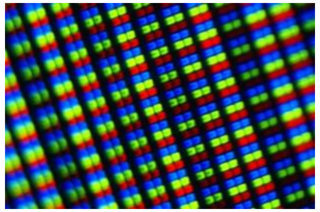
Example: robotics

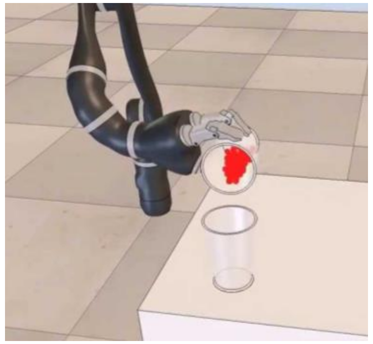


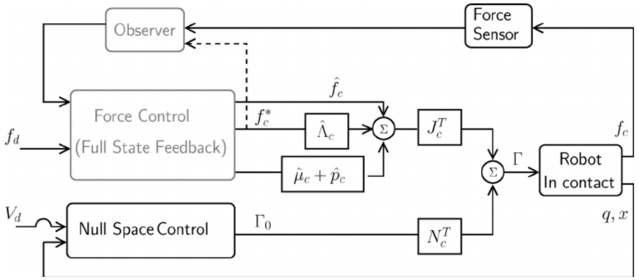


robotic control pipeline

observations → state estimation (e.g. vision) → modeling & prediction → planning → low-level control → controls

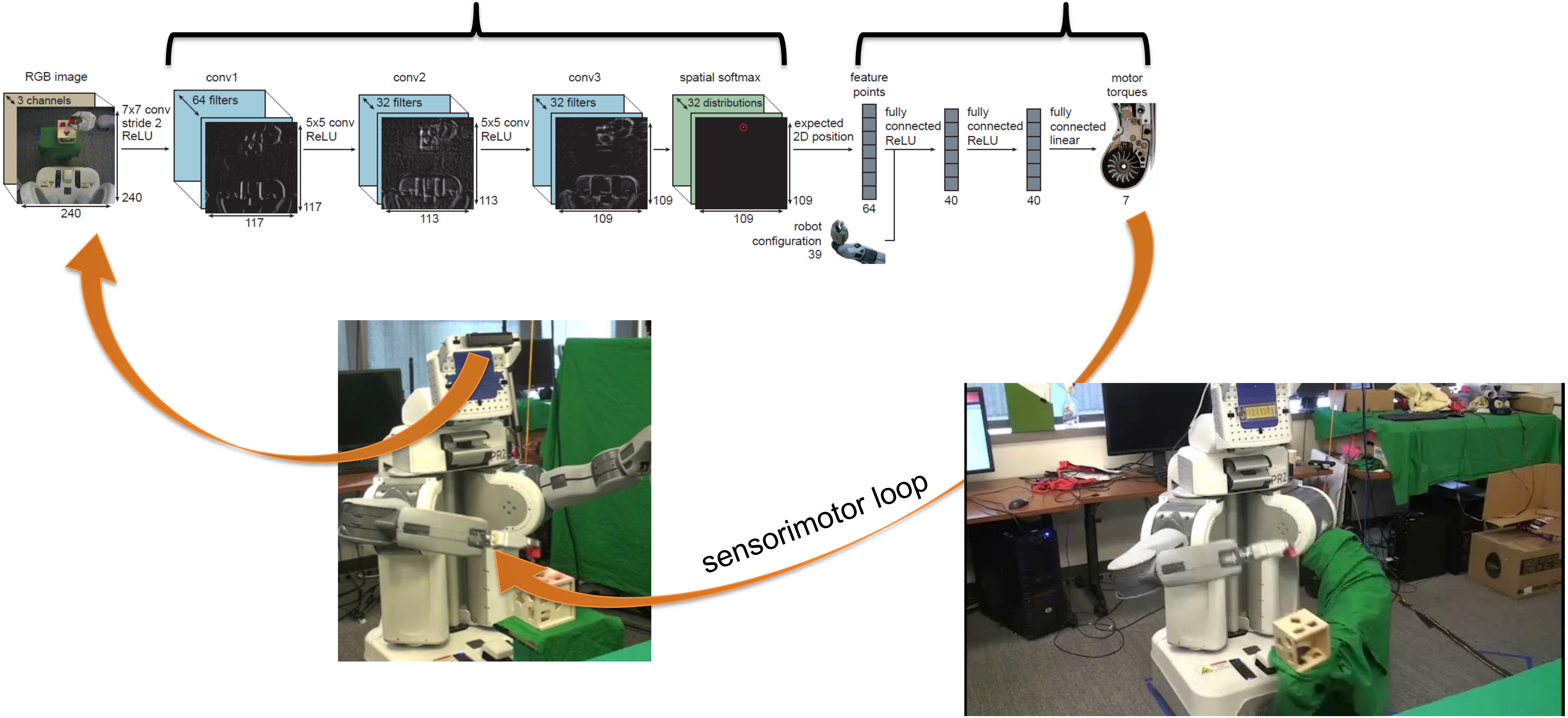






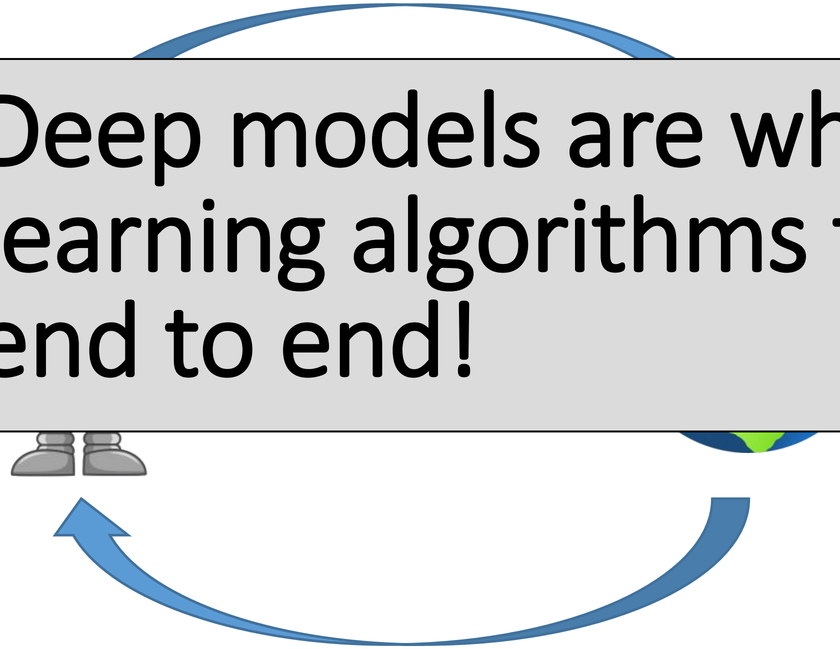
tiny, highly specialized
“visual cortex”

tiny, highly specialized
“motor cortex”



The reinforcement learning problem

decisions (actions)



consequences
observations
rewards

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

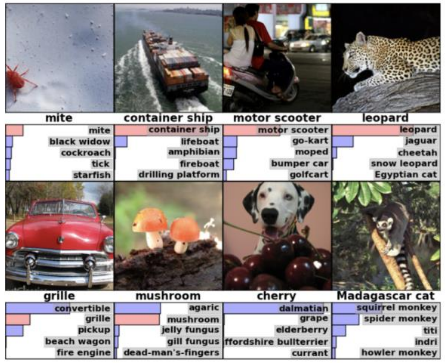
Deep models are what allow reinforcement learning algorithms to solve complex problems end to end!

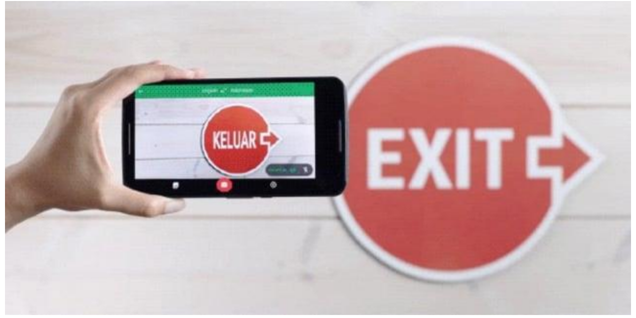
Observations: words in English
Rewards: BLEU score

The reinforcement learning problem **is** the AI problem!

When do we **not** need to worry about sequential decision making?

When your system is making single isolated decision, e.g. classification, regression
When that decision does not affect future decisions



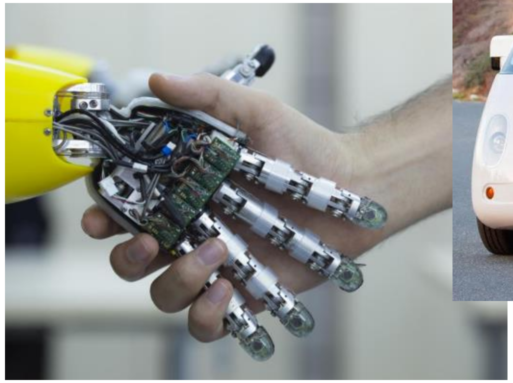


When **should** we worry about sequential decision making?

Limited supervision: you know **what** you want, but not **how** to get it

Actions have consequences

Common Applications



robotics

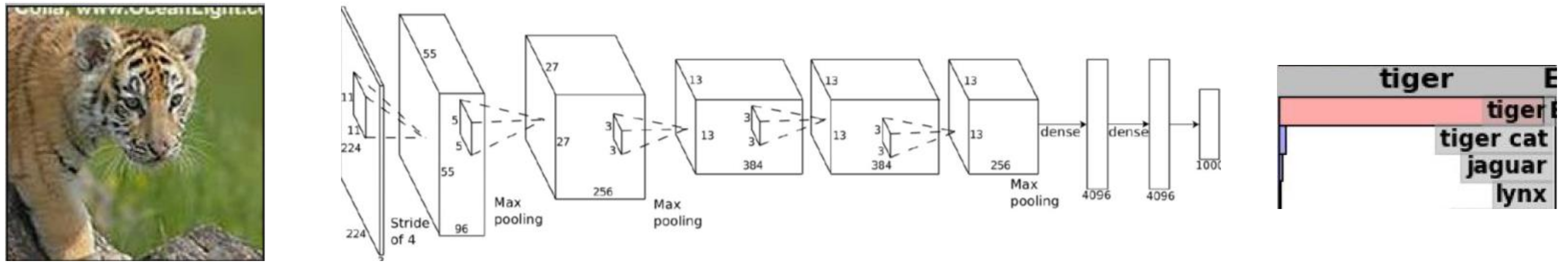
autonomous driving

language & dialogue (structured prediction)

business operations

finance

Why should we study this **now**?



1. Advances in deep learning

2. Advances in reinforcement learning

3. Advances in computational capability

Why should we study this **now**?

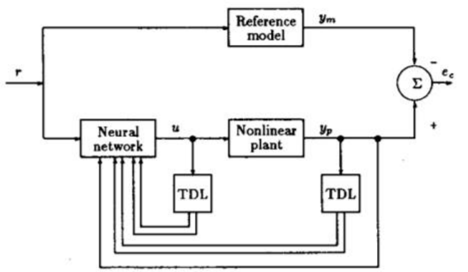
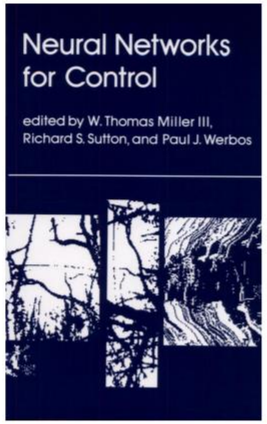
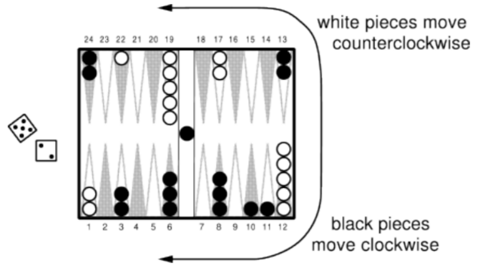


Fig. 21. Direct adaptive control of nonlinear plants using neural networks.





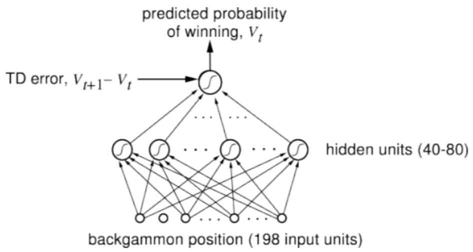


Table 11.1: Summary of TD-Gammon Results

Program	Hidden	Training	Opponents	Results
	Units	Games		
TD-Gam 0.0	40	300,000	other programs	tied for best
TD-Gam 1.0	80	300,000	Robertie, Magriel, ...	−13 pts / 51 games
TD-Gam 2.0	40	800,000	various Grandmasters	−7 pts / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	−1 pt / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 pts / 20 games

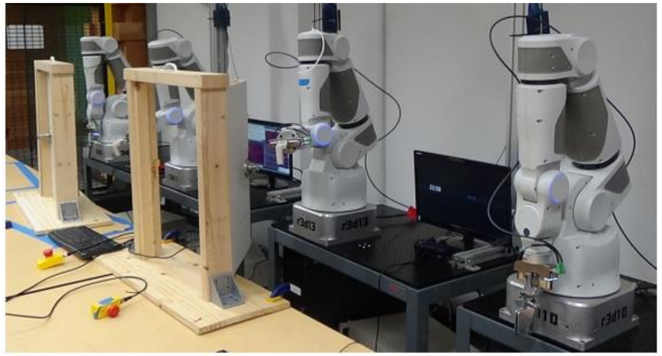
Tesauro, 1995

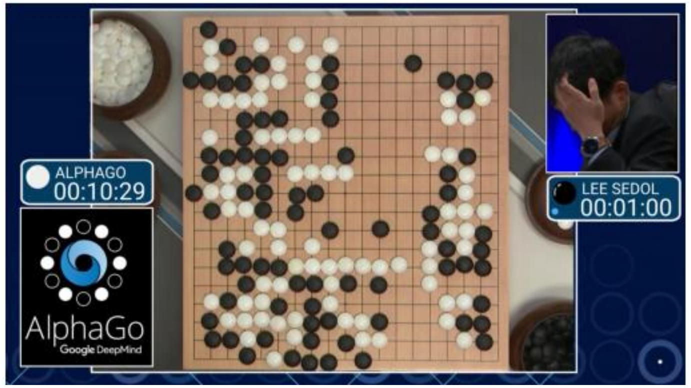
This dissertation demonstrates how we can possibly overcome the slow learning problem and tackle non-Markovian environments, making reinforcement learning more practical for realistic robot tasks:

- Reinforcement learning can be naturally integrated with artificial neural networks to obtain high-quality generalization, resulting in a significant learning speedup. Neural networks are used in this dissertation, and they generalize effectively even in the presence of noise and a large number of binary and real-valued inputs.
- Reinforcement learning agents can save many learning trials by using an action model, which can be learned on-line. With a model, an agent can mentally experience the effects of its actions without actually executing them. Experience replay is a simple technique that implements this idea, and is shown to be effective in reducing the number of action executions required.
- Reinforcement learning agents can take advantage of instructive training instances provided by human teachers, resulting in a significant learning speedup. Teaching can also help learning agents avoid local optima during the search for optimal control. Simulation experiments indicate that even a small amount of teaching can save agents many learning trials.
- Reinforcement learning agents can significantly reduce learning time by hierarchical learning— they first solve elementary learning problems and then combine solutions to the elementary problems to solve a complex problem. Simulation experiments indicate that a robot with hierarchical learning can solve a complex problem, which otherwise is hardly solvable within a reasonable time.
- Reinforcement learning agents can deal with a wide range of non-Markovian environments by having a memory of their past. Three memory architectures are discussed. They work reasonably well for a variety of simple problems. One of them is also successfully applied to a nontrivial non-Markovian robot task.

L.-J. Lin, “Reinforcement learning for robots using neural networks.” 1993

Why should we study this **now**?





Atari games:

Q-learning:

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. “Playing Atari with Deep Reinforcement Learning”. (2013).

Policy gradients:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. “Trust Region Policy Optimization”. (2015).
V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. “Asynchronous methods for deep reinforcement learning”. (2016).

Real-world robots:

Guided policy search:

S. Levine*, C. Finn*, T. Darrell, P. Abbeel. “End-to-end training of deep visuomotor policies”. (2015).

Q-learning:

S. Gu*, E. Holly*, T. Lillicrap, S. Levine. “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates”. (2016).

Beating Go champions:

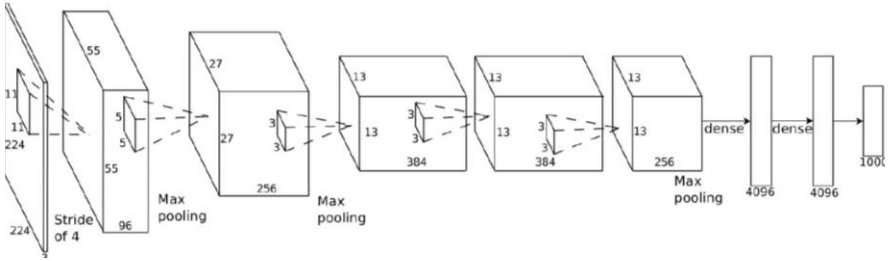
Supervised learning + policy gradients + value functions + Monte Carlo tree search:

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. “Mastering the game of Go with deep neural networks and tree search”. Nature (2016).

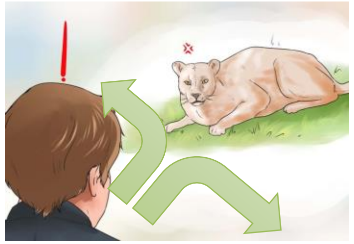
Reinforcement Learning by Gradient Descent

Terminology & notation

$\mathbf{o}_t$



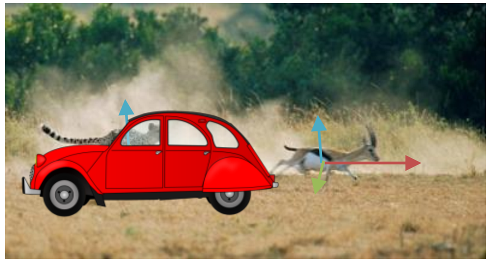
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



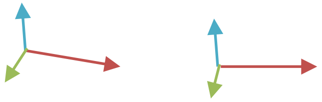
$\mathbf{a}_t$

$\mathbf{s}_t$ – 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)

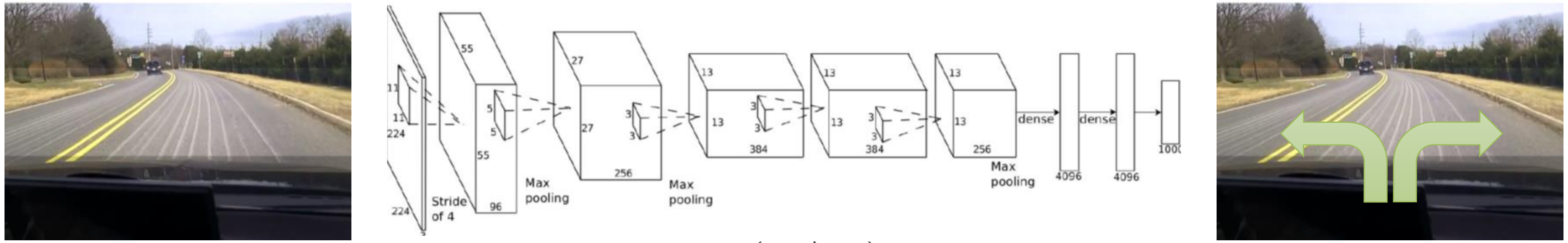


$\mathbf{o}_t$ – observation



$\mathbf{s}_t$ – state

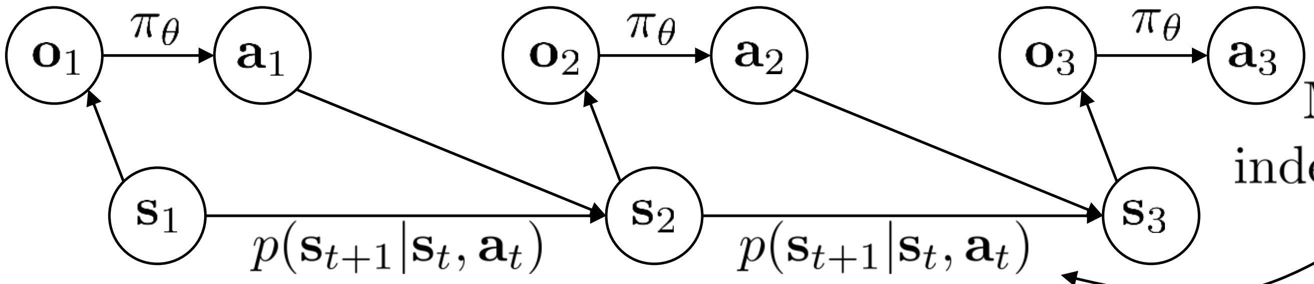
Terminology & notation



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

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

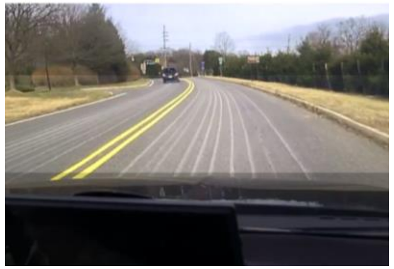
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



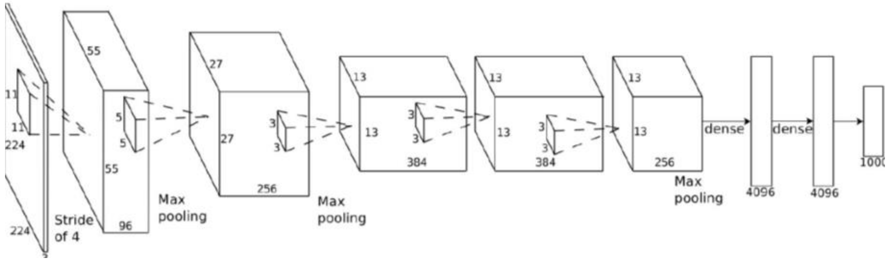
Markov property
independent of $\mathbf{s}_{t-1}$

Images: Bojarski et al. '16, NVIDIA

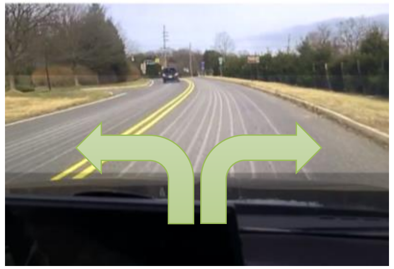
Imitation Learning



$\mathbf{o}_t$



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



$\mathbf{a}_t$

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

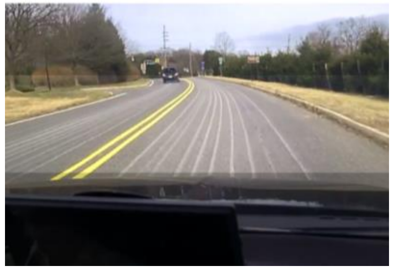
training data

supervised learning

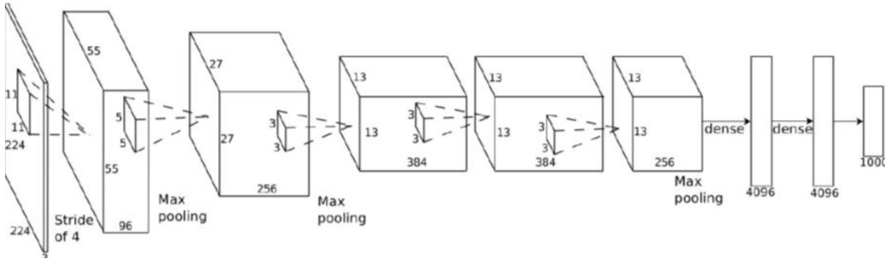
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

Images: Bojarski et al. '16, NVIDIA

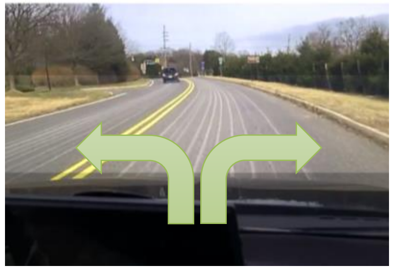
Reward functions



$\mathbf{o}_t$



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



$\mathbf{a}_t$

which action is better or worse?

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

tells us which states and actions are better

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

high reward

low reward

Definitions

Andrey Markov

Markov chain

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

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

$\mathcal{T}$ – transition operator $p(s_{t+1}|s_t)$

why "operator"? let $\mu_{t,i} = p(s_t = i)$ $\vec{\mu}_t$ is a vector of probabilities

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

$$\begin{tikzcd} \mathbf{s}_1 \arrow[r, "p(\mathbf{s}_{t+1}|\mathbf{s}_t)"'] & \mathbf{s}_2 \arrow[r, "p(\mathbf{s}_{t+1}|\mathbf{s}_t)"'] & \mathbf{s}_3 \end{tikzcd}$$

Markov property
independent of $\mathbf{s}_{t-1}$

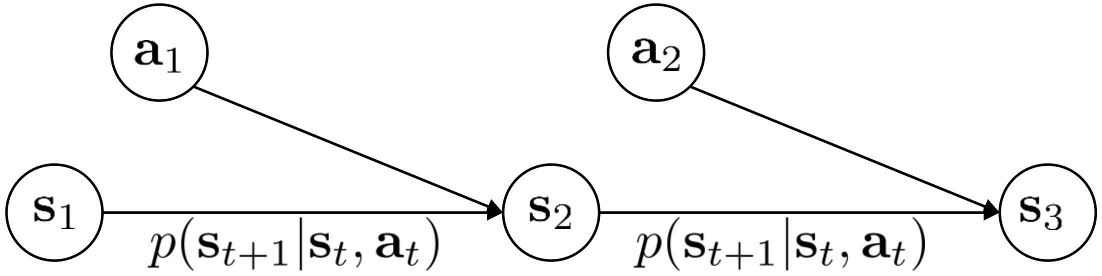
Definitions

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!)



Andrey Markov

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

$r(s_t, a_t)$ – reward

Andrey Markov

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)

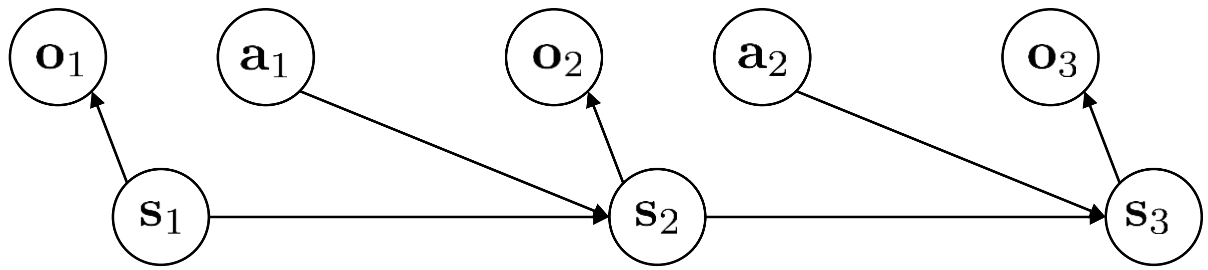
$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

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

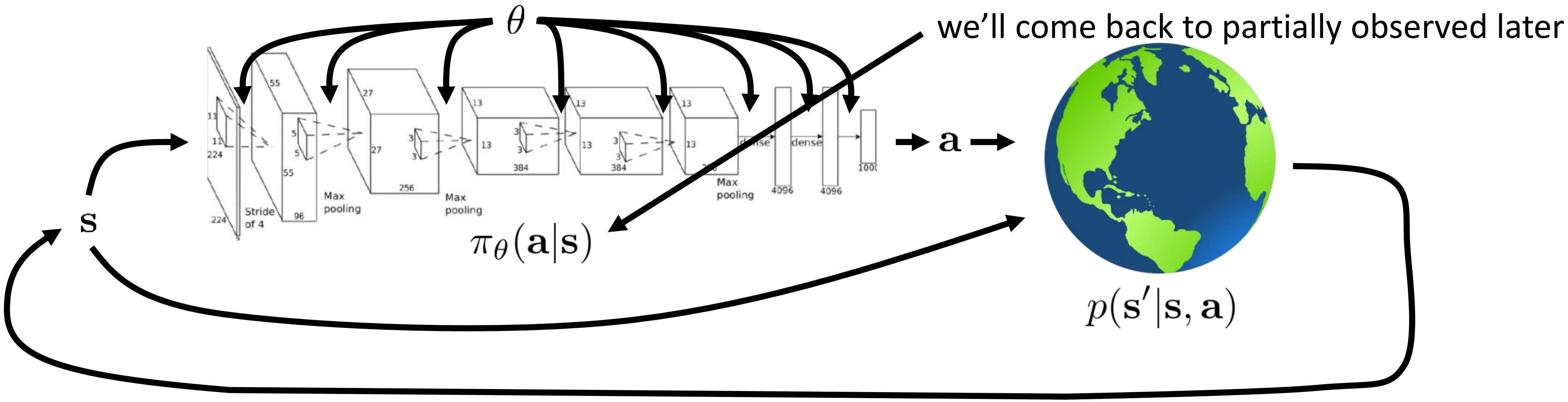
$\mathcal{T}$ – transition operator (like before)

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

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



The goal of reinforcement learning



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

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

The goal of reinforcement learning

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

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

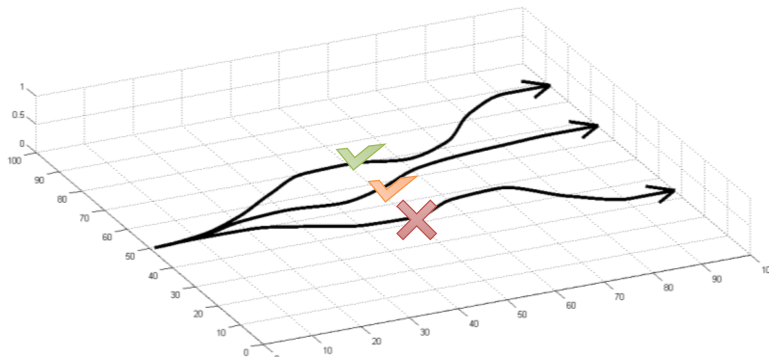
infinite horizon case

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

finite horizon case

Evaluating the objective

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



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

sum over samples from π_θ

Direct policy differentiation

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

a convenient identity

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau)\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

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

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

Direct policy differentiation

$$\theta^\star = \arg\max_\theta J(\theta)$$

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

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

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

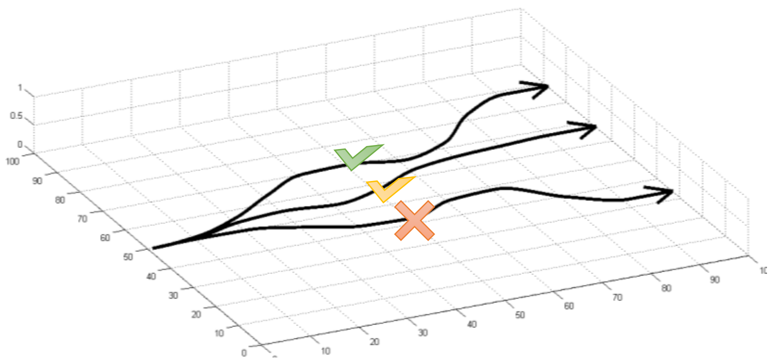
log of both sides

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

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

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

Evaluating the policy gradient

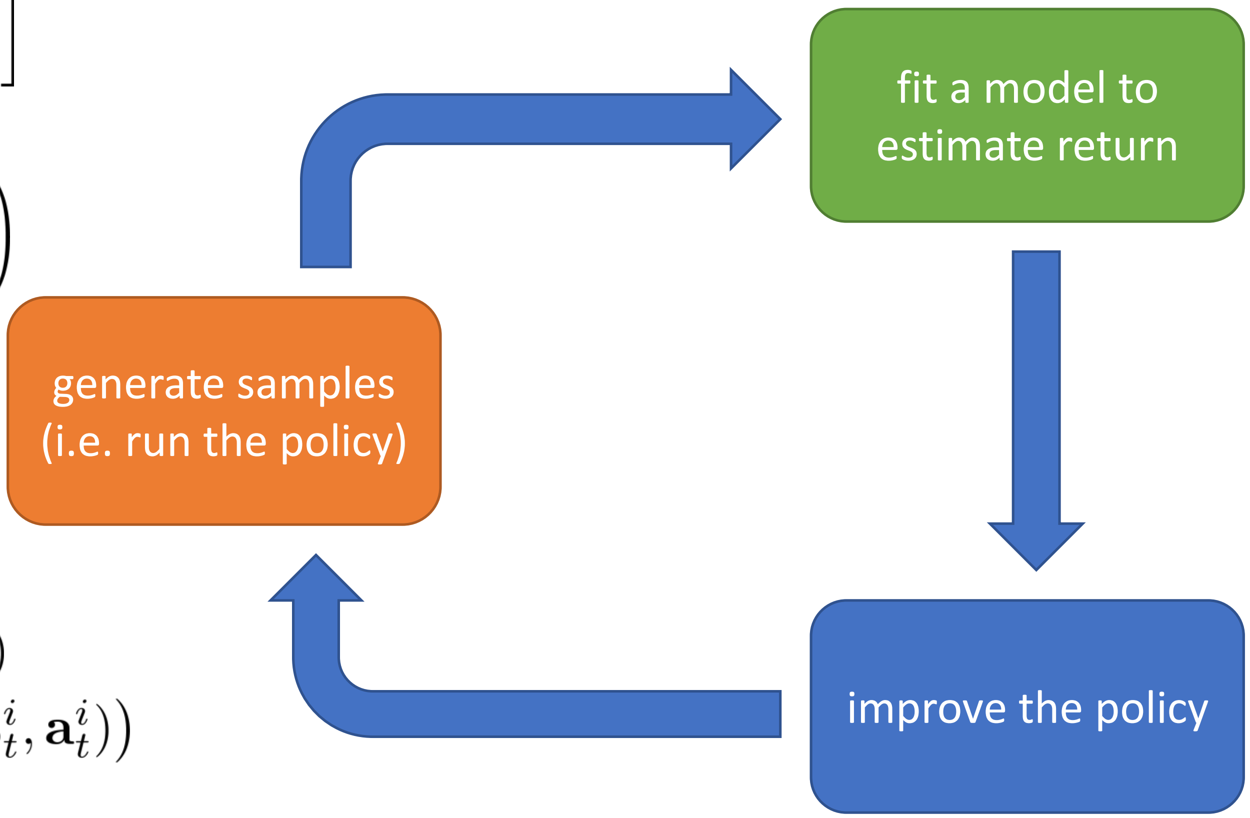


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

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

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

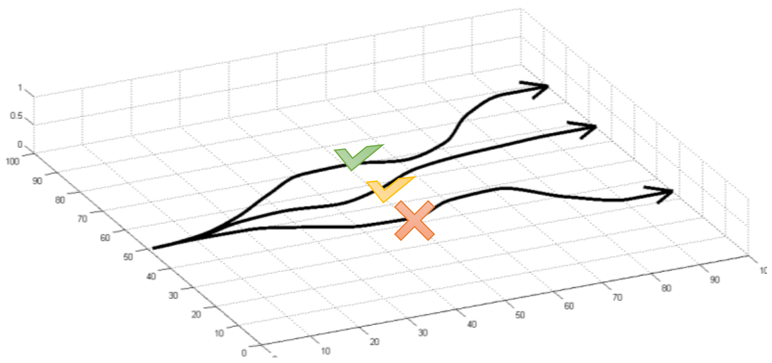
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$



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

Evaluating the policy gradient

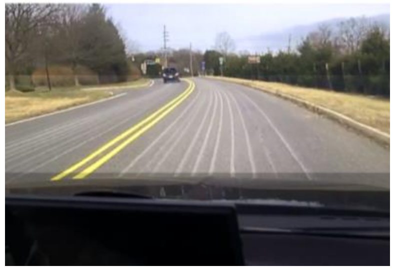


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

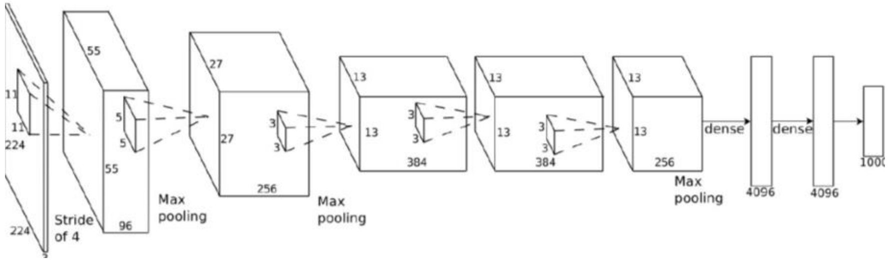
$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

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

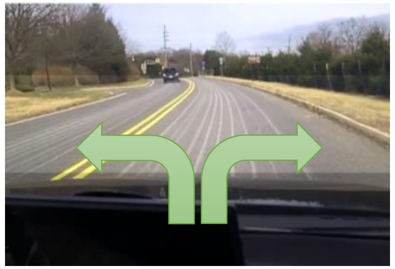
what is this?



$\mathbf{s}_t$



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

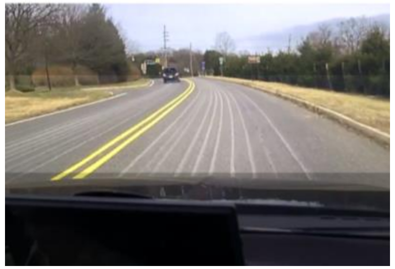


$\mathbf{a}_t$

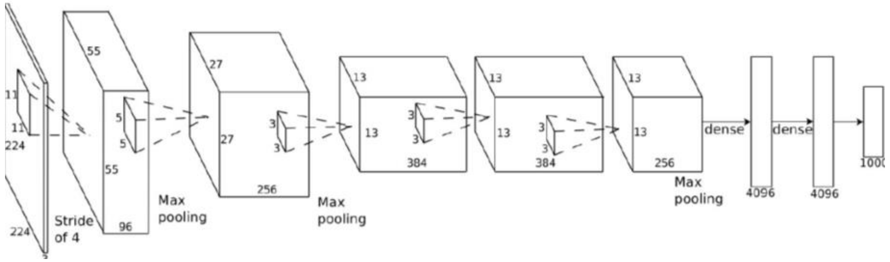
Comparison to maximum likelihood

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

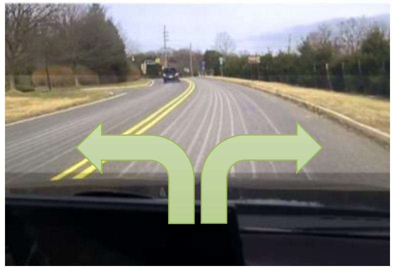
maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right)$



$\mathbf{s}_t$



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



$\mathbf{a}_t$

$\mathbf{s}_t$

$\mathbf{a}_t$

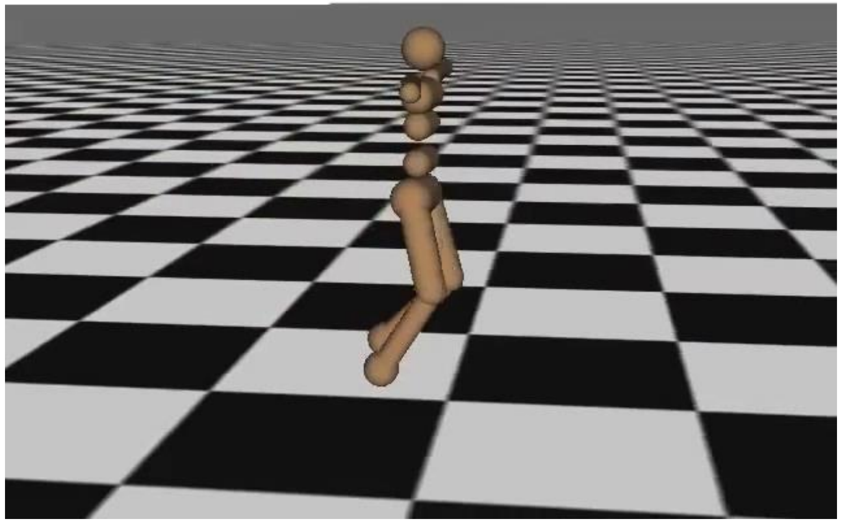
training data

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

What did we just do?

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

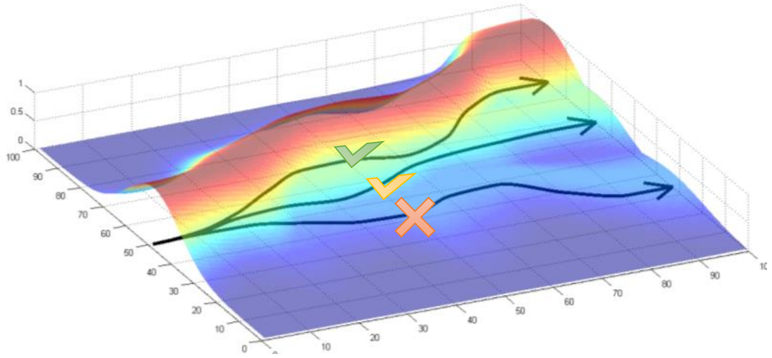
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T} \nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i)$$

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

good stuff is made more likely

bad stuff is made less likely

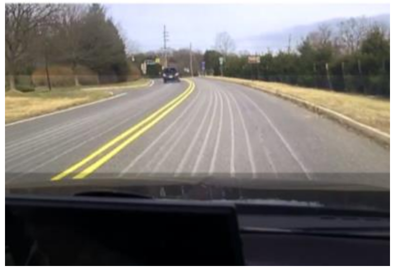
simply formalizes the notion of "trial and error"!

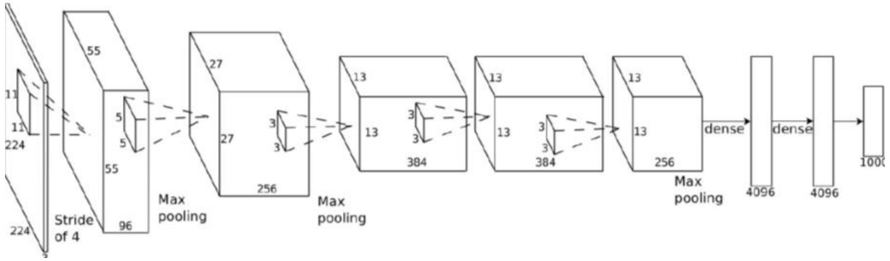


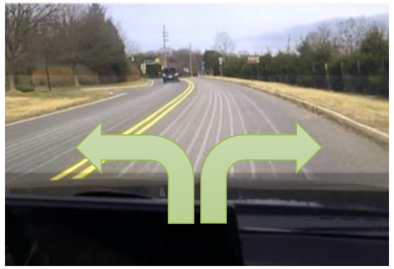
REINFORCE algorithm:

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

Partial observability







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

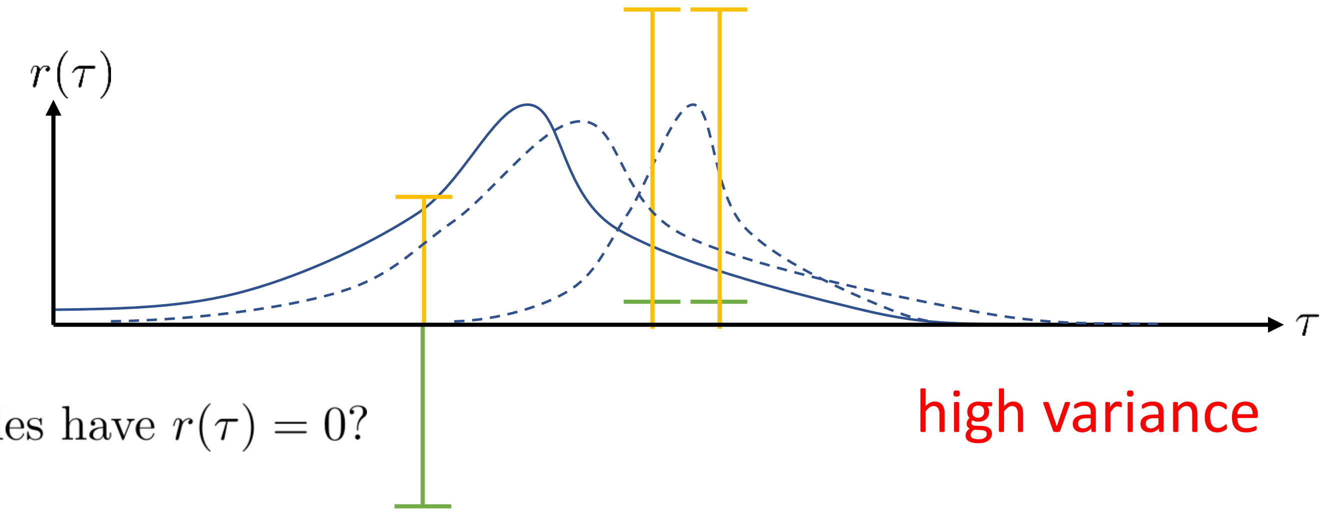
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{o}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{o}_{i,t},\mathbf{a}_{i,t})\right)$$

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 $r(\tau) = 0$?

high 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 $t < t'$

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

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

Baselines

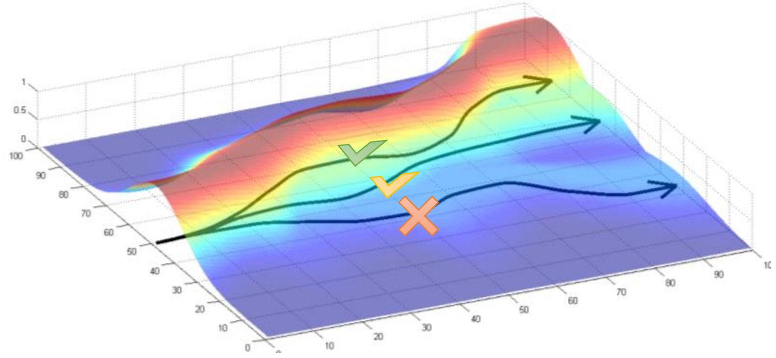
a convenient identity

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

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

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

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

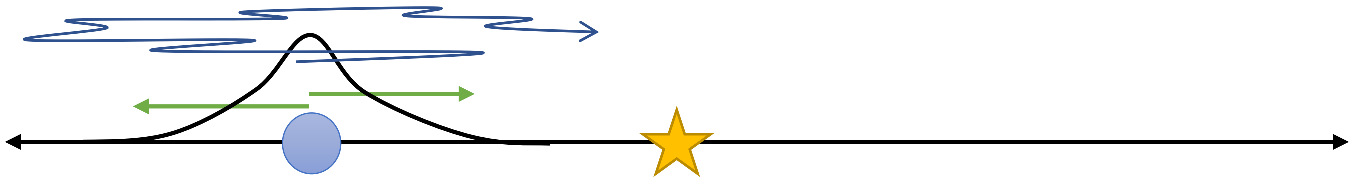


$$E[\nabla_\theta \log \pi_\theta(\tau) b] = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) b \, d\tau = \int \nabla_\theta \pi_\theta(\tau) b \, d\tau = b\nabla_\theta \int \pi_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

subtracting a baseline is *unbiased* in expectation!

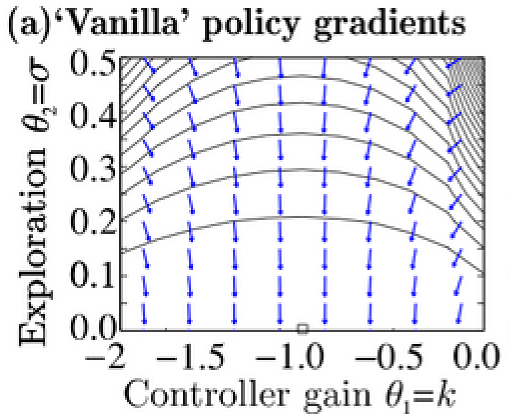
average reward is *not* the best baseline, but it's pretty good!

What *else* is wrong with the policy gradient?



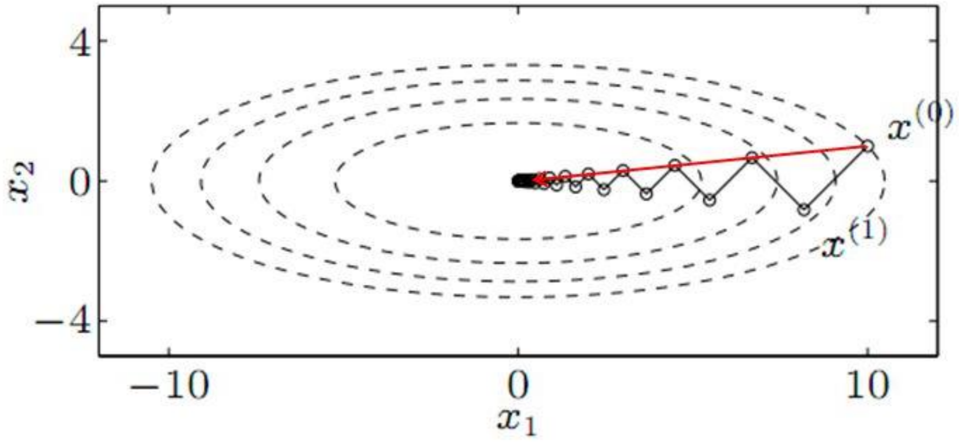
$$r(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^2 - \mathbf{a}_t^2$$

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



(image from Peters & Schaal 2008)

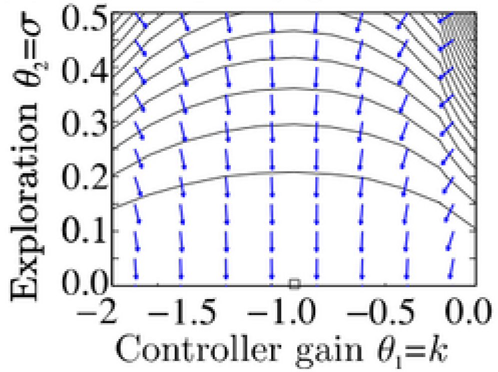
Essentially the same problem as this:



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!

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \text{ s.t. } \underbrace{\|\theta' - \theta\|^2 \leq \epsilon}_{\text{controls how far we go}}$$

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

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

parameterization-independent divergence measure

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

$$D_{\text{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^T \underbrace{\mathbf{F}}_{\text{Fisher-information matrix}} (\theta' - \theta)$$

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

can estimate with samples

Covariant/natural policy gradient

$$D_{\mathrm{KL}}(\pi_{\theta'} \| \theta_\pi) \approx (\theta' - \theta)^T \mathbf{F} (\theta' - \theta) \qquad \mathbf{F} = E_{\pi_\theta}[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \log \pi_\theta(\mathbf{a}|\mathbf{s})^T]$$

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \text{ s.t. } \|\theta' - \theta\|^2_{\mathbf{F}} \leq \epsilon \quad D(\pi_{\theta'}, \theta) \leq \epsilon$$

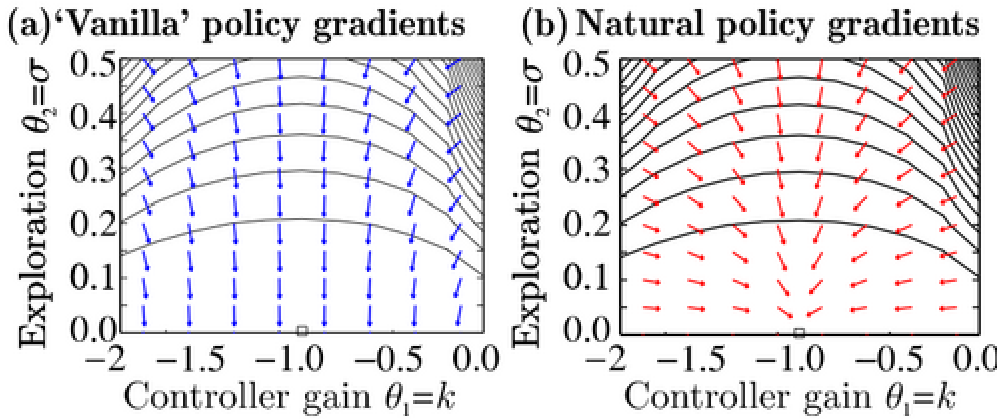
$$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$$

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 ths



(figure from Peters & Schaal 2008)

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

Deep Reinforcement Learning

Lecture 2

Sergey Levine

Last Time

- Deep reinforcement learning: deep networks + RL = end-to-end optimization of decision making and control
- Optimizing deep nets with SGD is great
- Let’s optimize the RL objective with SGD
  - This is called the policy gradient

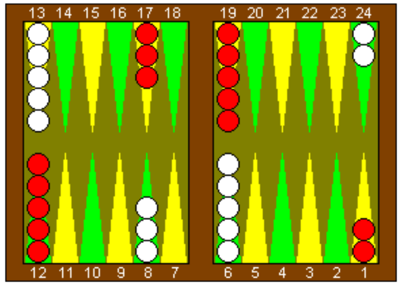
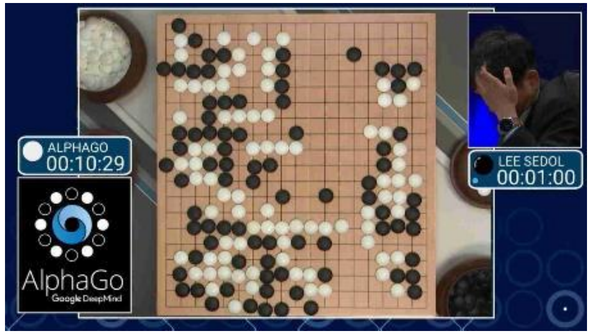
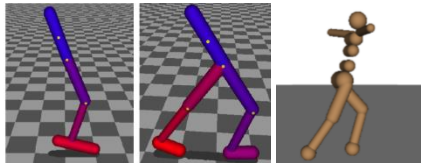
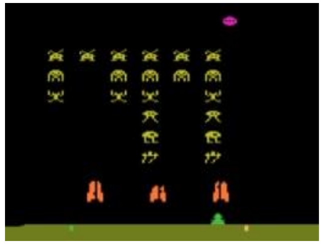


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference,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

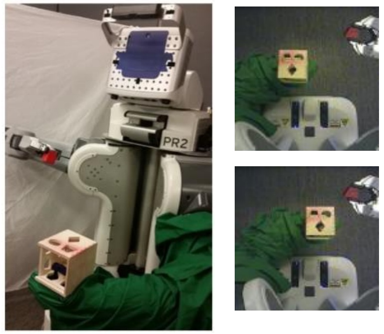




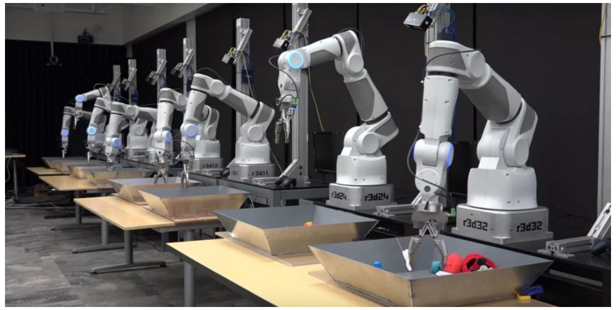
Schulman et al. ’14 & ‘15



Mnih et al. ‘13



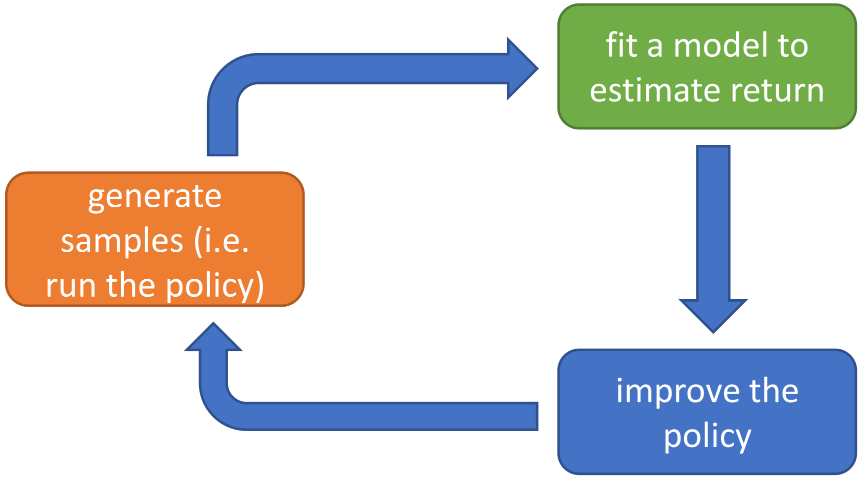
Levine*, Finn*, et al. ‘16



Review

- Policy gradient: directly differentiate RL objective and follow gradient
- Need to reduce variance to make it practical
- Often works best with natural gradient/trust region

$$\hat{Q}^\pi(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^{T} r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$



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

Policy gradient summary

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

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

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

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

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

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)
```

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

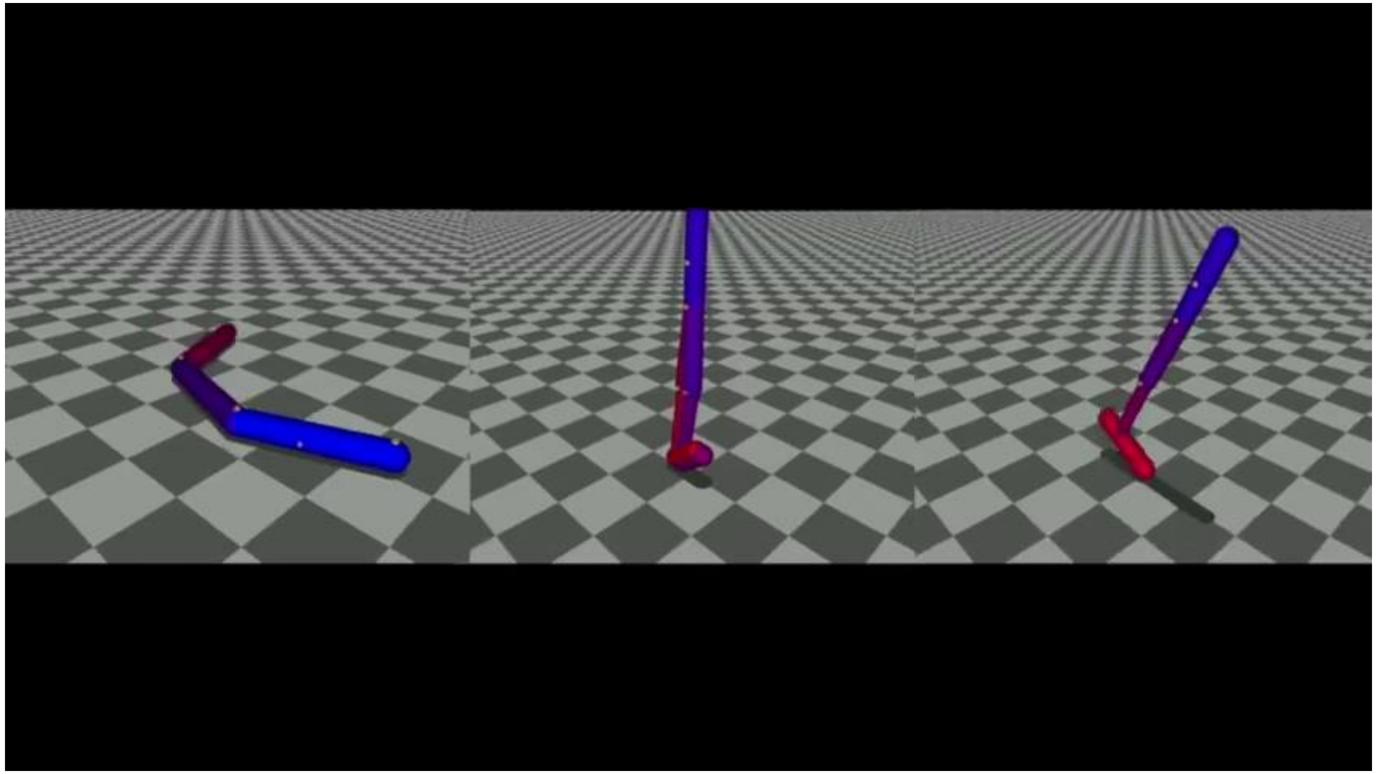
q_values

Policy gradient in practice

- Unfortunately only part of the story
  - Policy gradients have very high variance
  - Choosing step size is hard (much harder than regular SGD)
  - “Raw” (“vanilla”) policy gradients are hard to use
- What makes policy gradients easier to use?
  - Use natural gradient/trust region/etc.
  - Use automated step size adjustment (e.g., ADAM)
  - Reduce your variance
    - Use a baseline
    - Use a huge batch size
  - Use a critic (more on this next!)
- Key words to search for
  - TRPO (Schulman et al.) – natural gradient + trust region with value function estimator as control variate for variance reduction
  - PPO (Schulman et al.) – importance sampled policy gradient

Example: trust region policy optimization

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



Schulman, Levine, Moritz, Jordan, Abbeel. ‘15

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
  - 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
- Practice on your own!
  - Homework 2 here: https://github.com/berkeleydeeprlcourse/homework

Today’s Topics

- Actor-critic algorithm: reducing policy gradient variance using prediction
- Value-based algorithms: no more policy gradient, off-policy learning
- Model-based algorithms: control by predicting the future
- Open challenges and future directions

Today

- Actor-critic algorithm: reducing policy gradient variance using prediction
- Value-based algorithms: no more policy gradient, off-policy learning
- Model-based algorithms: control by predicting the future
- Open challenges and future directions

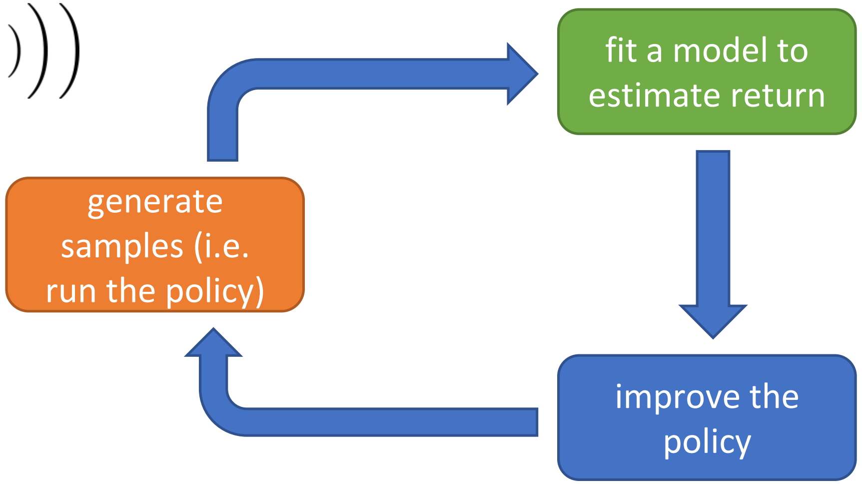
Improving the Gradient by Estimating the Value Function

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

$$\hat{Q}^\pi(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^T r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$



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

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

Improving the policy gradient

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

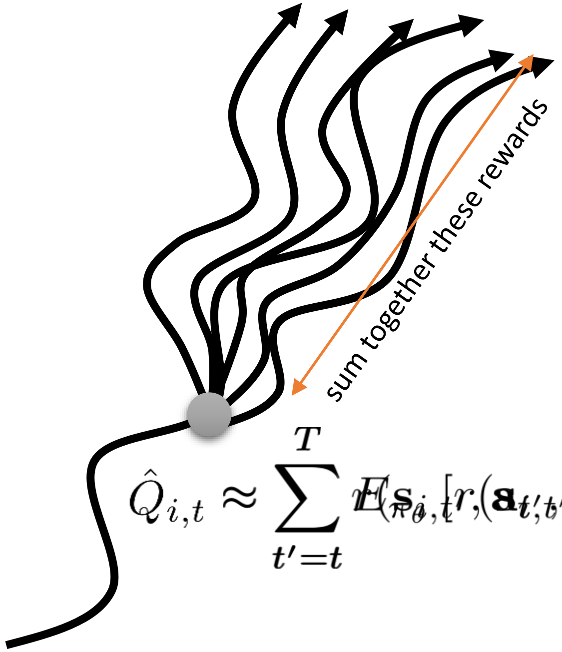
$$\hat{Q}_{i,t}$$

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

can we get a better estimate?

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

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



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

What about the baseline?

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

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

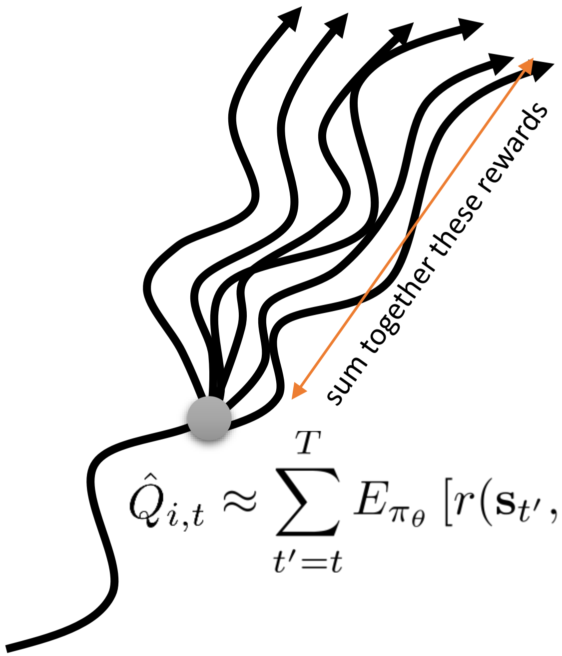
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})(Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t}))$$

b_t = average reward

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

average what?

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



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

State & state-action value functions

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

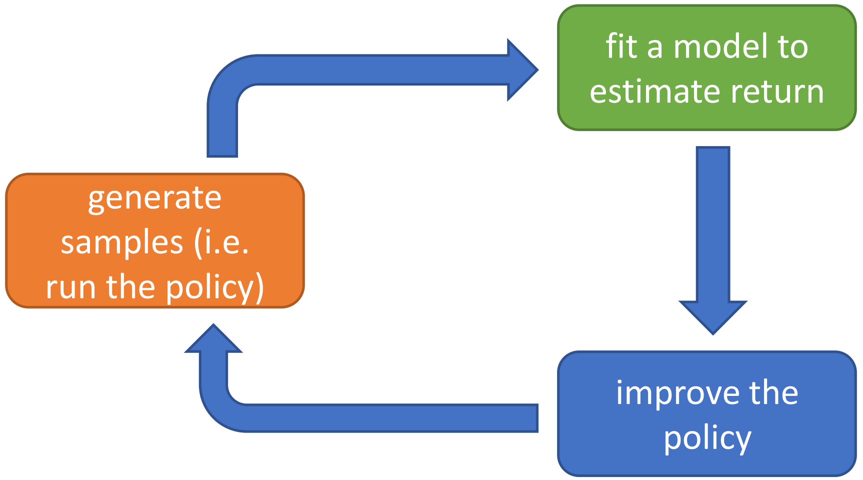
$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$: total reward from $\mathbf{s}_t$

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

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

the better this estimate, the lower the variance

fit Q^π, V^π, or A^π



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

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

unbiased, but high variance single-sample estimate

Value function fitting

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

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

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

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

fit *what* to *what*?

Q^π, V^π, A^π?

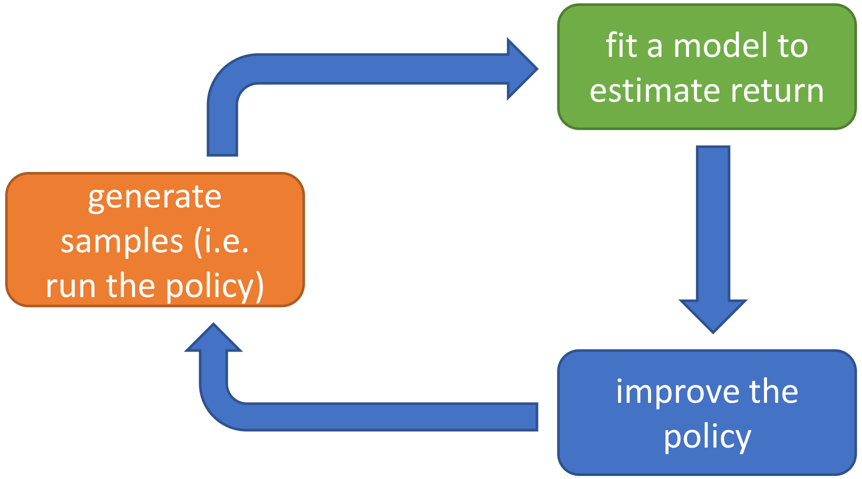
$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) \approx r(\mathbf{s}_t,\mathbf{a}_t) + \underline{\sum_{t'=t+1}^T E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]}$$

$$V^\pi(\mathbf{s}_{t+1})$$

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

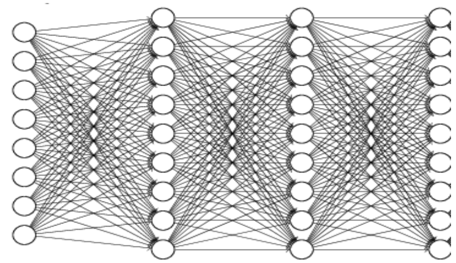
let's just fit $V^\pi(\mathbf{s})$!

fit Q^π, V^π, or A^π



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

$\mathbf{s}$



$\hat{V}^\pi(\mathbf{s})$

parameters ϕ

Policy evaluation

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

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

how can we perform policy evaluation?

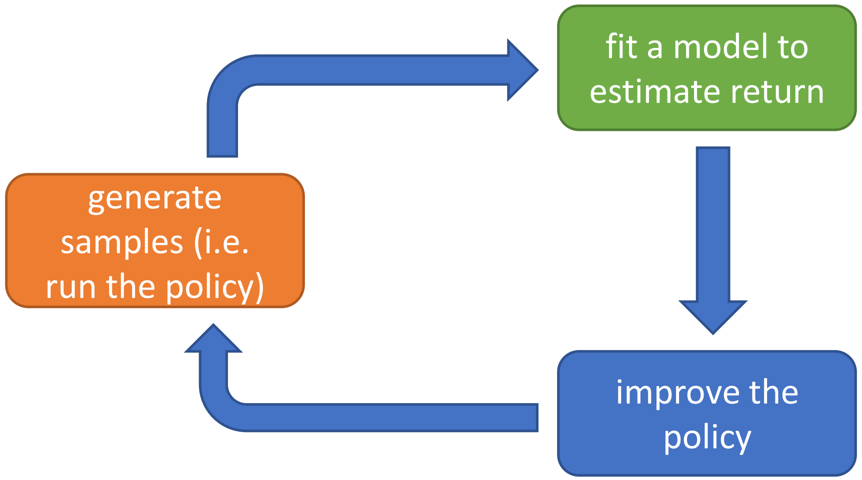
Monte Carlo policy evaluation (this is what policy gradient does)

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

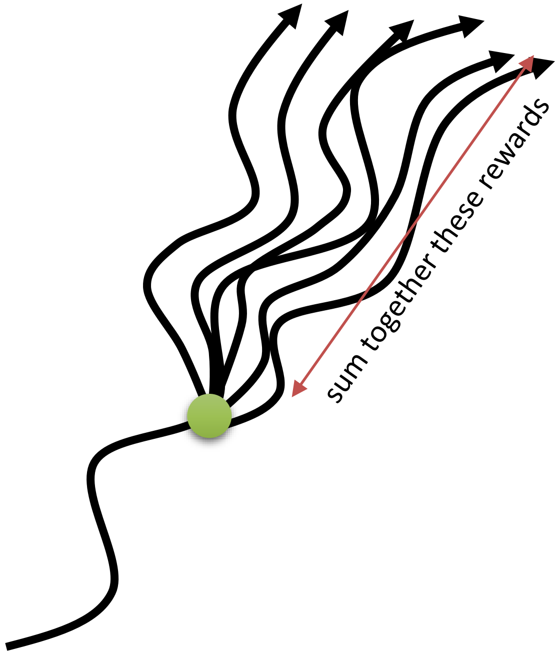
$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

(requires us to reset the simulator)

fit V^π



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

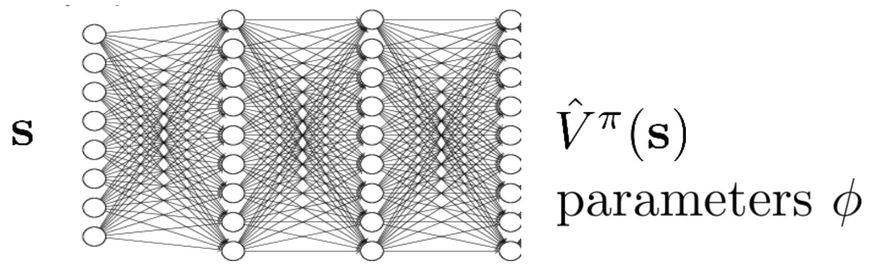


Monte Carlo evaluation with function approximation

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

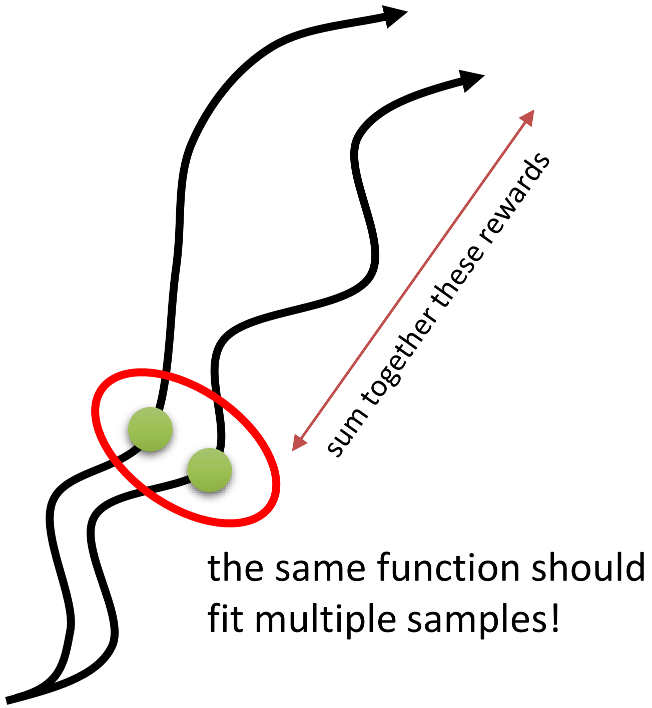
not as good as this: $V^\pi(\mathbf{s}_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$

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



Can we do better?

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

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

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

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

sometimes referred to as a “bootstrapped” estimate

An actor-critic algorithm

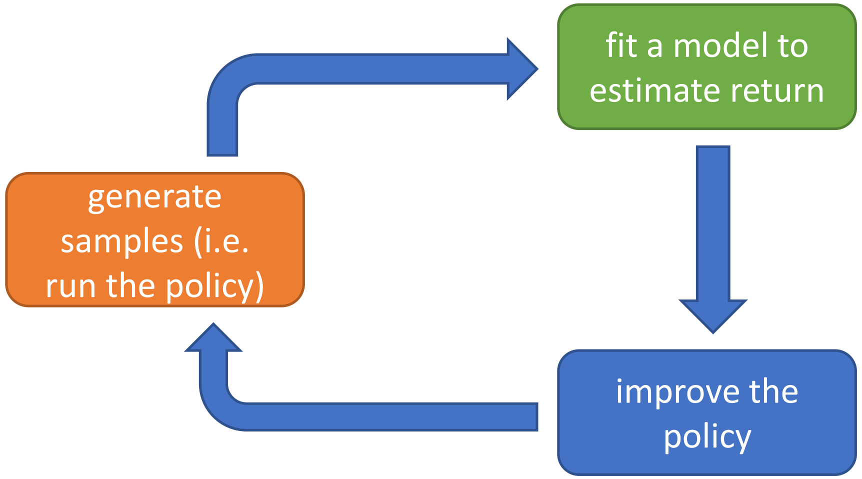
batch actor-critic algorithm:

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

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

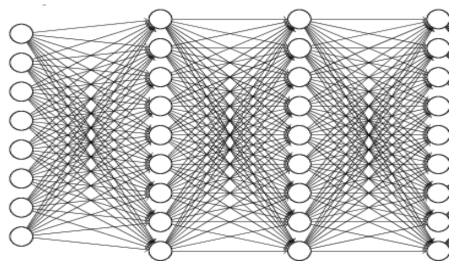
$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}^\pi_\phi(\mathbf{s}_i) - y_i \right\|^2$$

fit $\hat{V}^\pi$



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

$\mathbf{s}$



$\hat{V}^\pi(\mathbf{s})$

parameters ϕ

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

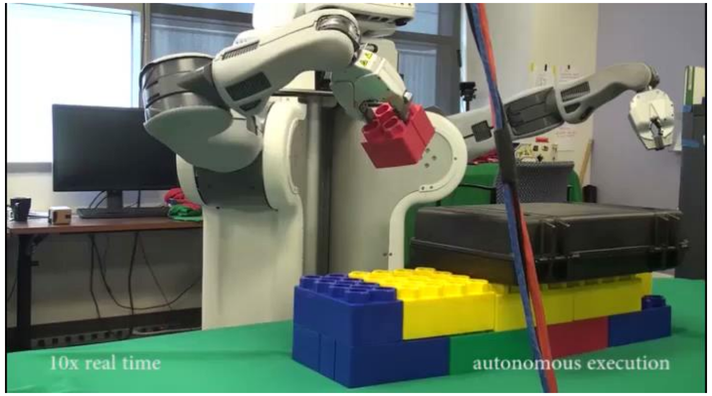
Aside: discount factors

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

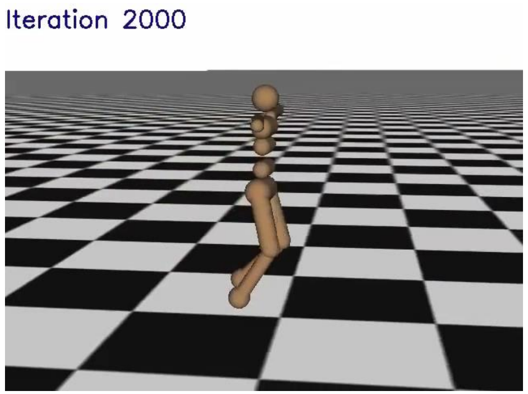
$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

what if T (episode length) is ∞?

$\hat{V}_\phi^\pi$ can get infinitely large in many cases



episodic tasks



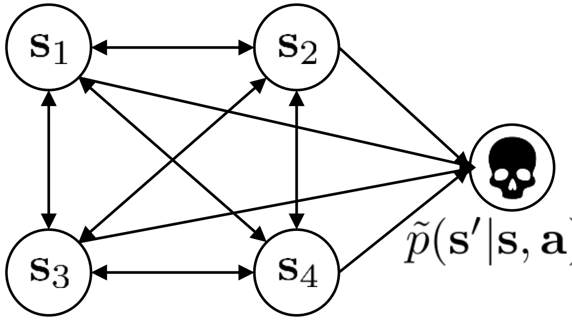
continuous/cyclical tasks

simple trick: better to get rewards sooner than later

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

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

γ changes the MDP:



$\tilde{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = (1-\gamma)$

$\tilde{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \gamma p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

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 $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
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:

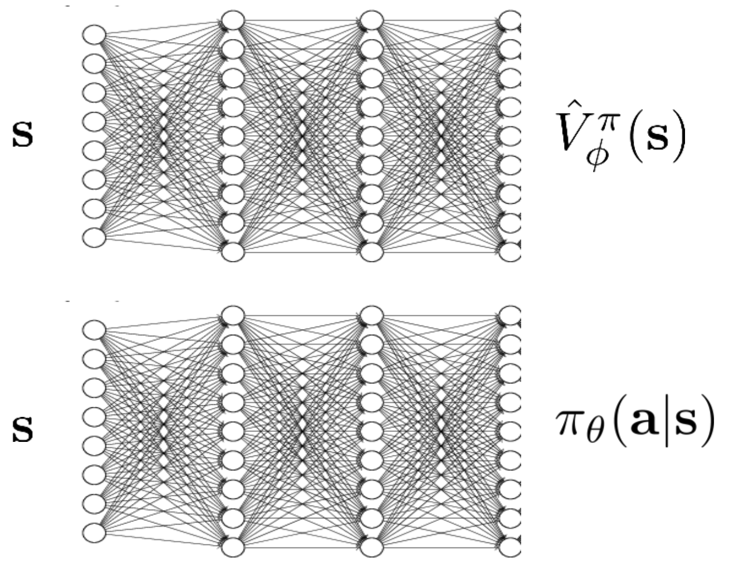
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}')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

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 $\hat{V}_\phi^\pi$ 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. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

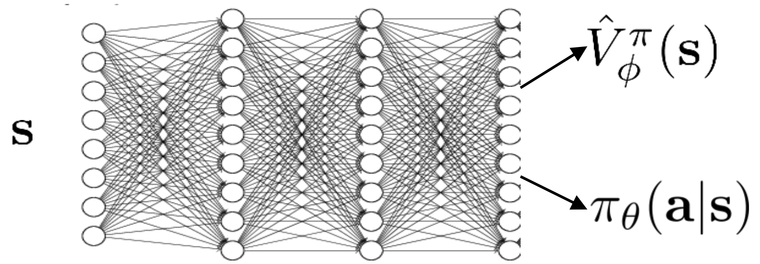
two network design



+ simple & stable

- no shared features between actor & critic

shared network design



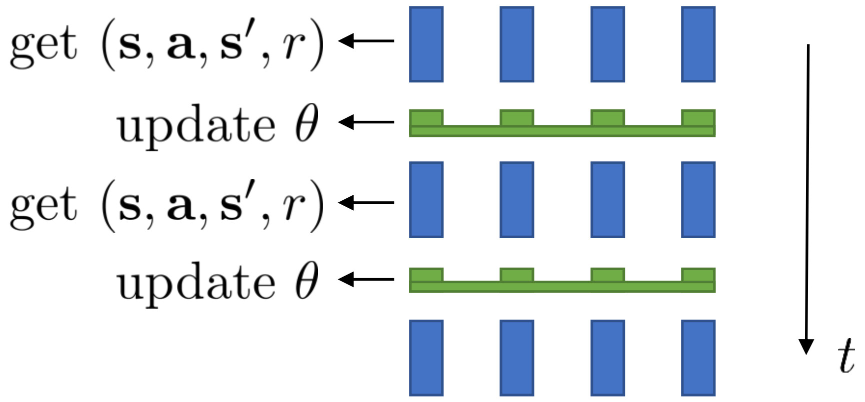
Online actor-critic in practice

online actor-critic algorithm:

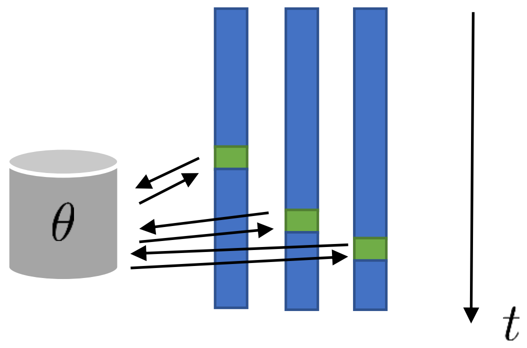
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}')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

works best with a batch (e.g., parallel workers)

synchronized parallel actor-critic

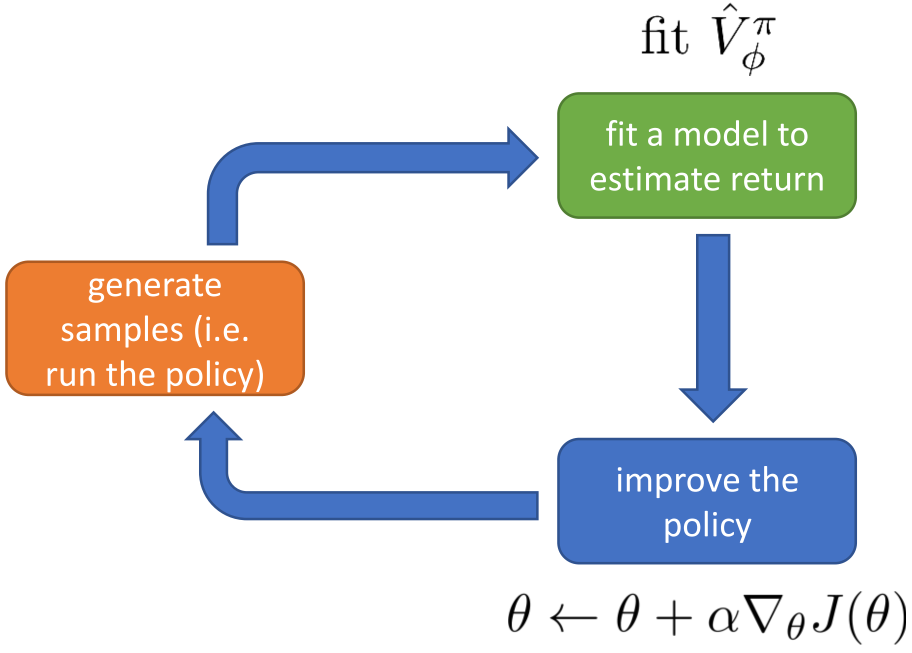


asynchronous parallel actor-critic



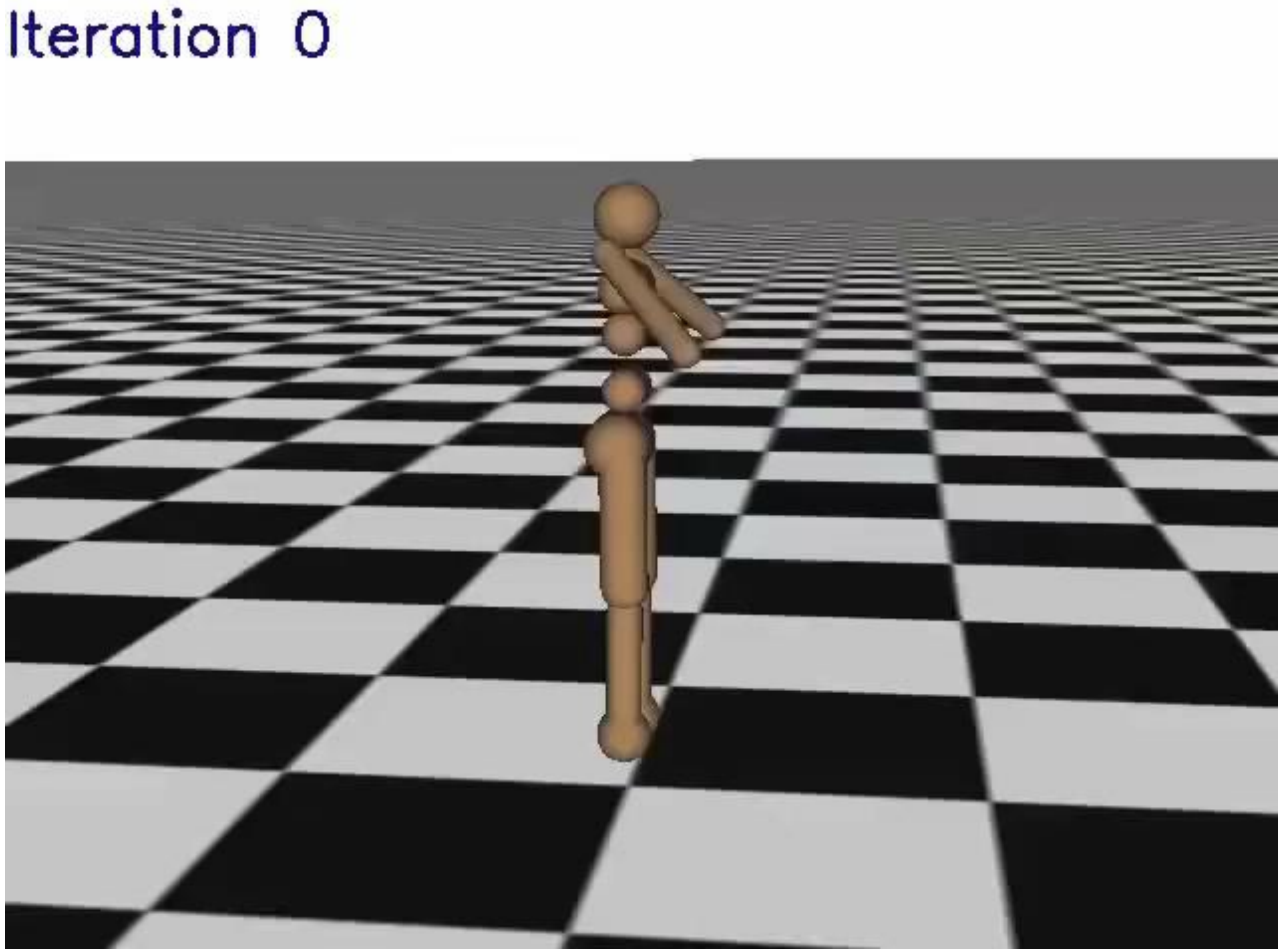
Review

- Actor-critic algorithms:
  - Actor: the policy
  - Critic: value function
  - Reduce variance of policy gradient
- Policy evaluation
  - Fitting value function to policy
- Discount factors
- Actor-critic algorithm design
  - One network (with two heads) or two networks
  - Batch-mode, or online (+ parallel)



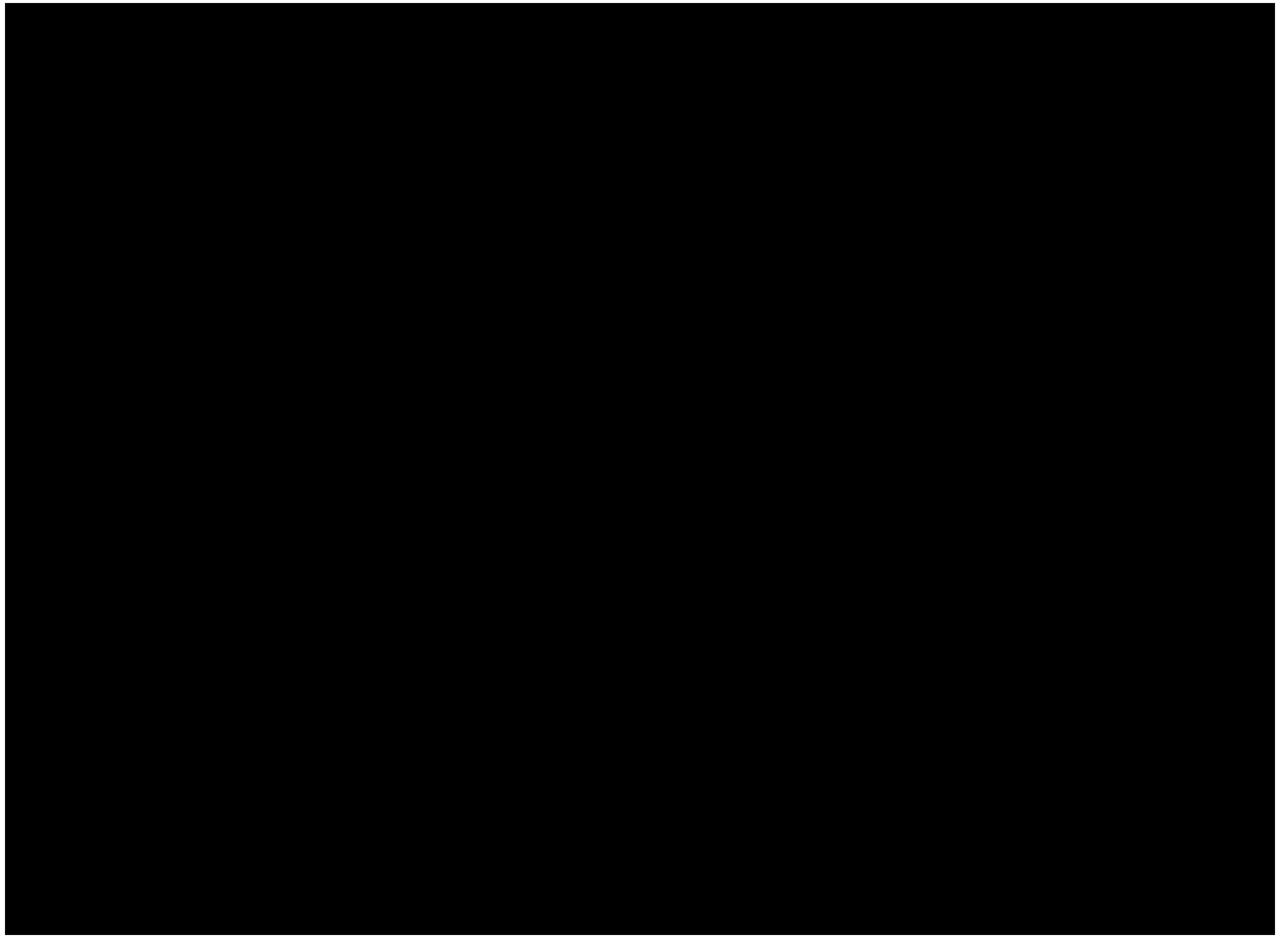
Actor-critic examples

- High dimensional continuous control with generalized advantage estimation (Schulman, Moritz, L., Jordan, Abbeel ‘16)
- Batch-mode actor-critic
- Hybrid blend of Monte Carlo return estimates and critic called generalized advantage estimation (GAE)



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

Today

- Actor-critic algorithm: reducing policy gradient variance using prediction
- Value-based algorithms: no more policy gradient, off-policy learning
- Model-based algorithms: control by predicting the future
- Open challenges and future directions

Improving the Gradient by… not using it anymore

Can we omit policy gradient completely?

$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: how much better is $\mathbf{a}_t$ than the average action according to π

$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: best action from $\mathbf{s}_t$, if we then follow π

at *least* as good as any $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$

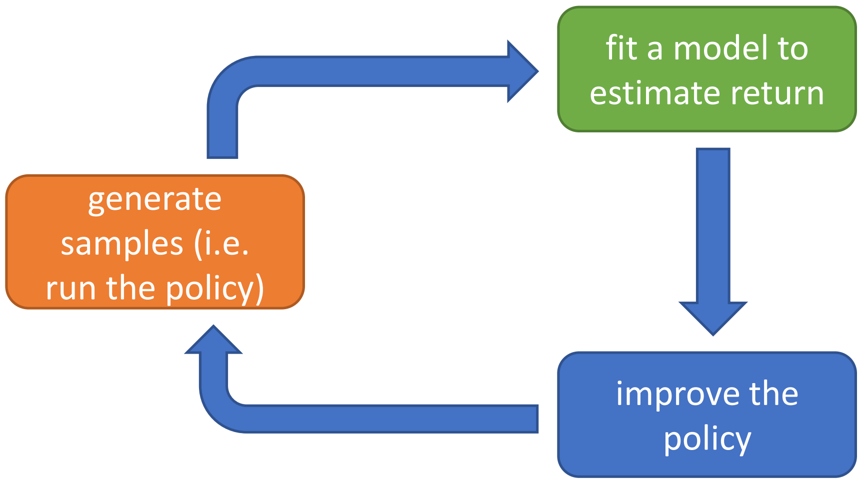
regardless of what $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is!

forget policies, let's just do this!

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

as good as π
(probably better)

fit A^π (or Q^π or V^π)



$\pi \leftarrow \pi'$

Policy iteration

High level idea:

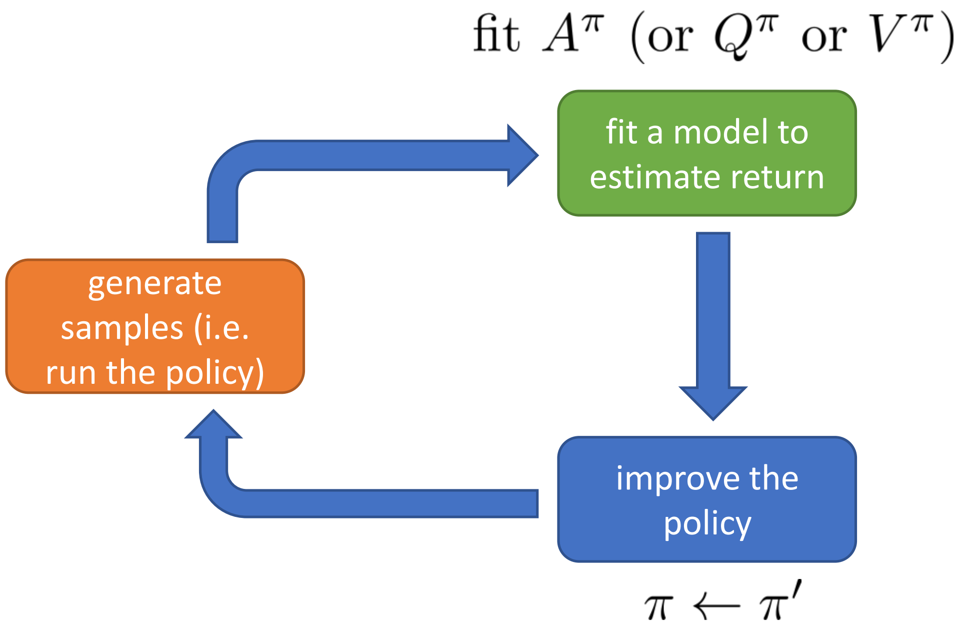
policy iteration algorithm:

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

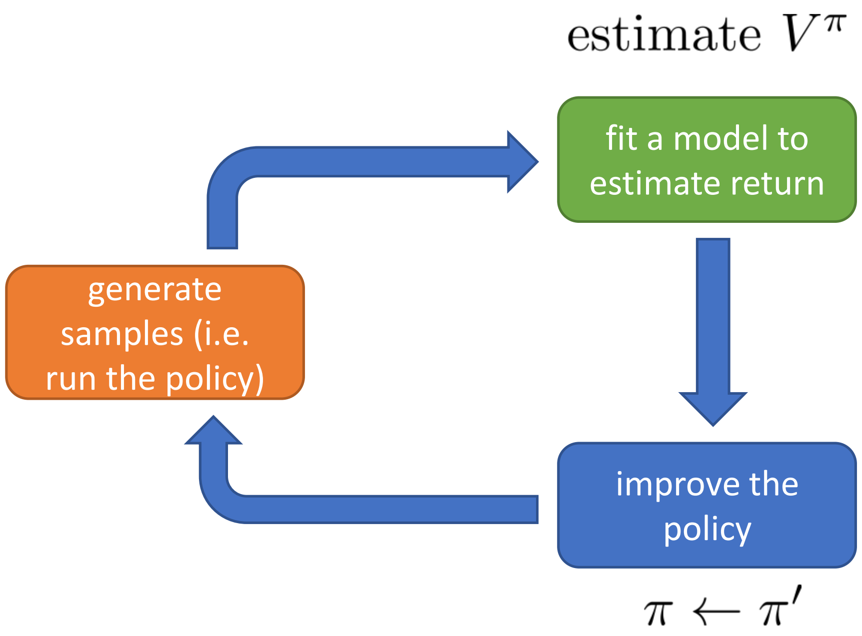
policy iteration:

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

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

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

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

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

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

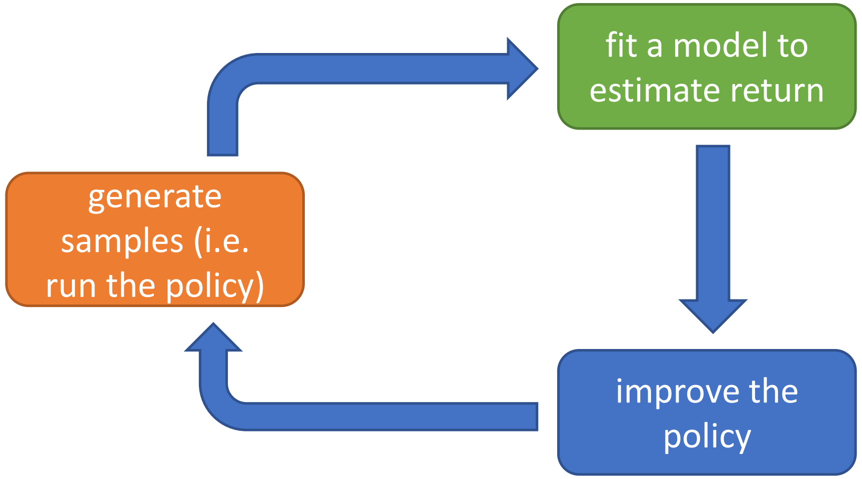
$\mathbf{a}$

$\mathbf{s}$				
	$Q(\mathbf{s},\mathbf{a})$		$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$
	$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$		$Q(\mathbf{s},\mathbf{a})$
		$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$
	$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$	
	$Q(\mathbf{s},\mathbf{a})$		$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$
	$Q(\mathbf{s},\mathbf{a})$	$Q(\mathbf{s},\mathbf{a})$		$Q(\mathbf{s},\mathbf{a})$

$\arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \rightarrow$ policy

approximates the new value!

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



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

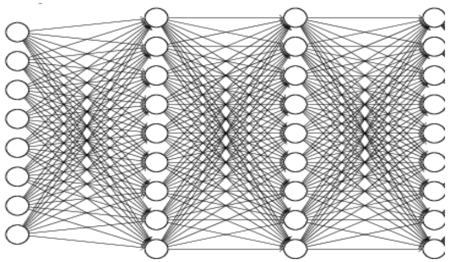
Fitted value iteration

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

big table, one entry for each discrete $\mathbf{s}$

neural net function $V : \mathcal{S} \rightarrow \mathbb{R}$

$\mathbf{s}$



$V(\mathbf{s})$

parameters ϕ

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

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 $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$

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

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

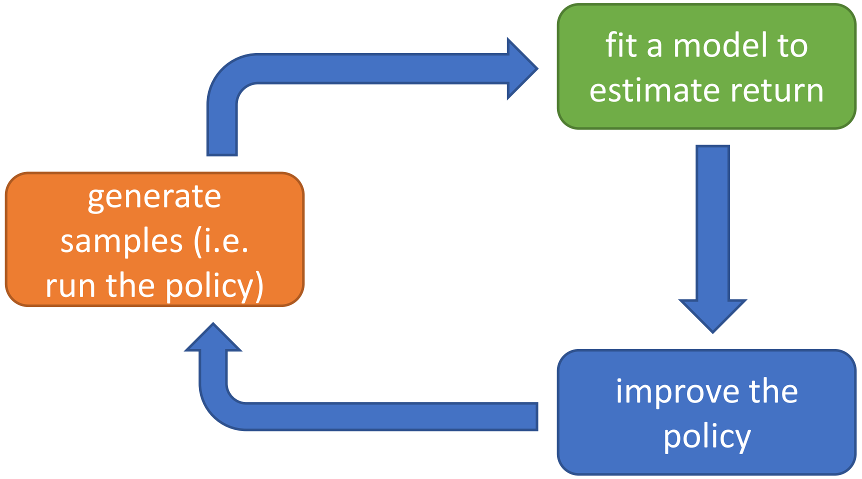
$\mathbf{s} = 2 : V(\mathbf{s}) = 0.5$

curse of dimensionality

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

(more than atoms in the universe)

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



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

What if we don't know the transition dynamics?

fitted value iteration algorithm:

1. set $\mathbf{y}_i \leftarrow \underline{\max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')])}$
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:

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

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

can fit this using samples

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

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

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

$K\times$

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 $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

parameters

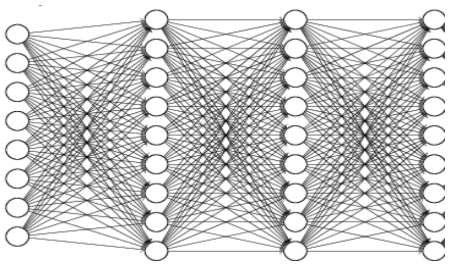
dataset size N, collection policy

iterations K

gradient steps S

$\mathbf{s}$

$\mathbf{a}$



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

parameters ϕ

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

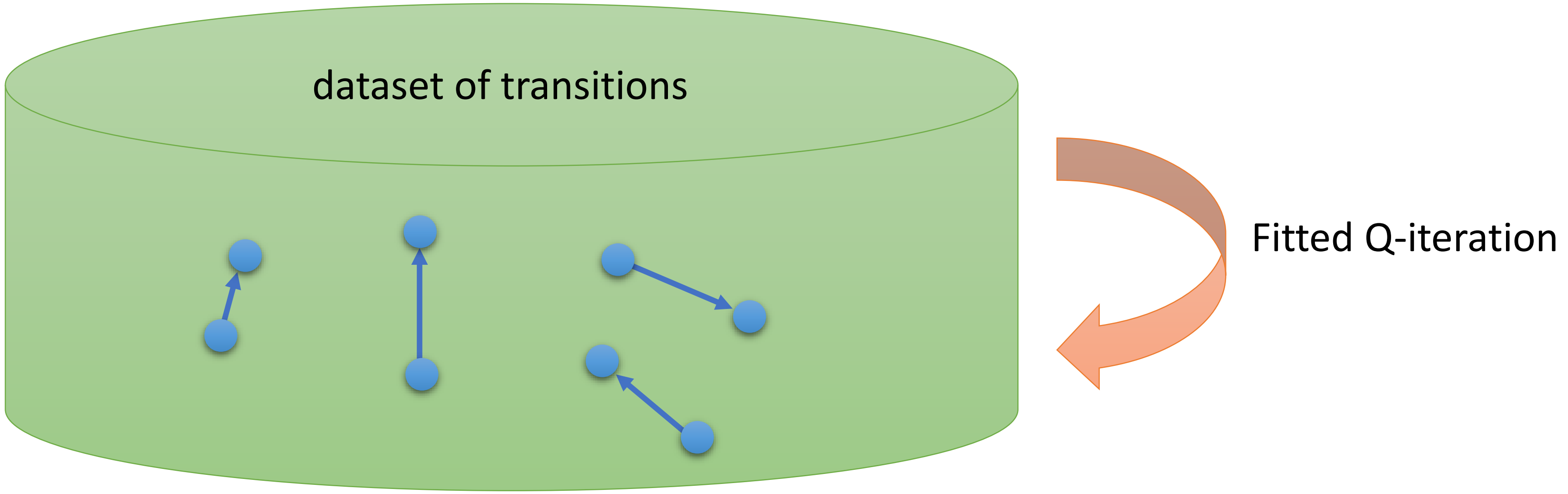
$K\times$

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 $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

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

$K\times$

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

error $\mathcal{E}$

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

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':

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

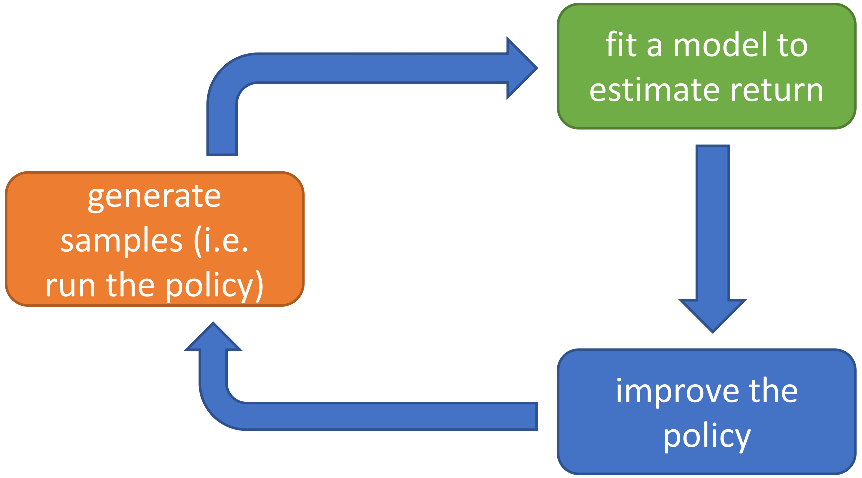
maximizes reward

sometimes written $Q^\star$ and $\pi^\star$

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

Online Q-learning algorithms

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



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

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 $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|Q_\phi(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{y}_i\|^2$

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"

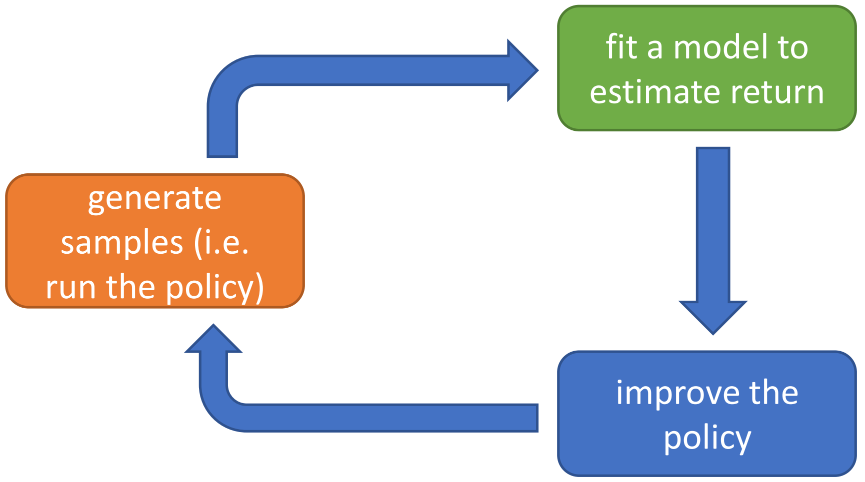
$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$$

"Boltzmann exploration"

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

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



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

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!

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

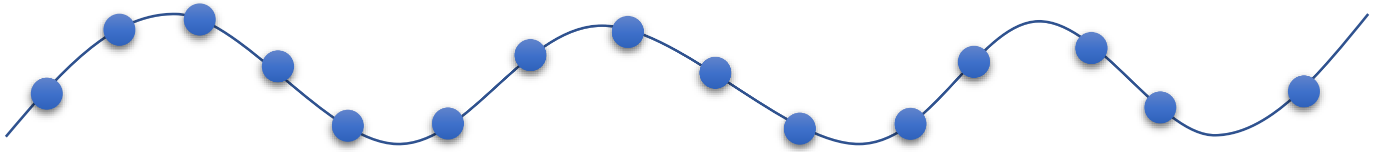
no gradient through target value

Correlated samples in online Q-learning

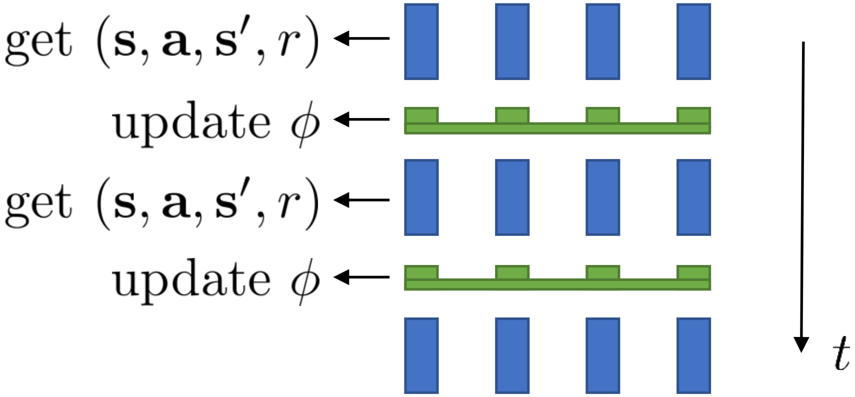
online Q iteration algorithm:

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

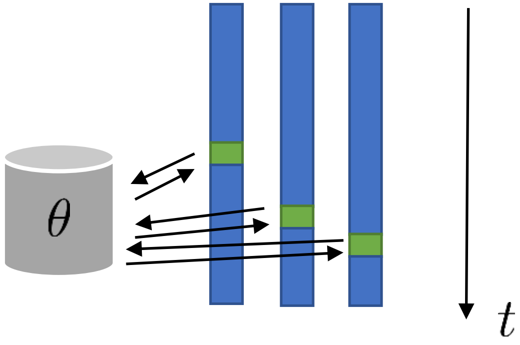
- sequential states are strongly correlated
- target value is always changing



synchronized parallel Q-learning



asynchronous parallel Q-learning



Another solution: replay buffers

online Q iteration algorithm:

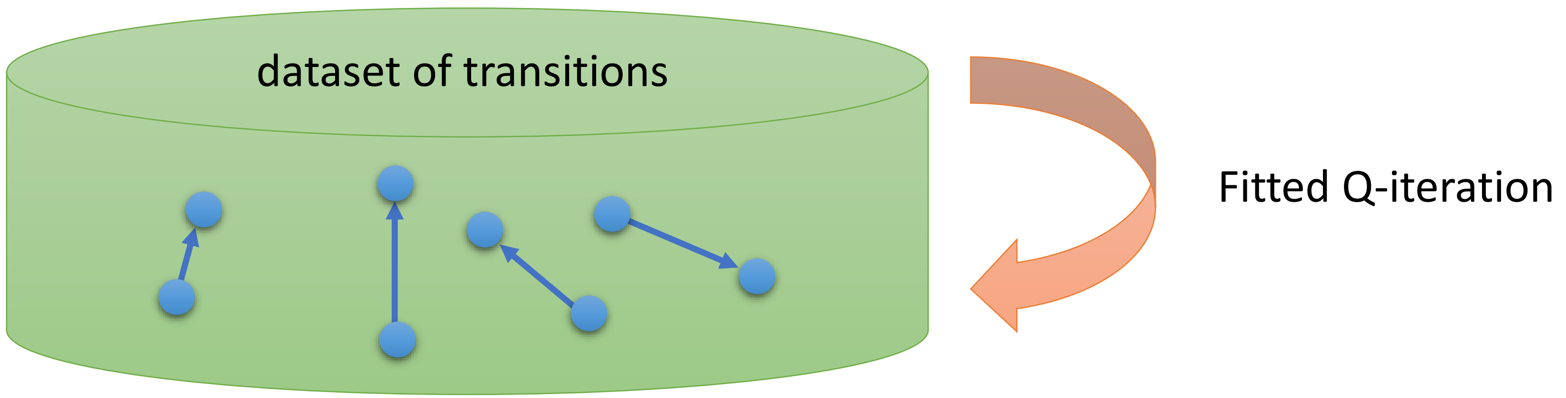
special case with K = 1, and one gradient step

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

full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy — **any policy will work! (with broad support)**
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ — **just load data from a buffer here**
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$ — **still use one gradient step**

$K\times$



Another solution: replay buffers

Q-learning with a replay buffer:

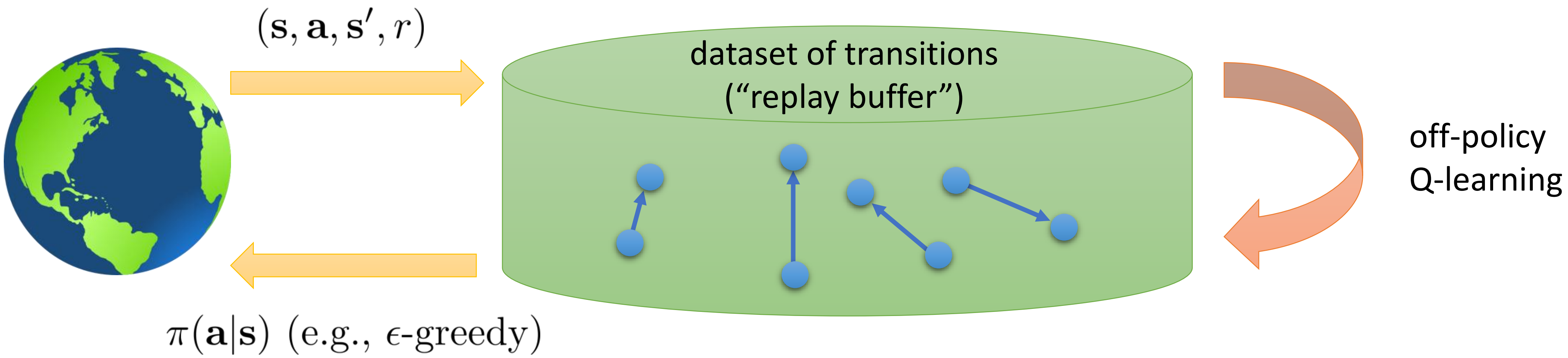
1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
2. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

+ samples are no longer correlated

+ multiple samples in the batch (low-variance gradient)

but where does the data come from?

need to periodically feed the replay buffer…



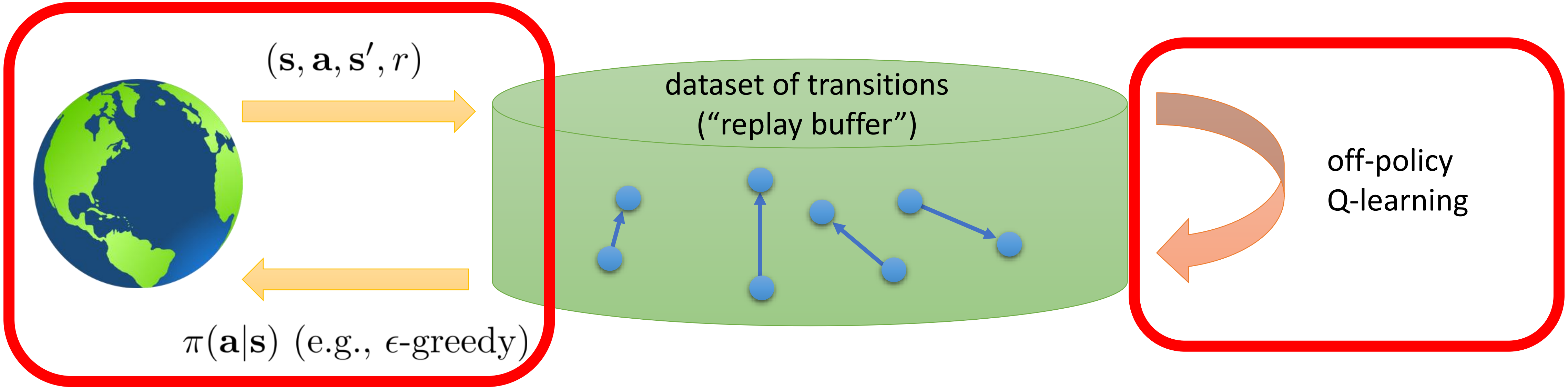
Putting it together

full Q-learning with replay buffer:

$K\times$

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



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!~~

use replay buffer

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 $\mathcal{B}$

$K \times$

2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
3. $\phi \leftarrow \phi - \alpha \sum_i \underline{\frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])}$

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

$K \times$

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 $\underline{\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2}$

perfectly well-defined, stable regression

Q-Learning with target networks

Q-learning with replay buffer and target network:

$N\times$ $K\times$

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

targets don't change in inner loop!

supervised regression

"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 $\mathcal{B}$

$N\times$

3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$

$K\times$

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

$K = 1$

Mnih et al. '13

Fitted Q-iteration and Q-learning

Q-learning with replay buffer and target network:

DQN: $N = 1,\ K = 1$

$N\times$

1. save target network parameters: $\phi' \leftarrow \phi$

   $K\times$

   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):

$N\times$

1. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to $\mathcal{B}$

   $K\times$

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

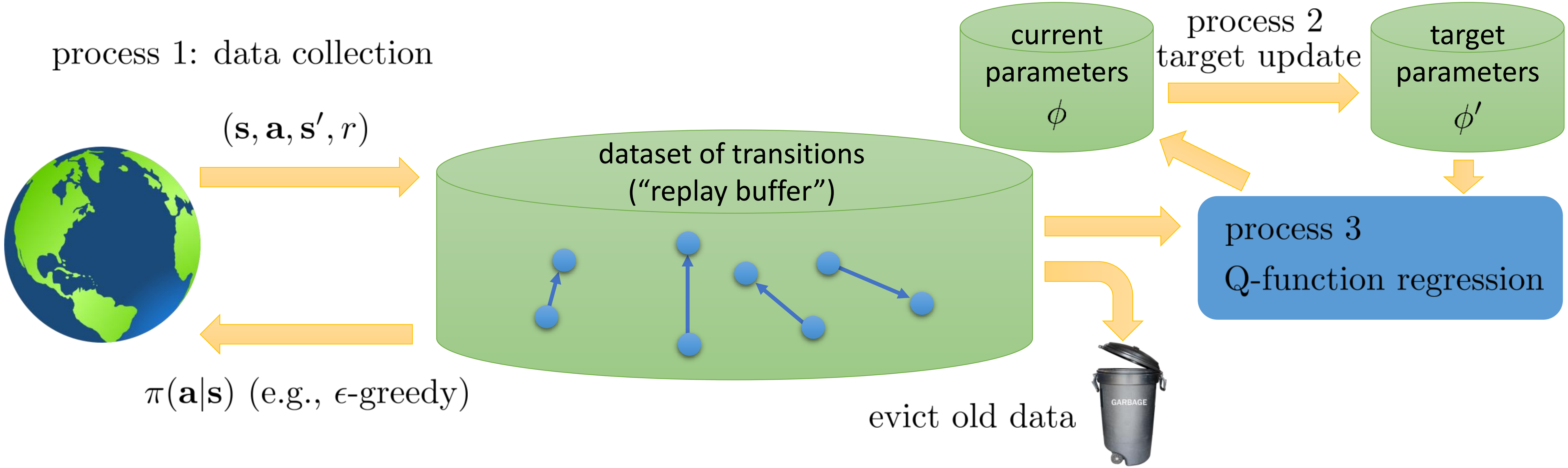
just SGD

A more general view

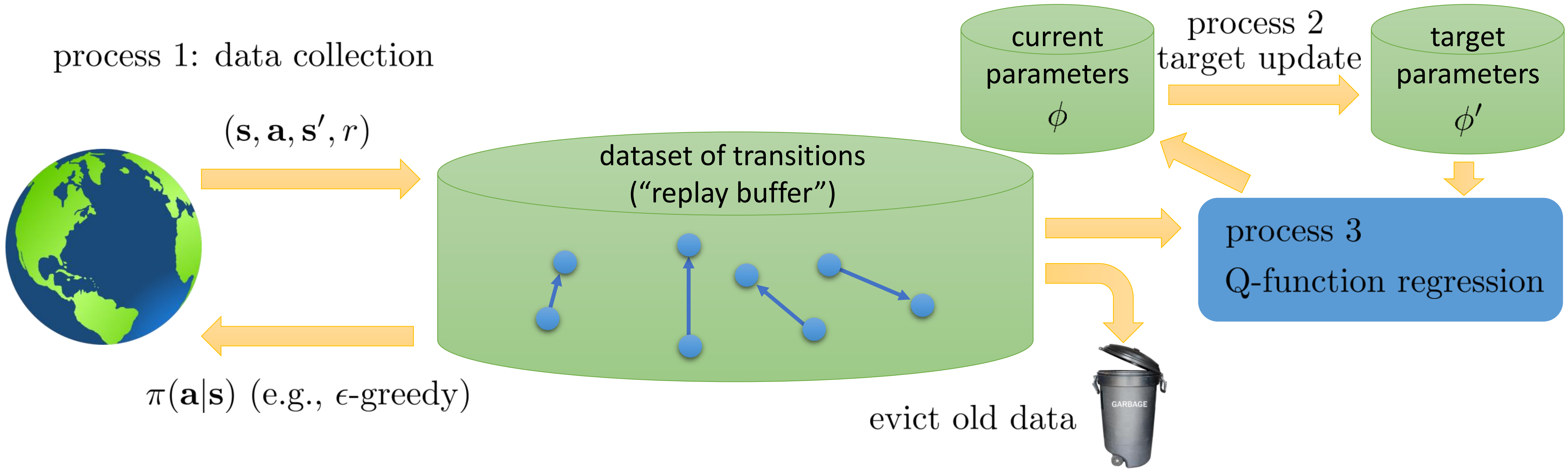
Q-learning with replay buffer and target network:

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

$N\times$ $K\times$



A more general view



- Online Q-learning: 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

Q-learning with continuous actions

What's the problem with continuous actions?

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

this max

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

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:

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

$(\mathbf{a}_1, \ldots, \mathbf{a}_N)$ 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:

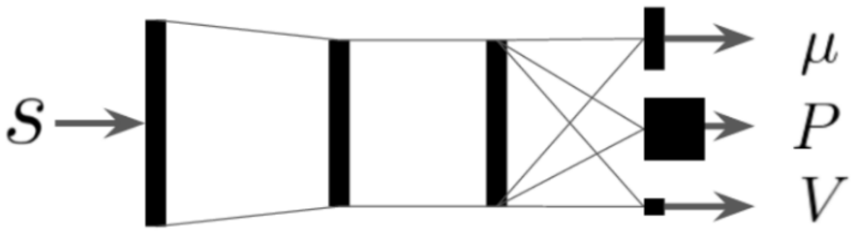
- cross-entropy method (CEM)
  - simple iterative stochastic optimization
- CMA-ES
  - substantially less simple iterative stochastic optimization

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 Architecture.

NAF: **N**ormalized **A**dvantage **F**unctions

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

+ no change to algorithm

+ just as efficient as Q-learning

- loses representational power

Gu, Lillicrap, Sutskever, L., ICML 2016

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

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

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

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 $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ 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. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mathbf{a})$
6. update ϕ' and θ' (e.g., Polyak averaging)

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

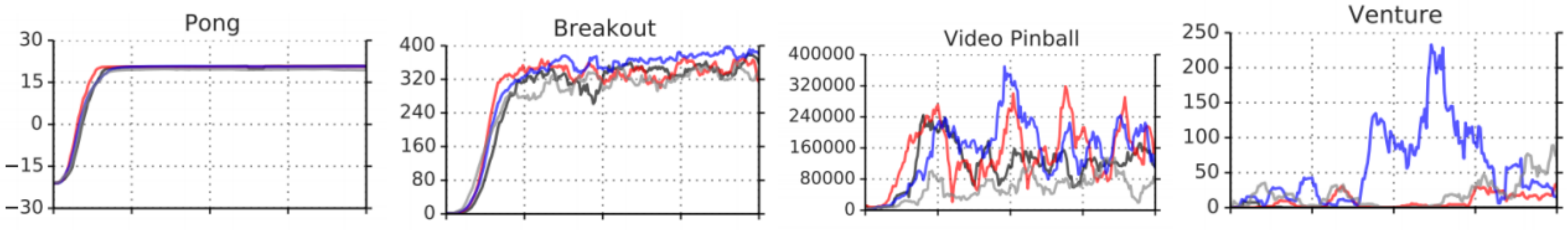


Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

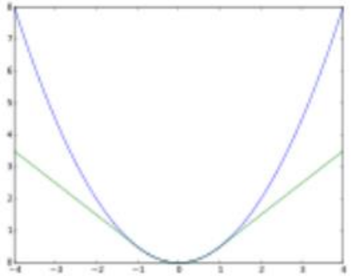
- Large replay buffers help improve stability
  - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Slide partly borrowed from J. Schulman

Advanced tips for Q-learning

- Bellman error gradients can be big; clip gradients or user Huber loss

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$

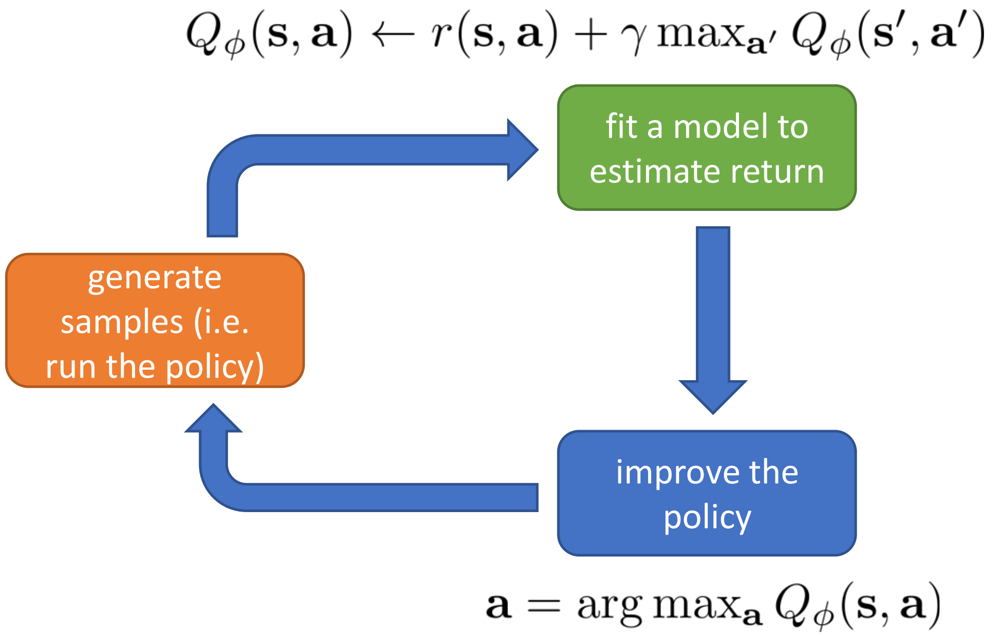


- Double Q-learning helps *a lot* in practice (see readings at the end), simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it's very inconsistent between runs

Slide partly borrowed from J. Schulman

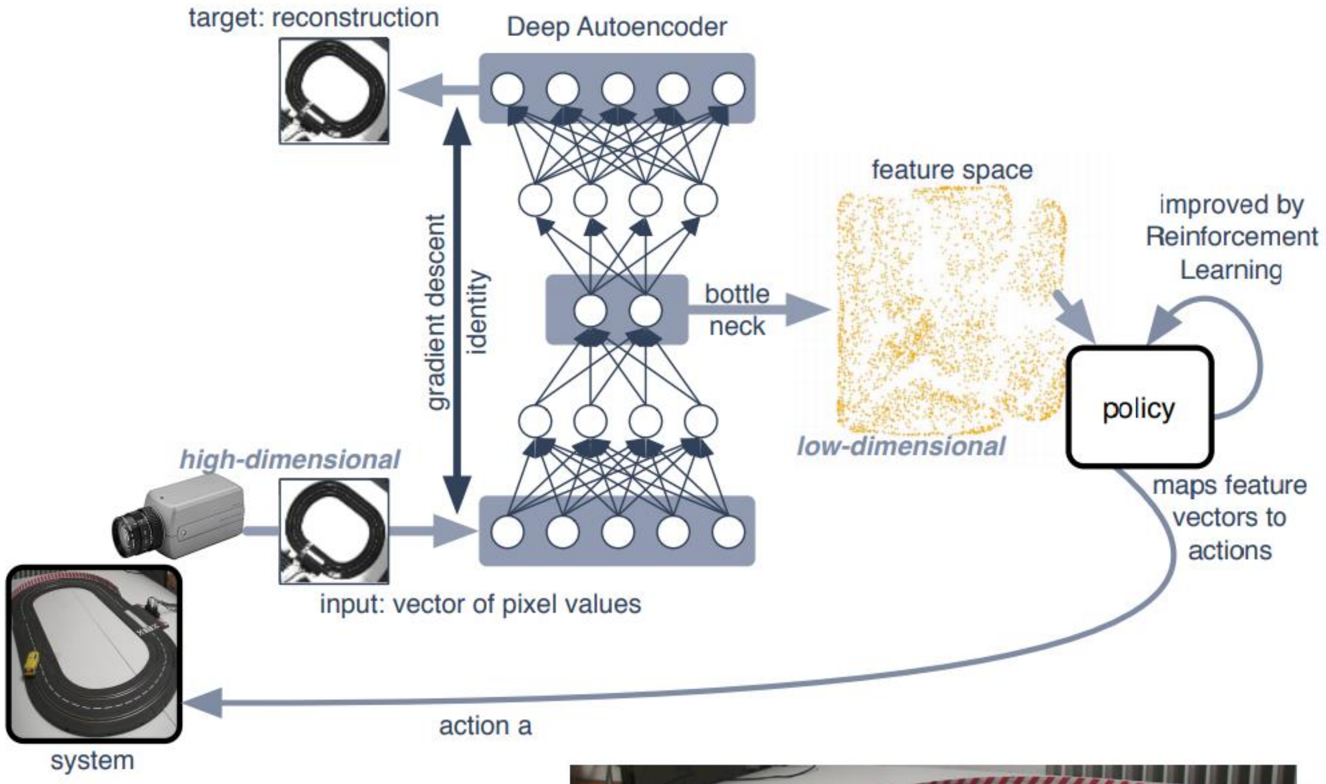
Review

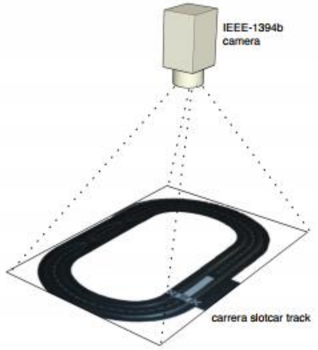
- Q-learning in practice
  - Replay buffers
  - Target networks
- Generalized fitted Q-iteration
- Q-learning with continuous actions
  - Random sampling
  - Analytic optimization
  - Second “actor” network



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)



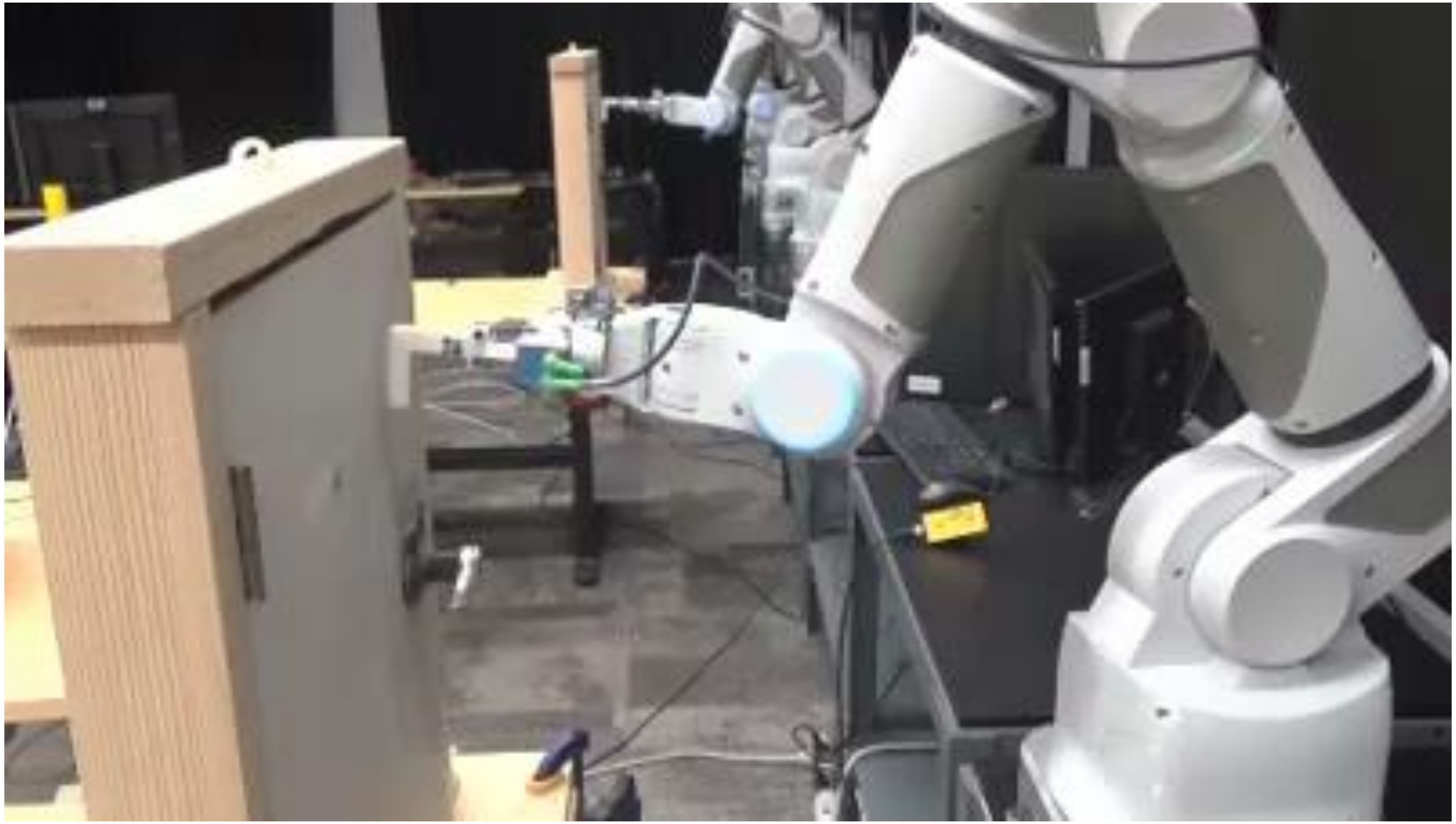


Q-learning with convolutional networks

- "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)

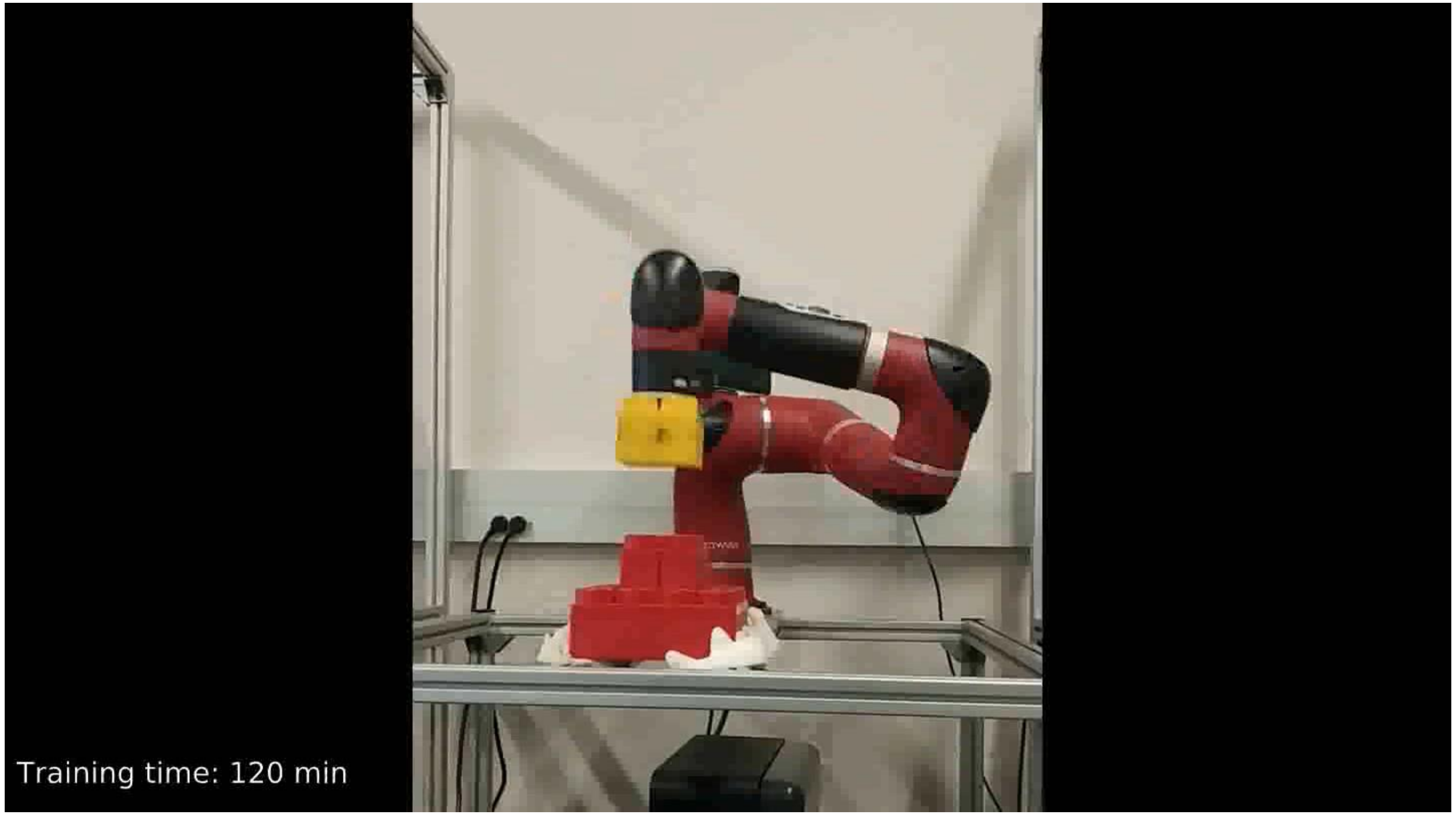
Q-learning on a real robot

- "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



Q-learning on a real robot

- "Composable Deep Reinforcement Learning for Robotic Manipulation," Haarnoja, et al. '18
  - See also "Soft Actor Critic" and "Reinforcement Learning with Deep Energy-Based Policies" (Haarnoja et al. '18 & '17)
- Continuous actions with maximum entropy Q-learning (we'll cover this if time permits)
- Uses replay buffer and target network
- Multiple gradient steps per sim step
- Entropy maximization provides for robust policies



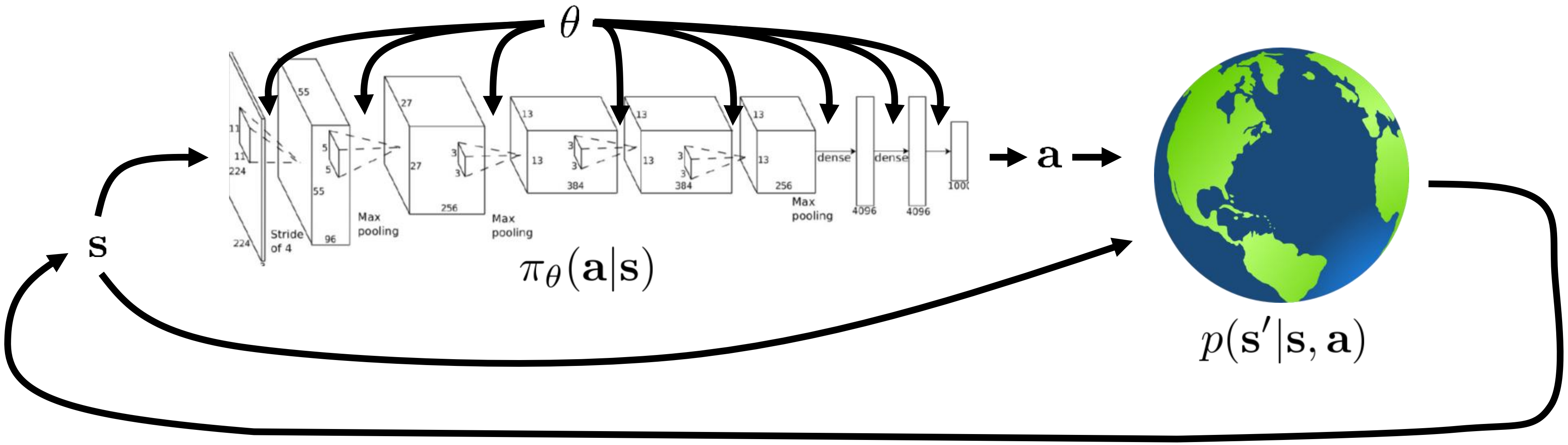
Q-learning suggested readings

- 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.
- Practice on your own!
  - Homework 3 here: https://github.com/berkeleydeeprlcourse/homework

Today

- Actor-critic algorithm: reducing policy gradient variance using prediction
- Value-based algorithms: no more policy gradient, off-policy learning
- Model-based algorithms: control by predicting the future
- Open challenges and future directions

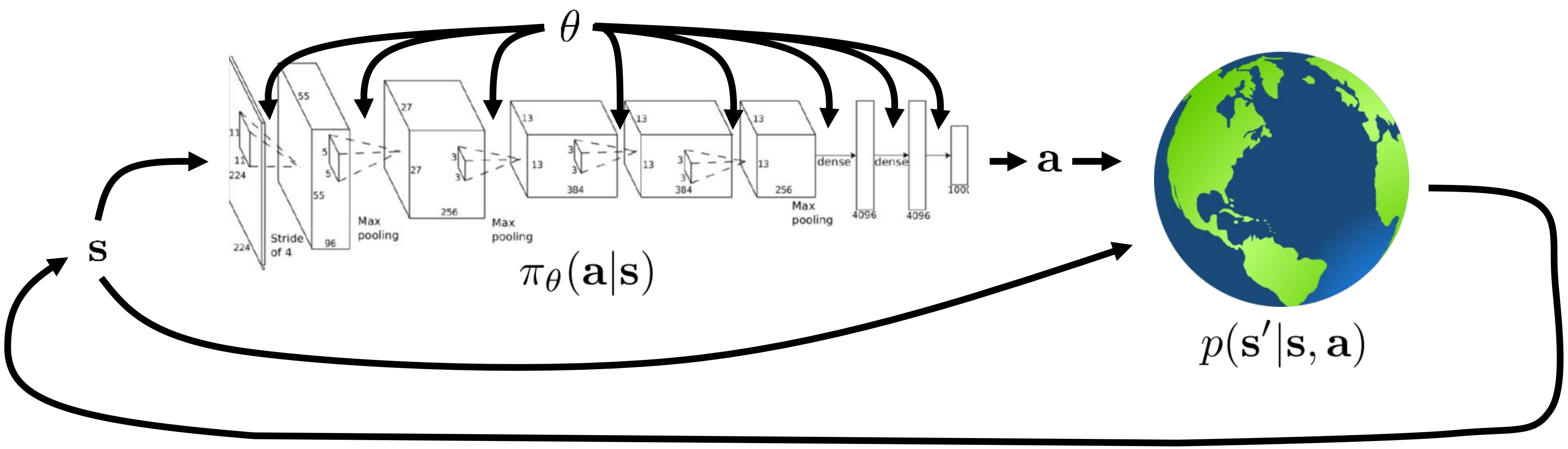
Recap: the reinforcement learning objective



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

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

Recap: model-free reinforcement learning



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

assume this is unknown
don't even attempt to learn it

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

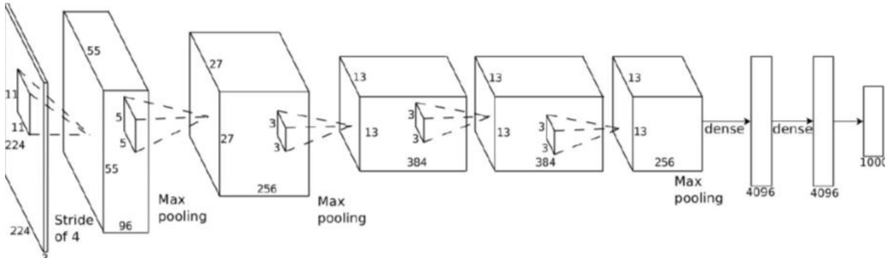
What if we knew the transition dynamics?

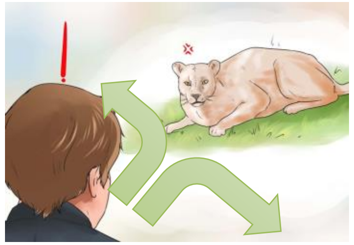
- Often we do know the dynamics
  1. Games (e.g., Go)
  2. Easily modeled systems (e.g., navigating a car)
  3. Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
  1. System identification – fit unknown parameters of a known model
  2. Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier?

Often, yes!

The objective





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

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

Stochastic optimization

abstract away optimal control/planning:

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \ldots, \mathbf{a}_T)}_{\text{don't care what this is}} \qquad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

simplest method: guess & check "random shooting method"

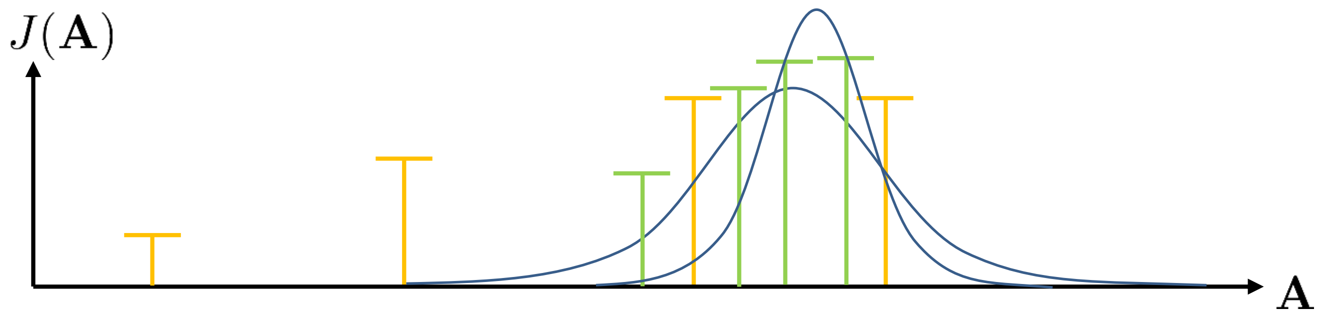
1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

Cross-entropy method (CEM)

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)

can we do better?

2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$



typically use Gaussian distribution

cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from $p(\mathbf{A})$
2. evaluate $J(\mathbf{A}_1), \ldots, J(\mathbf{A}_N)$
3. pick the *elites* $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$

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

What else can we do?

1. Discrete actions: Monte Carlo tree search
2. Continuous actions: trajectory optimization, LQR/iterative LQR

What if we don't know the model?

If we knew $f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$, no more learning, just planning!

(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:

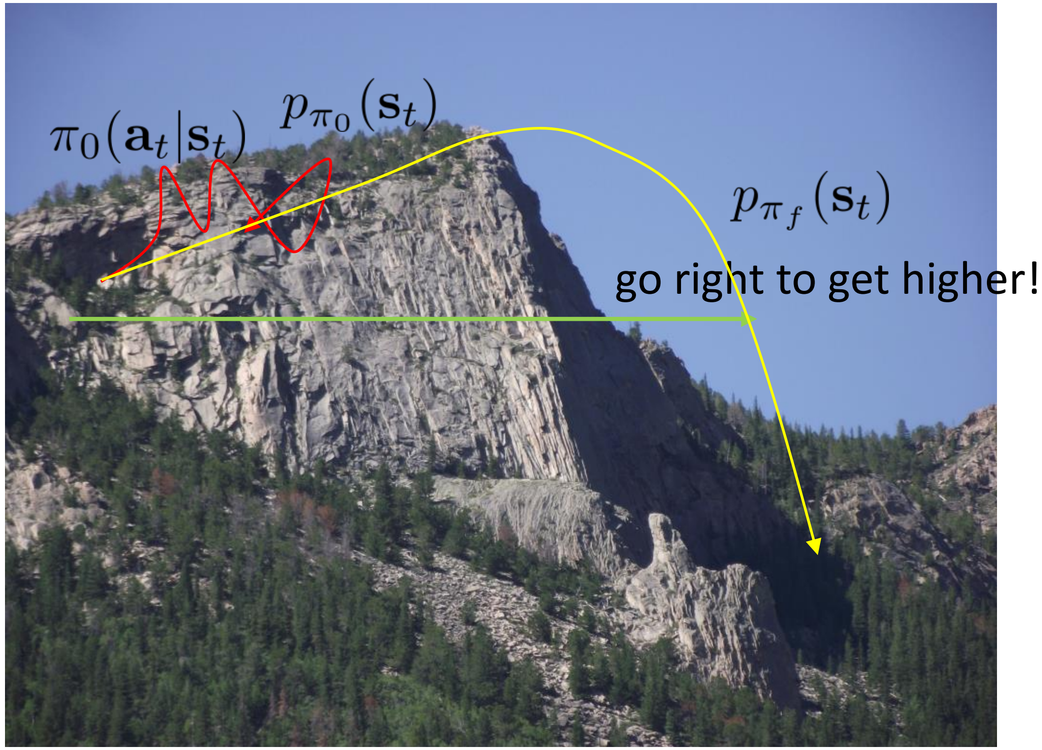
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ 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?

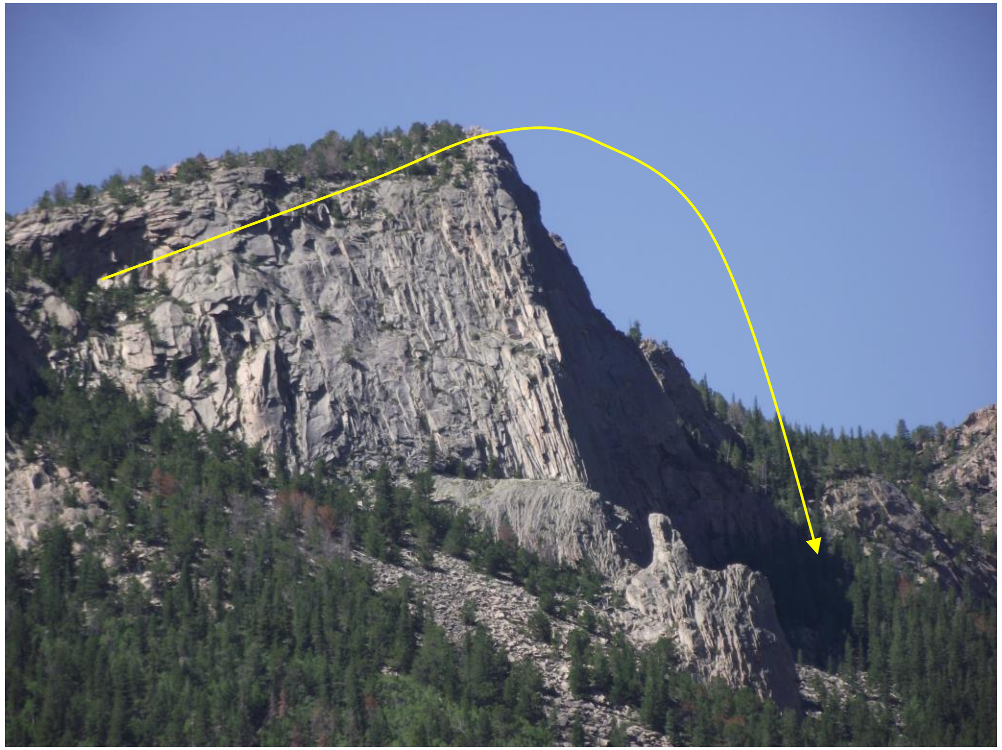
can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

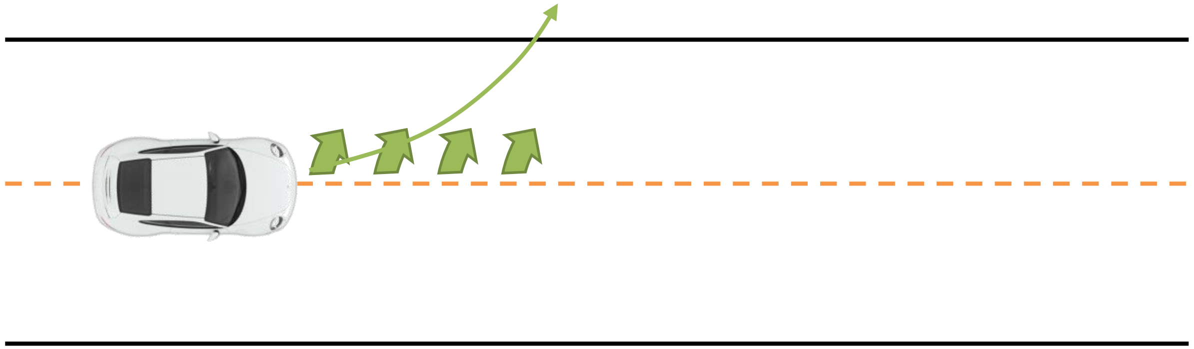
need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

model-based reinforcement learning version 1.0:

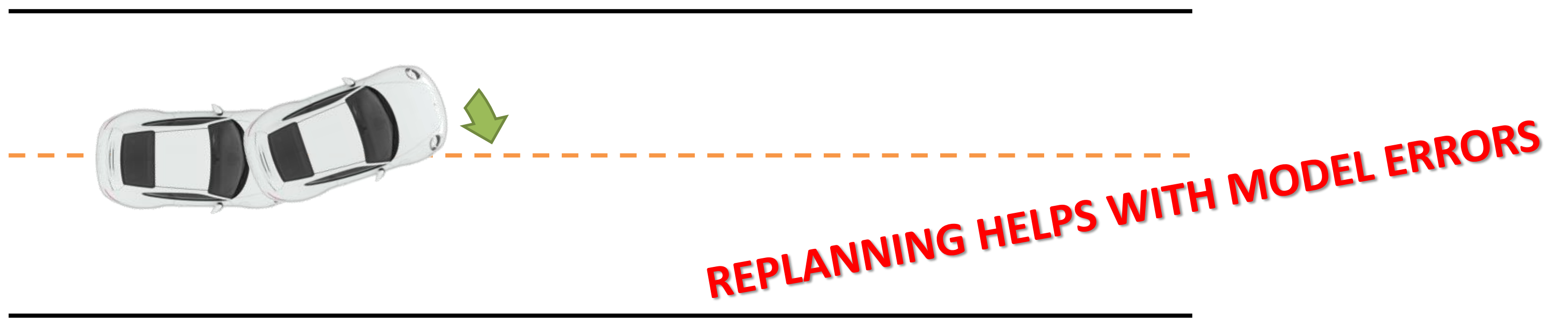
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ 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

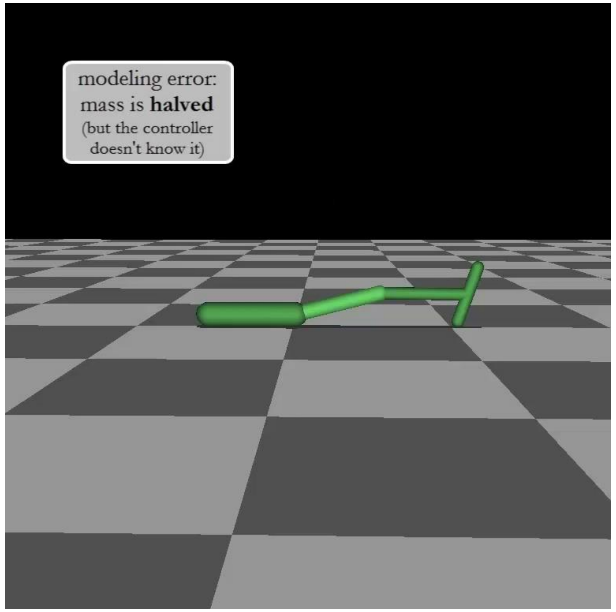
How to replan?

model-based reinforcement learning version 1.5:

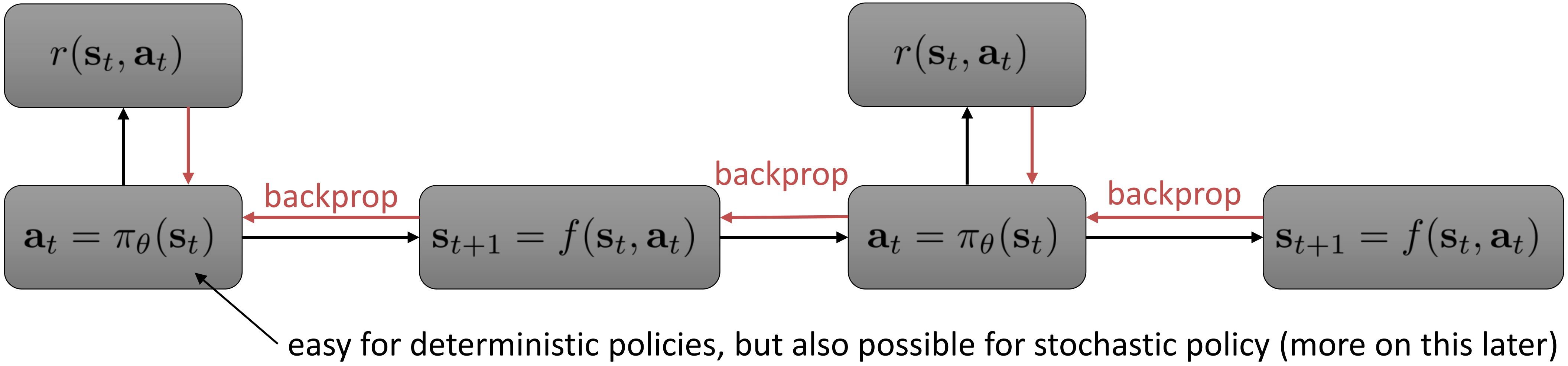
every N steps

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 more you replan, the less perfect each individual plan needs to be
- Can use shorter horizons
- Even random sampling can often work well here!



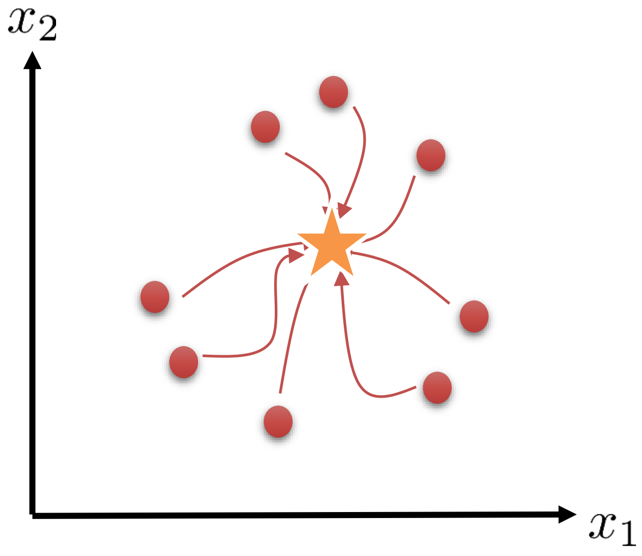
Backpropagate directly into the policy?



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 $\mathcal{D}$

Learning policies without BPTT: Guided policy search



optimize trajectory $\mathbf{s}_0, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T$ from $\mathbf{s}_0 = \mathbf{s}_0^0$

optimize trajectory $\mathbf{s}_0, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T$ from $\mathbf{s}_0 = \mathbf{s}_0^1$

optimize trajectory $\mathbf{s}_0, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T$ from $\mathbf{s}_0 = \mathbf{s}_0^2$

model-based RL provides **data**

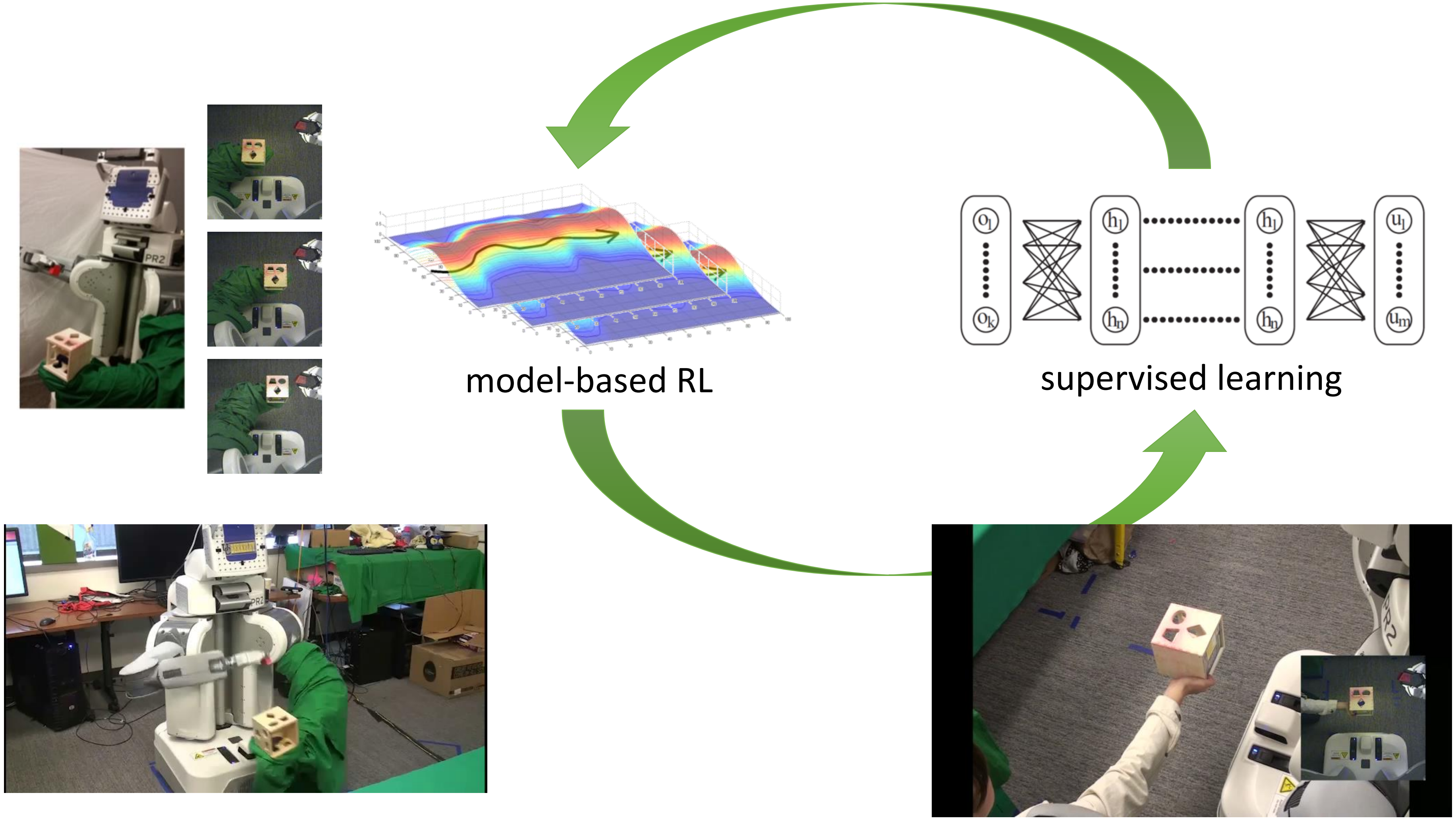
train $\pi_\theta(\mathbf{a}|\mathbf{s})$ using supervised learning

supervised learning provides **reward term**

$$\tilde{r}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \eta \log \pi_\theta(\mathbf{a}|\mathbf{s})$$

Levine*, Finn*, et al. End-to-end training of deep visuomotor policies. 2016.

Guided Policy Search



Summary

- Version 0.5: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem
- Version 1.0: iteratively collect data, replan, collect data
  - Pro: simple, solves distribution mismatch
  - Con: open loop plan might perform poorly, esp. in stochastic domains
- Version 1.5: iteratively collect data using MPC (replan at each step)
  - Pro: robust to small model errors
  - Con: computationally expensive, but have a planning algorithm available
- Version 2.0: backpropagate directly into policy
  - Pro: computationally cheap at runtime
  - Con: direct backprop doesn't usually work, but decomposition does (look up "guided policy search")

Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning

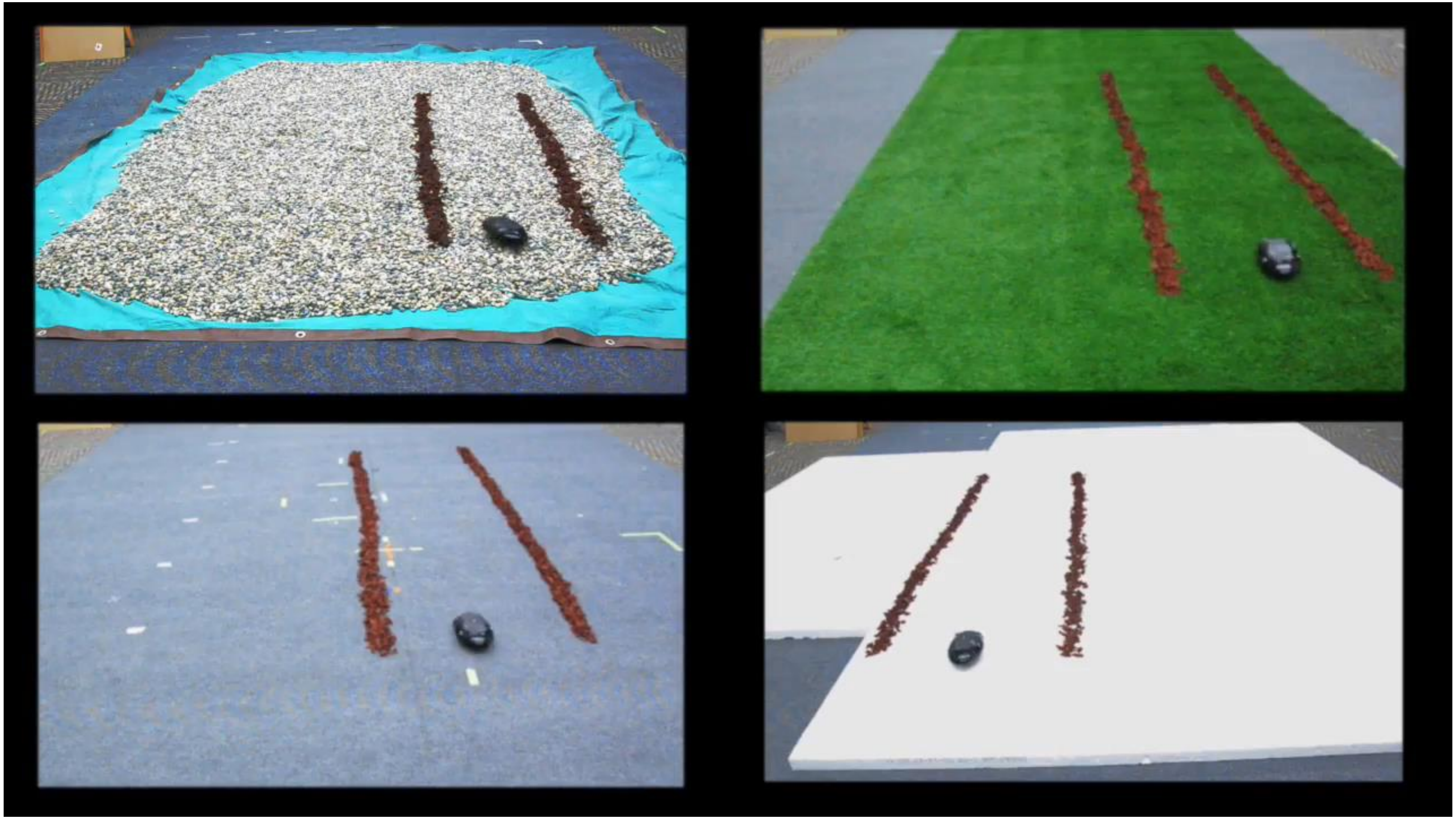
Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, Sergey Levine
University of California, Berkeley

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 (random sampling)
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

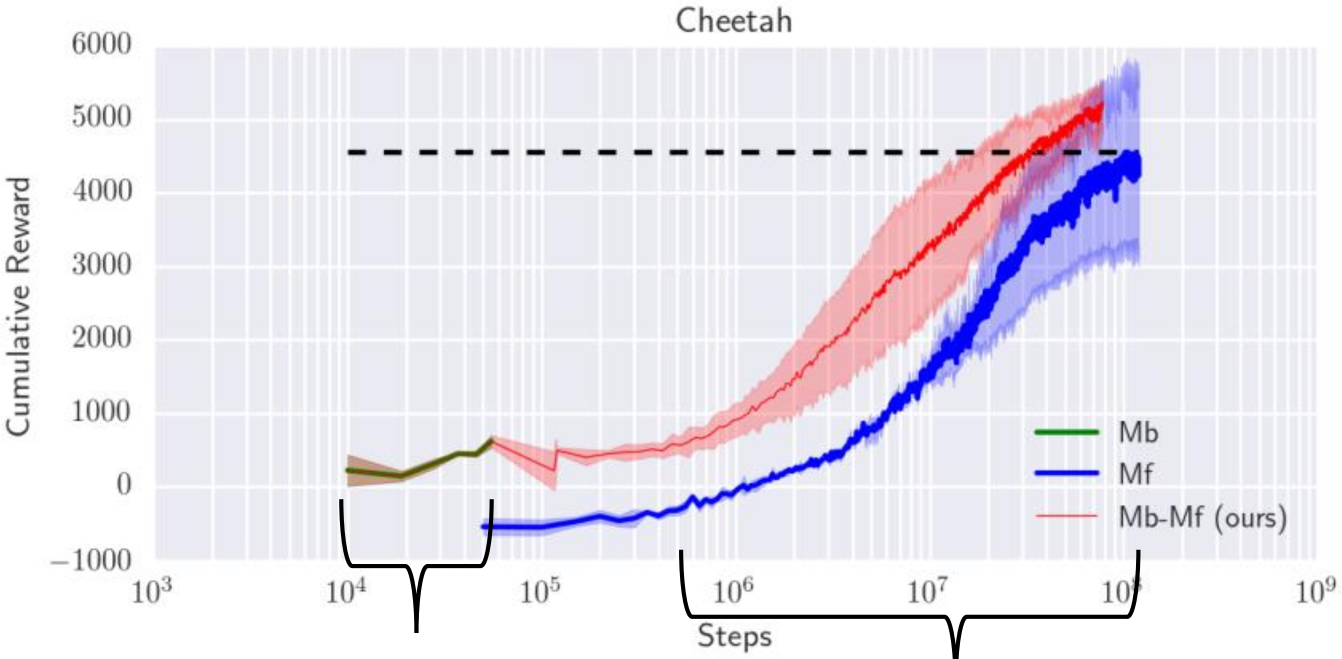
every N steps

Running on terrain with random shooting



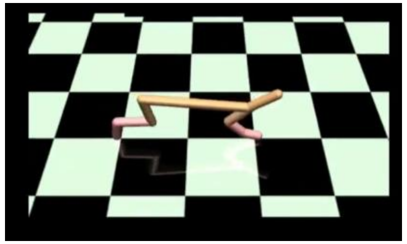
Nagabandi, Yang, Asmar, Pandya, Kahn, L., Fearing. arxiv 2018

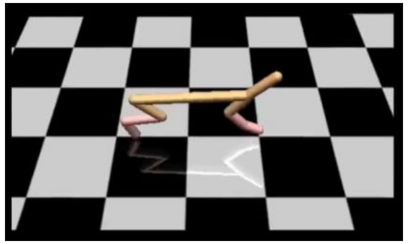
Bridging the Gap in Performance



pure model-based
(about 10 minutes real time)

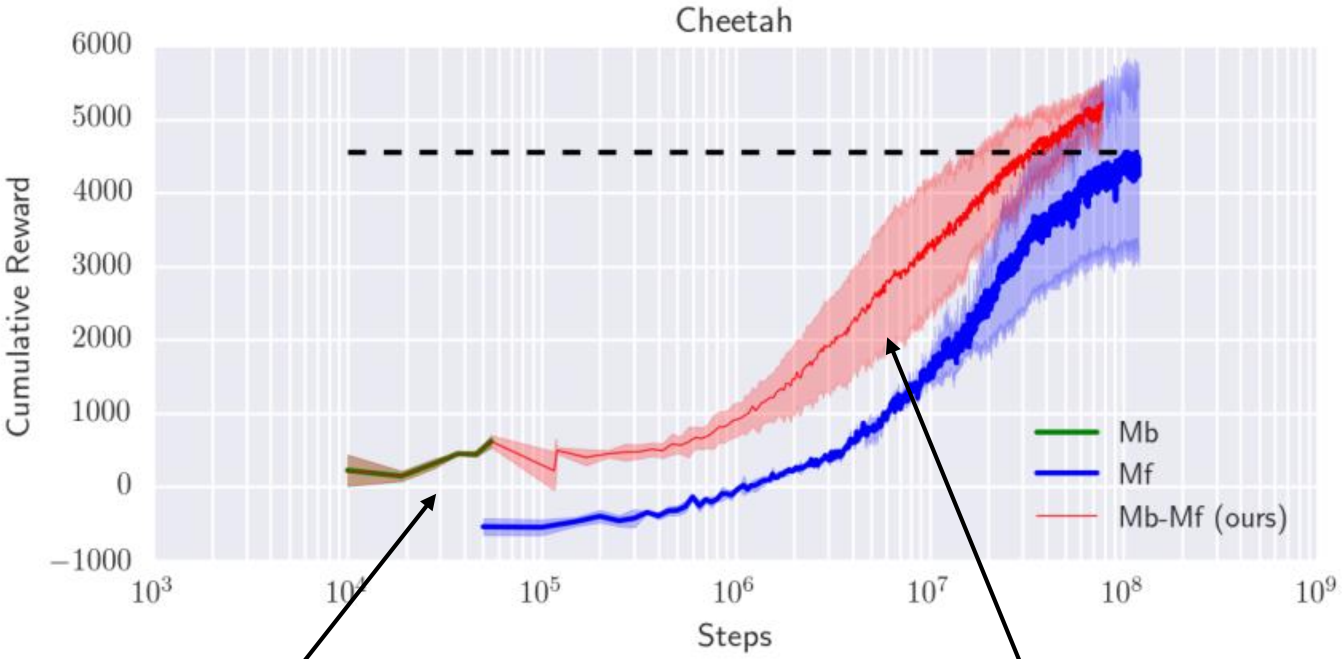
model-free training
(about 10 days…)





Nagabandi, Kahn, Fearing, L. ICRA 2018

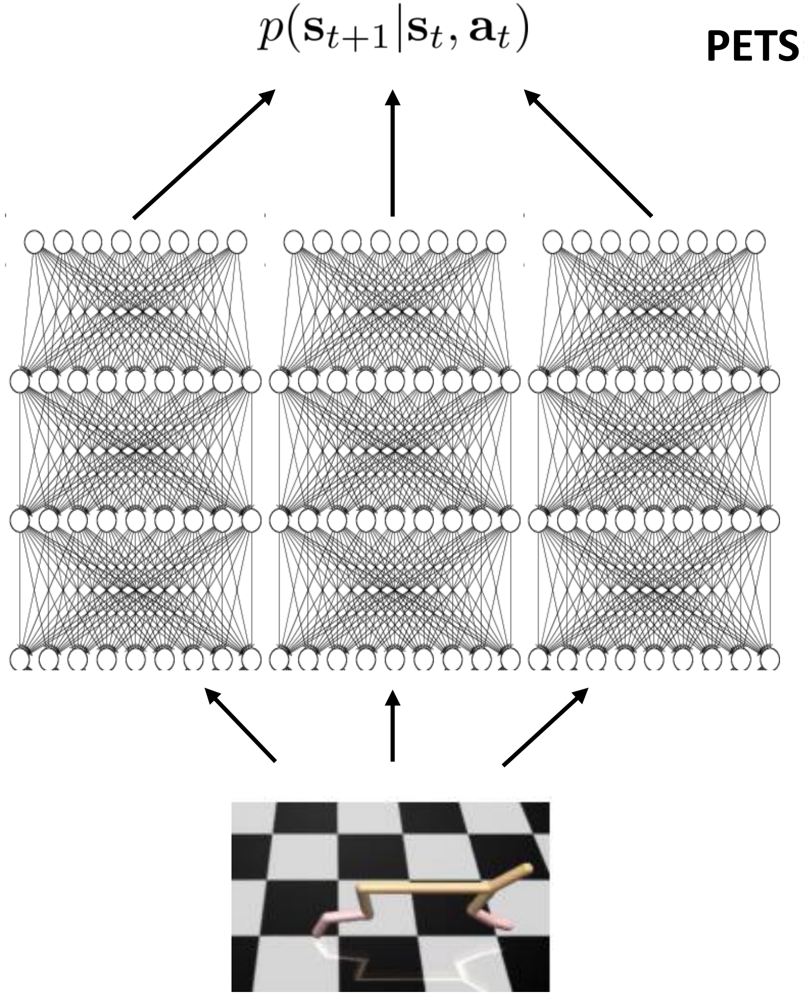
Bridging the Gap in Performance

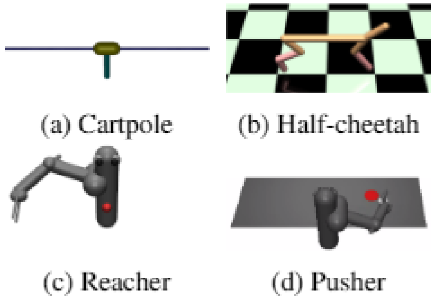


need to not overfit here...

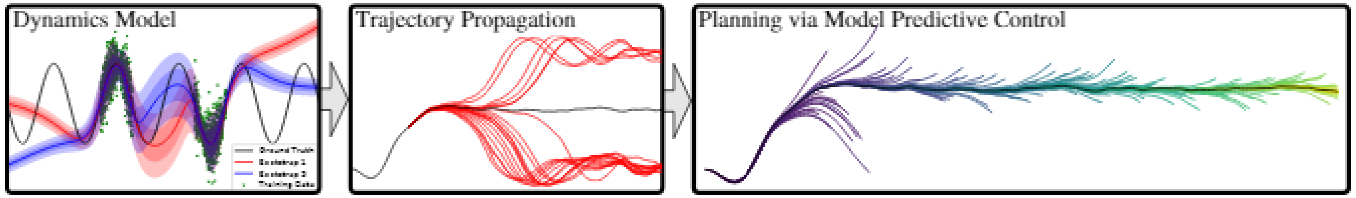
...but still have high capacity over here

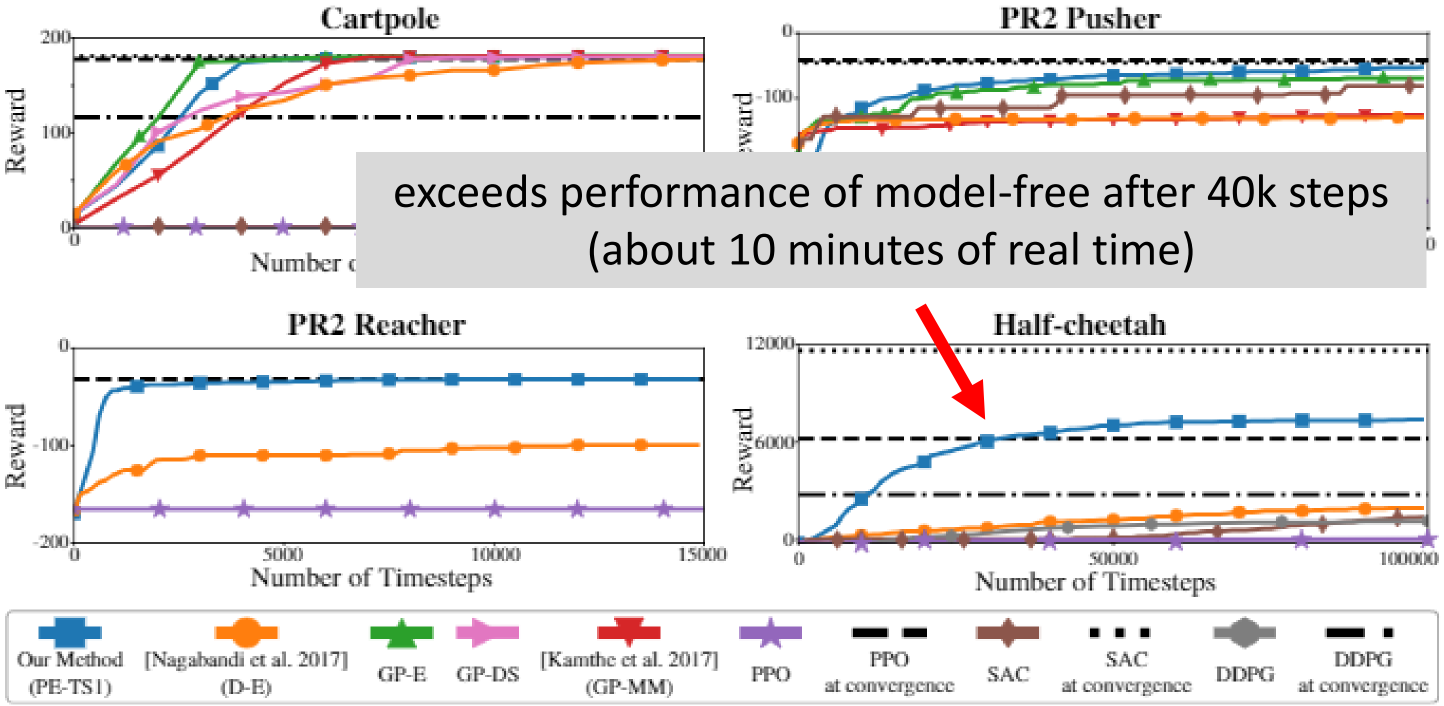
Incorporating Uncertainty





PETS: Probabilistic Ensembles with Trajectory Sampling

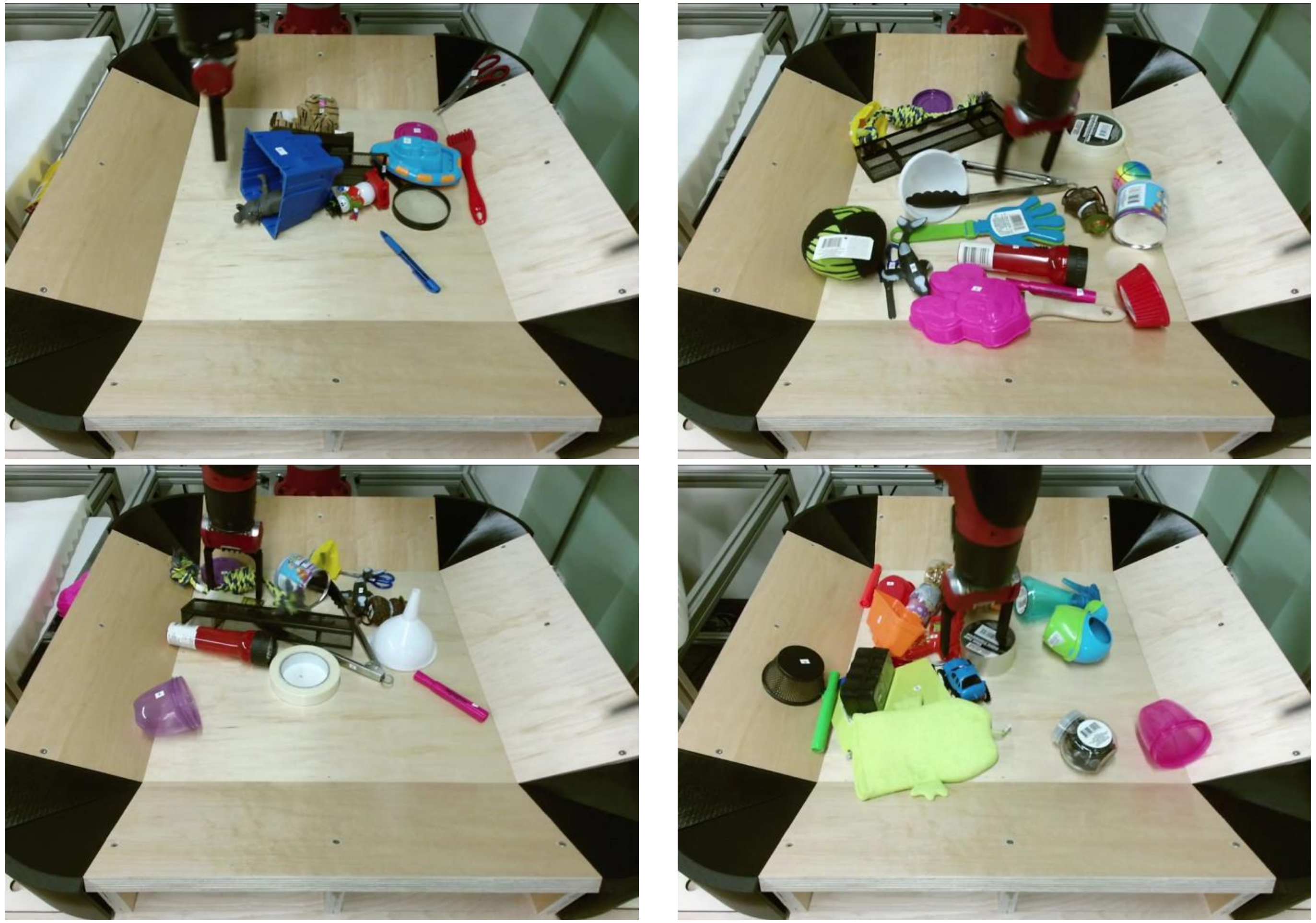




Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

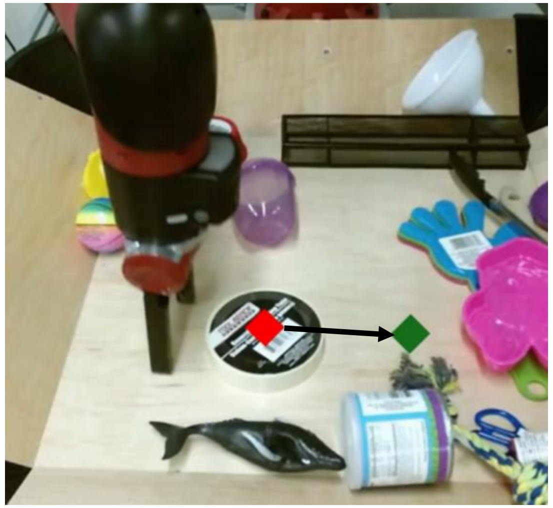
Chua, Calandra, McAllister, L.

Can we do this with pixels?



Ebert, Finn, Lee, Levine. 2017. Self-Supervised Visual Planning with Temporal Skip Connections. CoRL 2017.

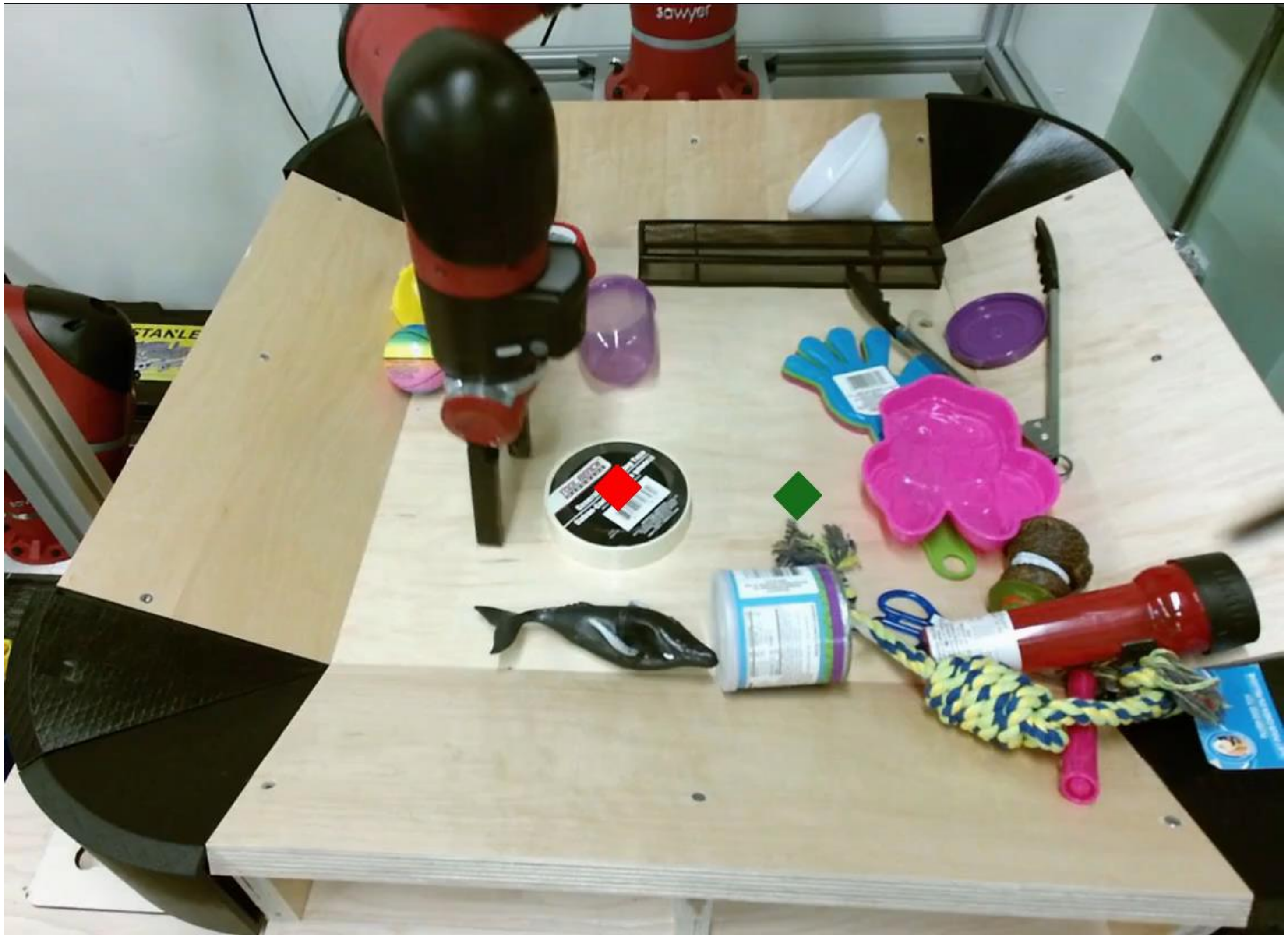
The model on pixels



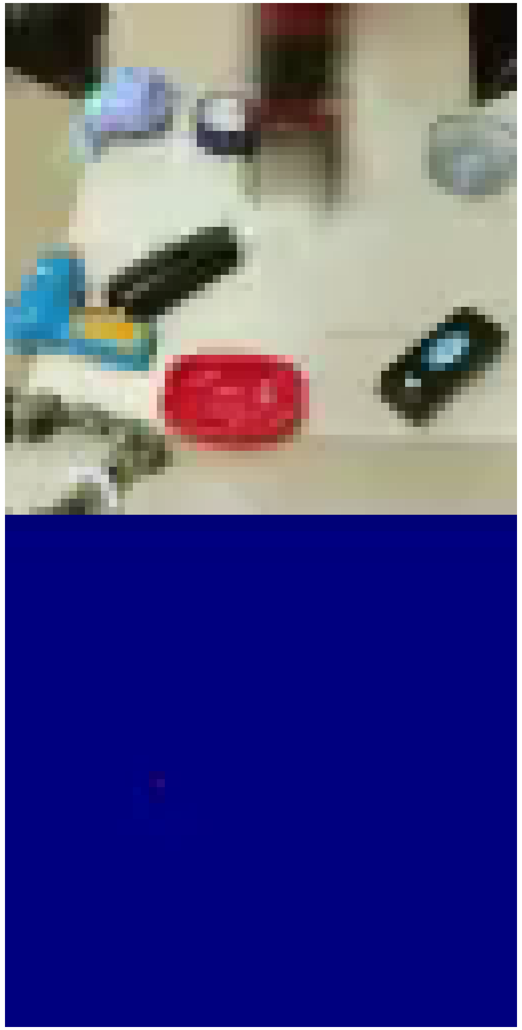
Designated Pixel ◆

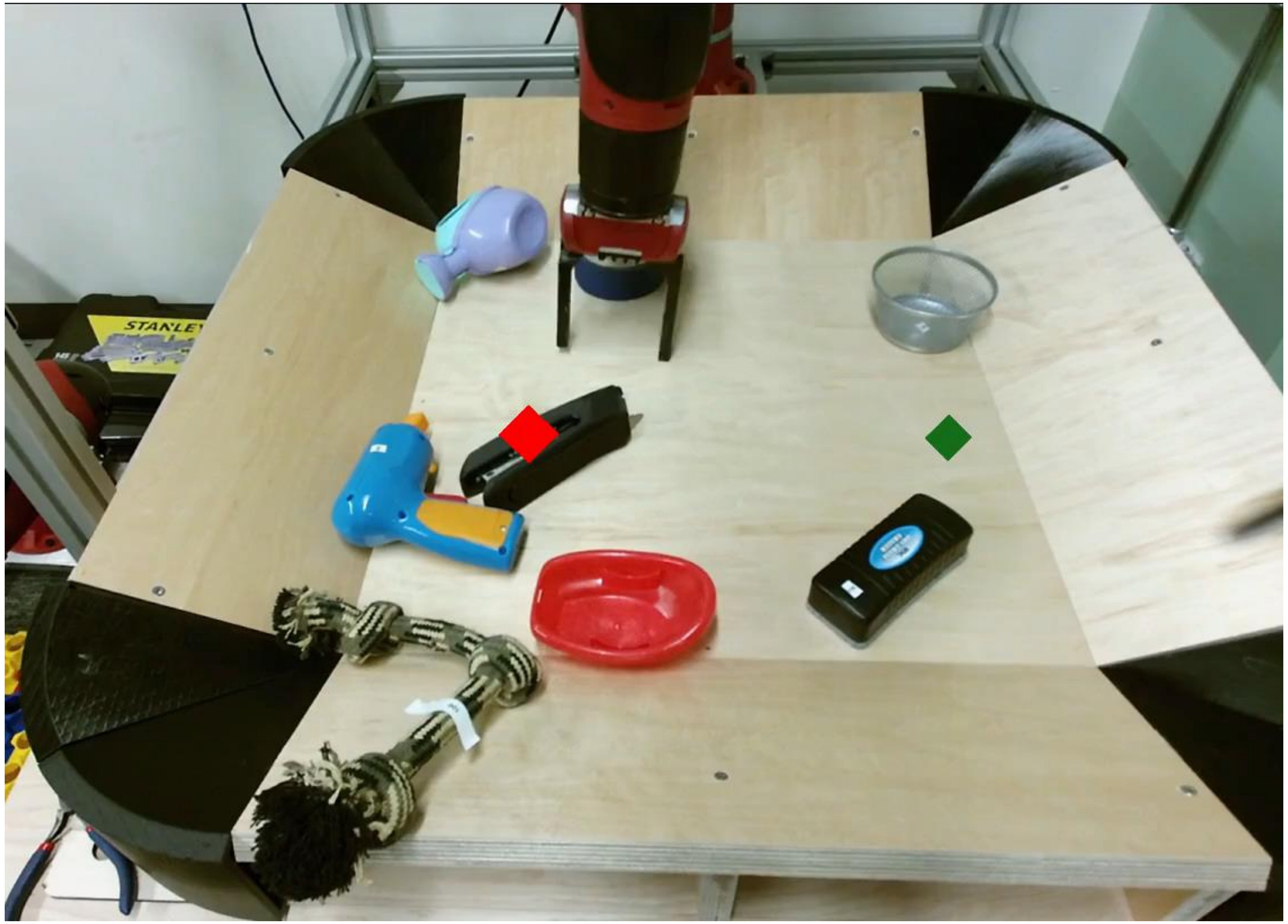
Goal Pixel ◆

Using the model to act

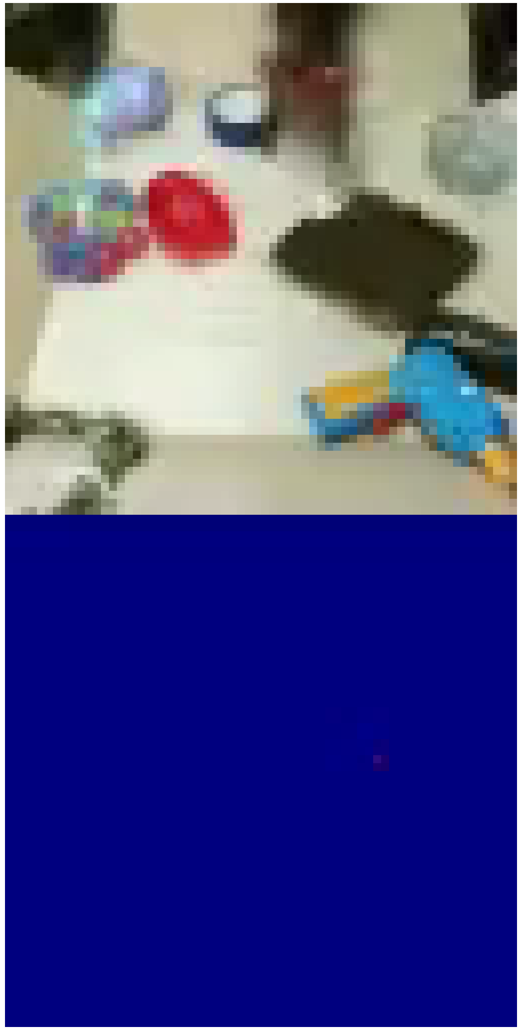


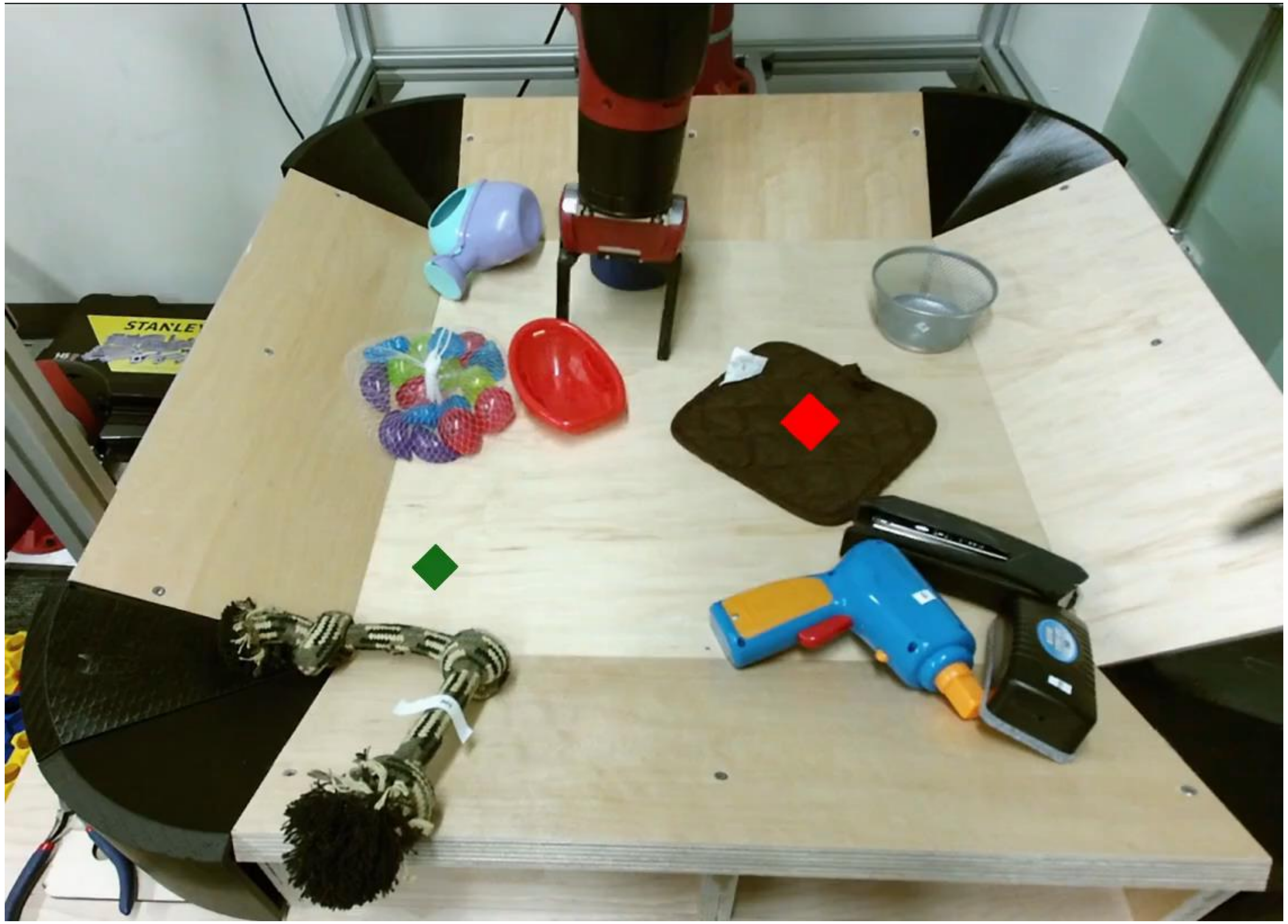
More examples





More examples





Today

- Actor-critic algorithm: reducing policy gradient variance using prediction
- Value-based algorithms: no more policy gradient, off-policy learning
- Model-based algorithms: control by predicting the future
- Open challenges and future directions

So… which algorithm do I use?

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)

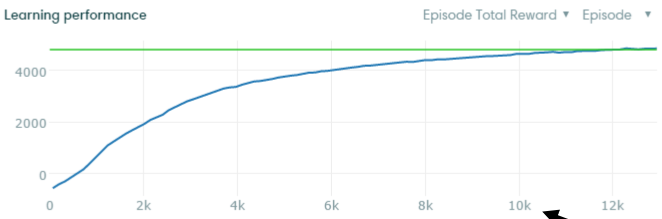
10x

model-based "shallow" RL
(e.g. PILCO)

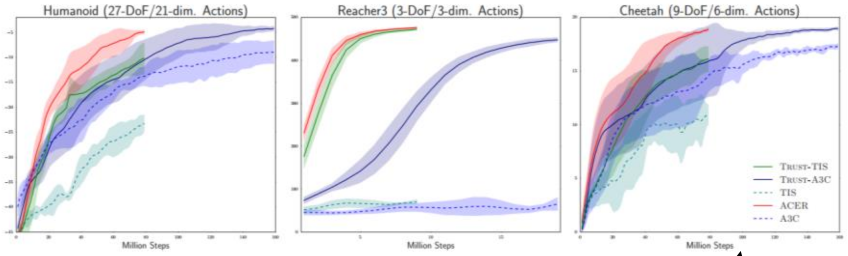

Evolution Strategies as a
Scalable Alternative to Reinforcement Learning

Tim Salimans [1] Jonathan Ho [1] Xi Chen [1] Ilya Sutskever [1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

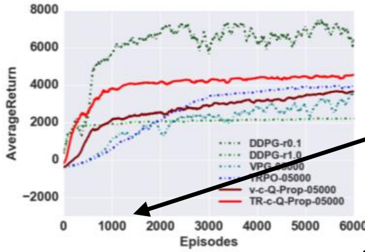


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

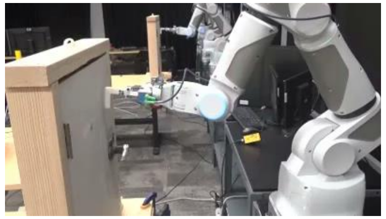
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

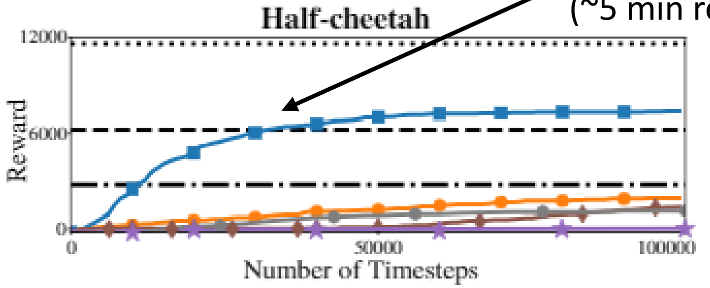


Gu et al. '16

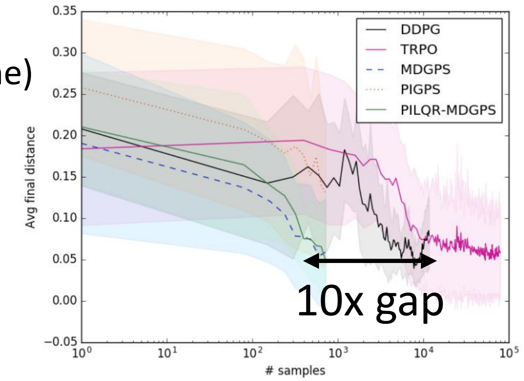
1,000,000 steps
(1,000 episodes)
(~3 hours real time)



30,000 steps
(30 episodes)
(~5 min real time)



Chua et a. '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

Which RL algorithm to use?

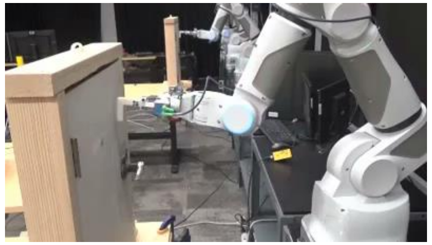
are you learning in a simulator?

no → how patient are you?

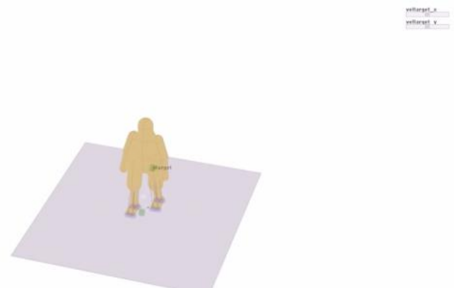
- impatient → model-based RL (GPS, PETS)
- I can wait → DDPG, NAF, SQL, SAC, TD3

yes → is simulation cost negligible compared to training cost?

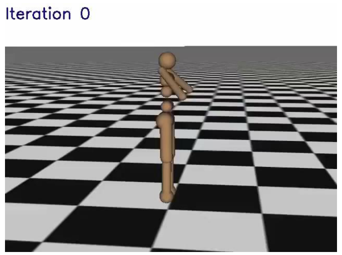
- no → DDPG, NAF, SQL, SAC, TD3
- yes → TRPO, PPO, A3C



Lillicrap et al. "Continuous control…"
Gu et al. "Continuous deep Q-learning…"
Haarnoja et al. "Reinforcement learning with deep energy-based…"
Haarnoja et al. "Soft actor-critic"
Fujimoto et al. "Addressing function approximation error…"



BUT: if you have a simulator, you can compute gradients through it – do you need model-free RL?



Today

- Actor-critic algorithm: reducing policy gradient variance using prediction
- Value-based algorithms: no more policy gradient, off-policy learning
- Model-based algorithms: control by predicting the future
- Open challenges and future directions

What are the big challenges in deep RL?

- Stability and hyperparameters
- Sample Complexity
- Generalization

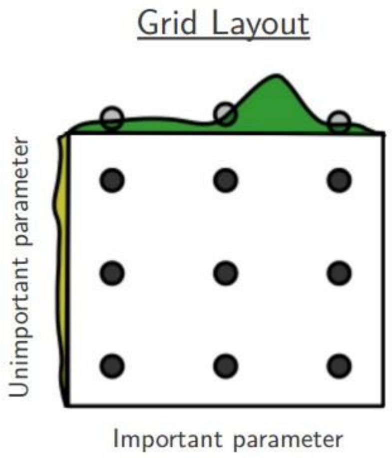
Stability and Hyperparameters

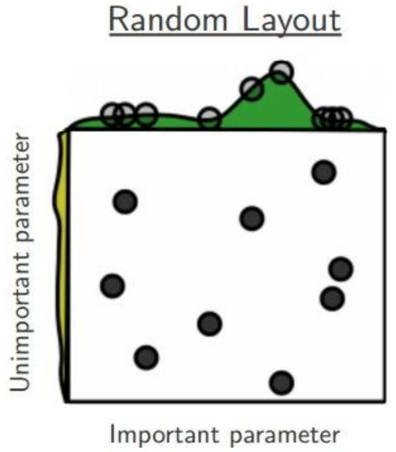
Stability and hyperparameter tuning

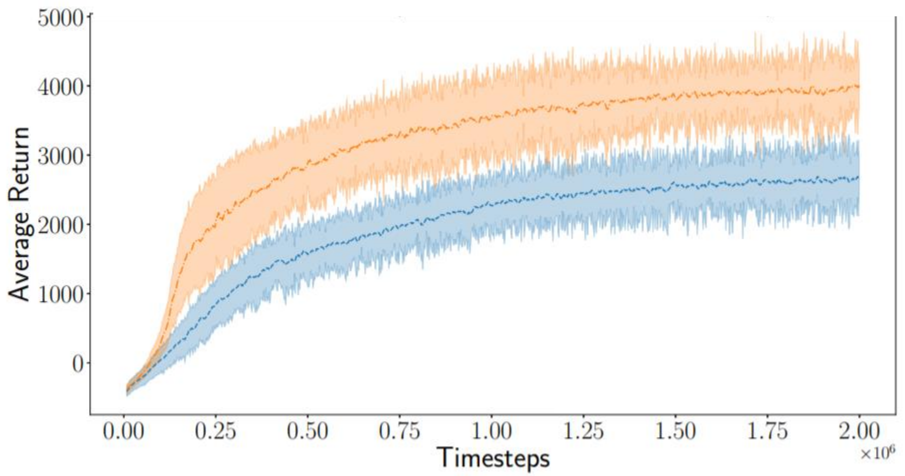
- Devising stable RL algorithms is very hard
- Q-learning/value function estimation
  - Fitted Q/fitted value methods with deep network function estimators are typically not contractions, hence no guarantee of convergence
  - Lots of parameters for stability: target network delay, replay buffer size, 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

Tuning hyperparameters

- Get used to running multiple hyperparameters
  - `learning_rate = [0.1, 0.5, 1.0, 5.0, 20.0]`
- Grid layout for hyperparameter sweeps OK when sweeping 1 or 2 parameters
- Random layout generally more optimal, the only viable option in higher dimensions
- Don't forget the random seed!
  - RL is self-reinforcing, very likely to get local optima
  - Don't assume it works well until you test a few random seeds
  - Remember that random seed is not a hyperparameter!



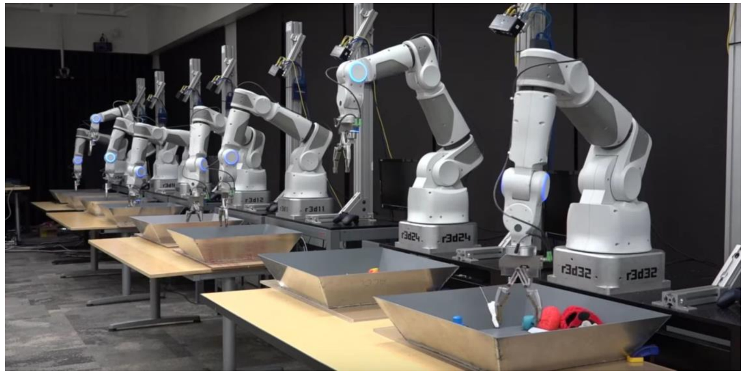




Henderson et al. '17, "Deep Reinforcement Learning that Matters."

The challenge with hyperparameters

- Can't run hyperparameter sweeps in the real world
  - How representative is your simulator? Usually the answer is "not very"
- Actual sample complexity = time to run algorithm x number of runs to sweep
  - In effect stochastic search + gradient-based optimization
- Can we develop more stable algorithms that are less sensitive to hyperparameters?



What can we do?

- Algorithms with favorable improvement and convergence properties
  - Trust region policy optimization [Schulman et al. ‘16]
  - Safe reinforcement learning, High-confidence policy improvement [Thomas ‘15]
- Algorithms that adaptively adjust parameters
  - Q-Prop [Gu et al. ‘17]: adaptively adjust strength of control variate/baseline

- More research needed here!
- Not great for beating benchmarks, but absolutely essential to make RL a viable tool for real-world problems

Sample Complexity

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)

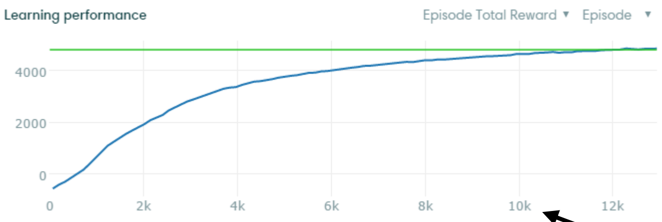
10x

model-based “shallow” RL
(e.g. PILCO)

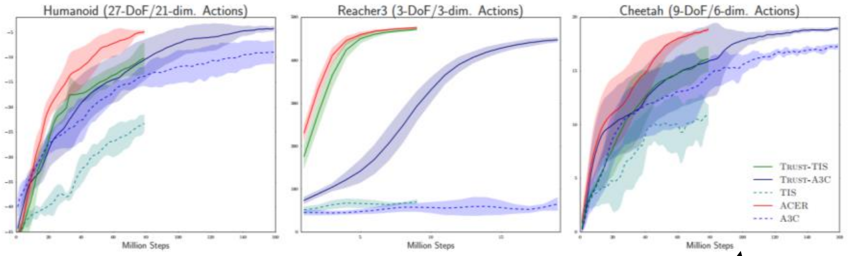

Evolution Strategies as a
Scalable Alternative to Reinforcement Learning

Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. ‘16)

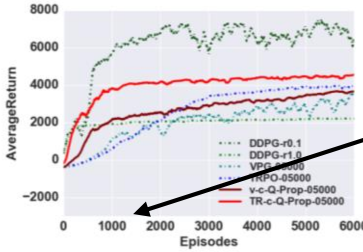


Wang et al. ‘17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

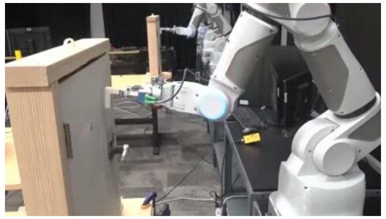
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

half-cheetah

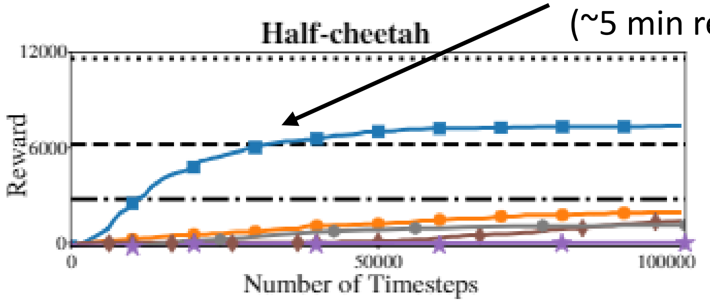


Gu et al. ‘16

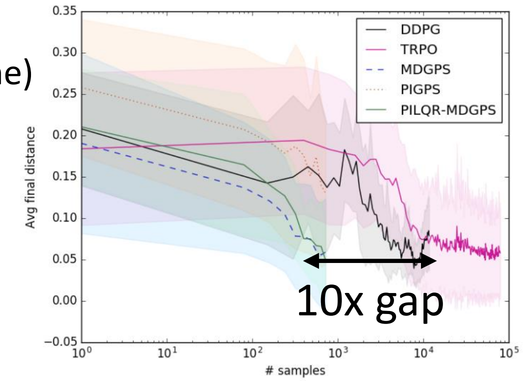
1,000,000 steps
(1,000 episodes)
(~3 hours real time)



30,000 steps
(30 episodes)
(~5 min real time)



Chua et a. ’18: Deep Reinforcement Learning in a Handful of Trials

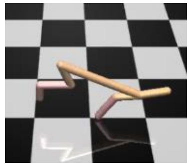


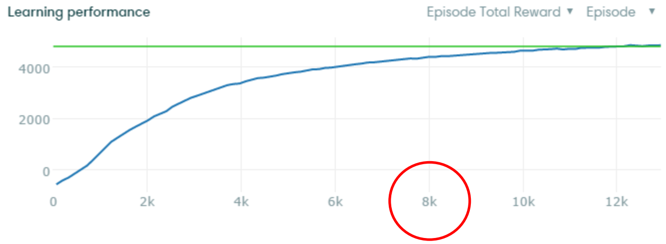
Chebotar et al. ’17 (note log scale)

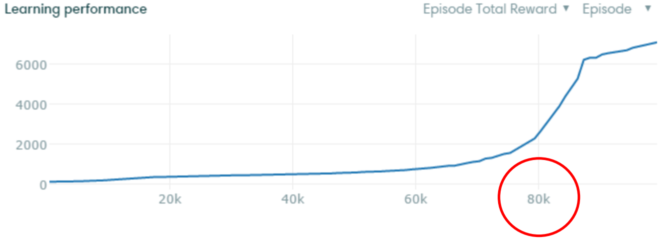
about 20
minutes of
experience on a
real robot

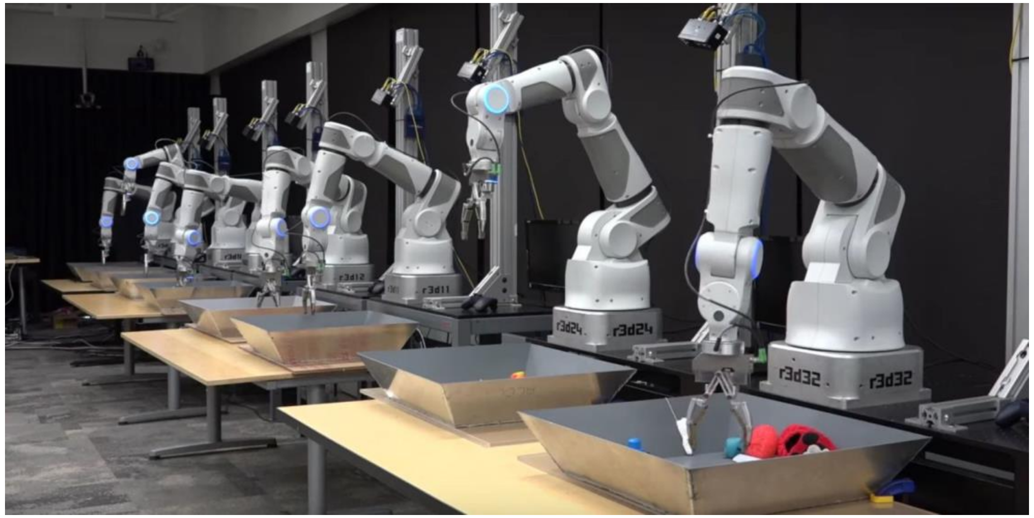
What about more realistic tasks?

- Big cost paid for dimensionality
- Big cost paid for using raw images
- Big cost in the presence of real-world diversity (many tasks, many situations, etc.)



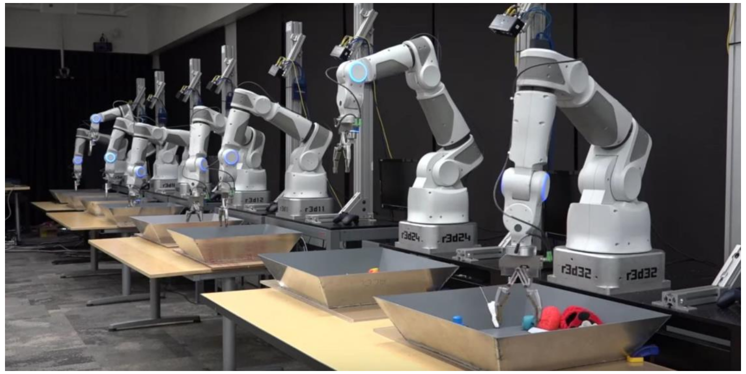






The challenge with sample complexity

- 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



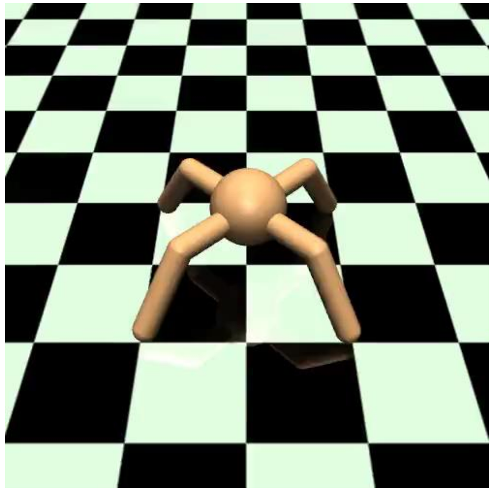
What can we do?

- Better model-based RL algorithms
- Design faster algorithms
  - Q-Prop (Gu et al. ‘17): policy gradient algorithm that is as fast as value estimation
  - Learning to play in a day (He et al. ‘17): Q-learning algorithm that is much faster on Atari than DQN
- Reuse prior knowledge to accelerate reinforcement learning
  - RL2: Fast reinforcement learning via slow reinforcement learning (Duan et al. ‘17)
  - Learning to reinforcement learning (Wang et al. ‘17)
  - Model-agnostic meta-learning (Finn et al. ‘17)

Generalization

Scaling up deep RL & generalization

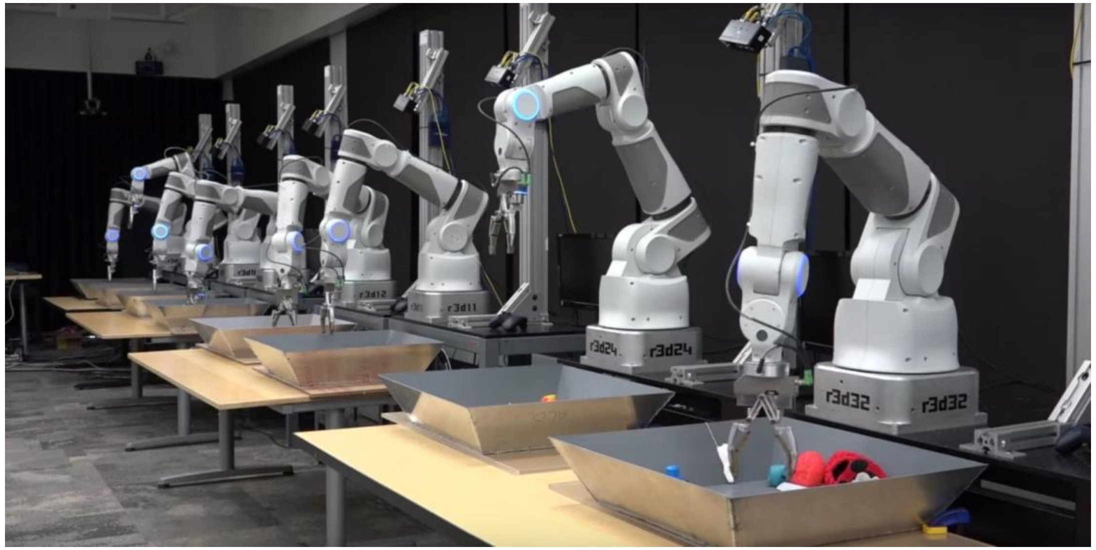
- Large-scale
- Emphasizes diversity
- Evaluated on generalization



- Small-scale
- Emphasizes mastery
- Evaluated on performance
- Where is the generalization?

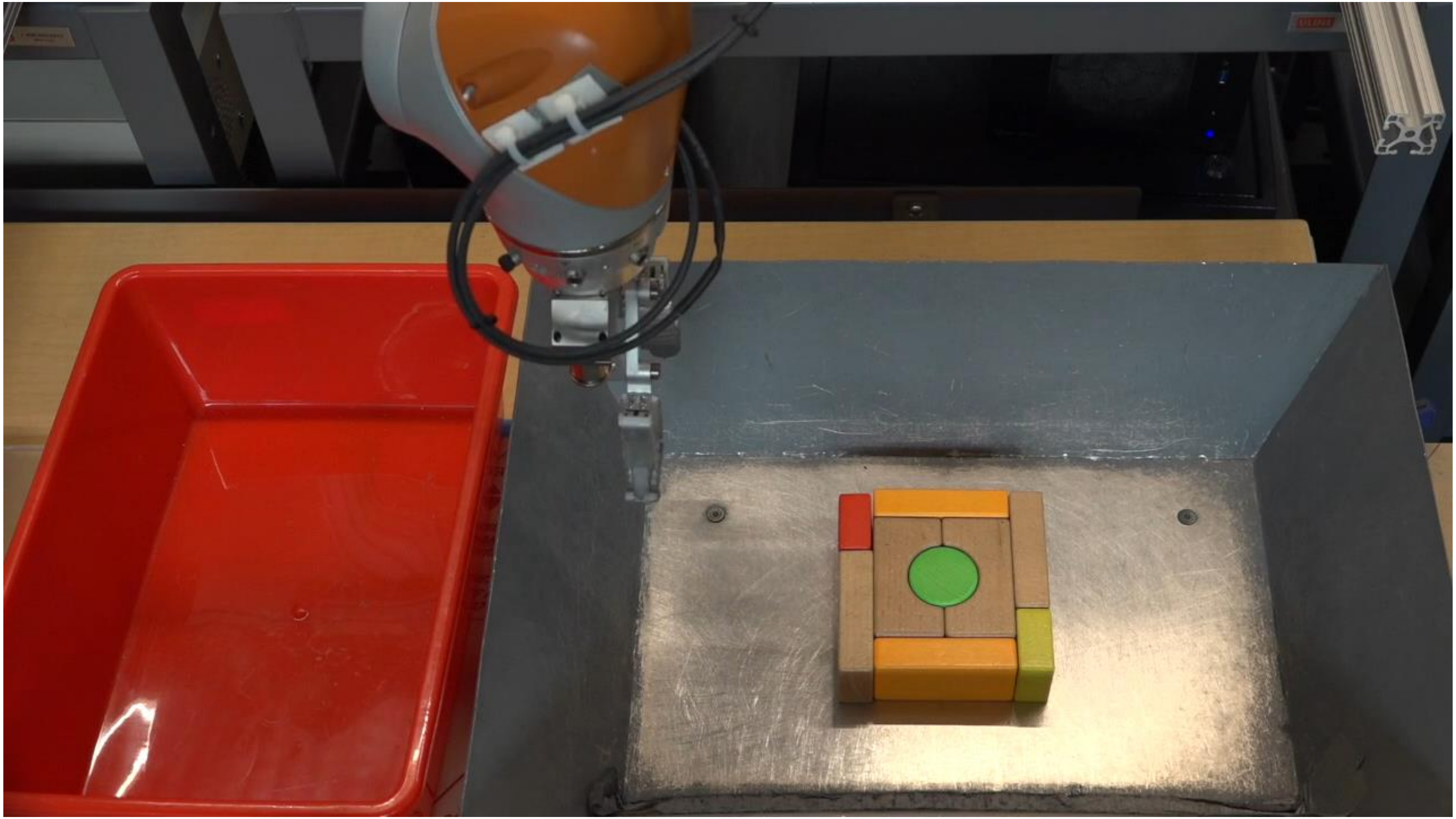
Generalizing from massive experience

Pinto & Gupta, 2015



Levine et al. 2016

Policy learned with large-scale Q-learning



Generalizing from multi-task learning

- Train on multiple tasks, then try to generalize or finetune
  - Policy distillation (Rusu et al. ‘15)
  - Actor-mimic (Parisotto et al. ‘15)
  - Model-agnostic meta-learning (Finn et al. ‘17)
  - many others…
- Unsupervised or weakly supervised learning of diverse behaviors
  - Stochastic neural networks (Florensa et al. ‘17)
  - Reinforcement learning with deep energy-based policies (Haarnoja et al. ‘17)
  - many others…

Generalizing from prior knowledge & experience

- Can we get better generalization by leveraging off-policy data?
- Model-based methods: perhaps a good avenue, since the model (e.g. physics) is more task-agnostic
- What does it mean to have a “feature” of decision making, in the same sense that we have “features” in computer vision?
  - Options framework (mini behaviors)
    - Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning (Sutton et al. ’99)
    - The option-critic architecture (Bacon et al. ‘16)
  - Muscle synergies & low-dimensional spaces
    - Unsupervised learning of sensorimotor primitives (Todorov & Gahramani ’03)

Reward specification

- 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?

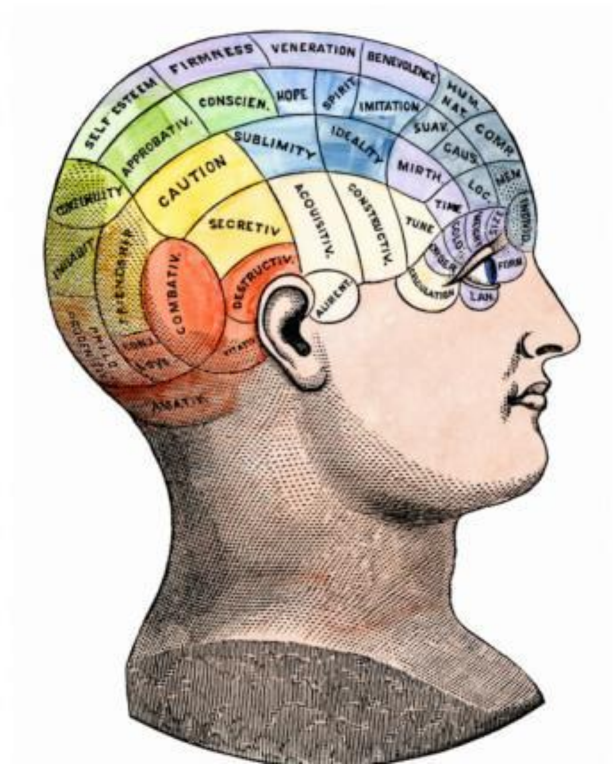
reward

Mnih et al. '15

reinforcement learning agent

what is the reward?

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

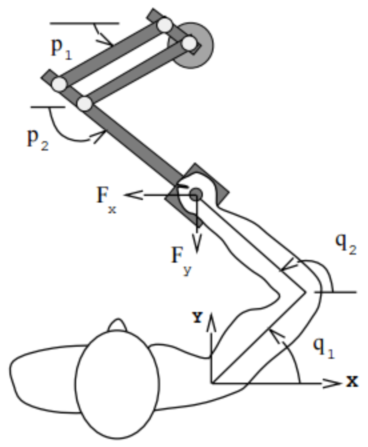
Deep models are what allow reinforcement learning algorithms to solve complex problems end to end!

What can deep learning & RL do well now?

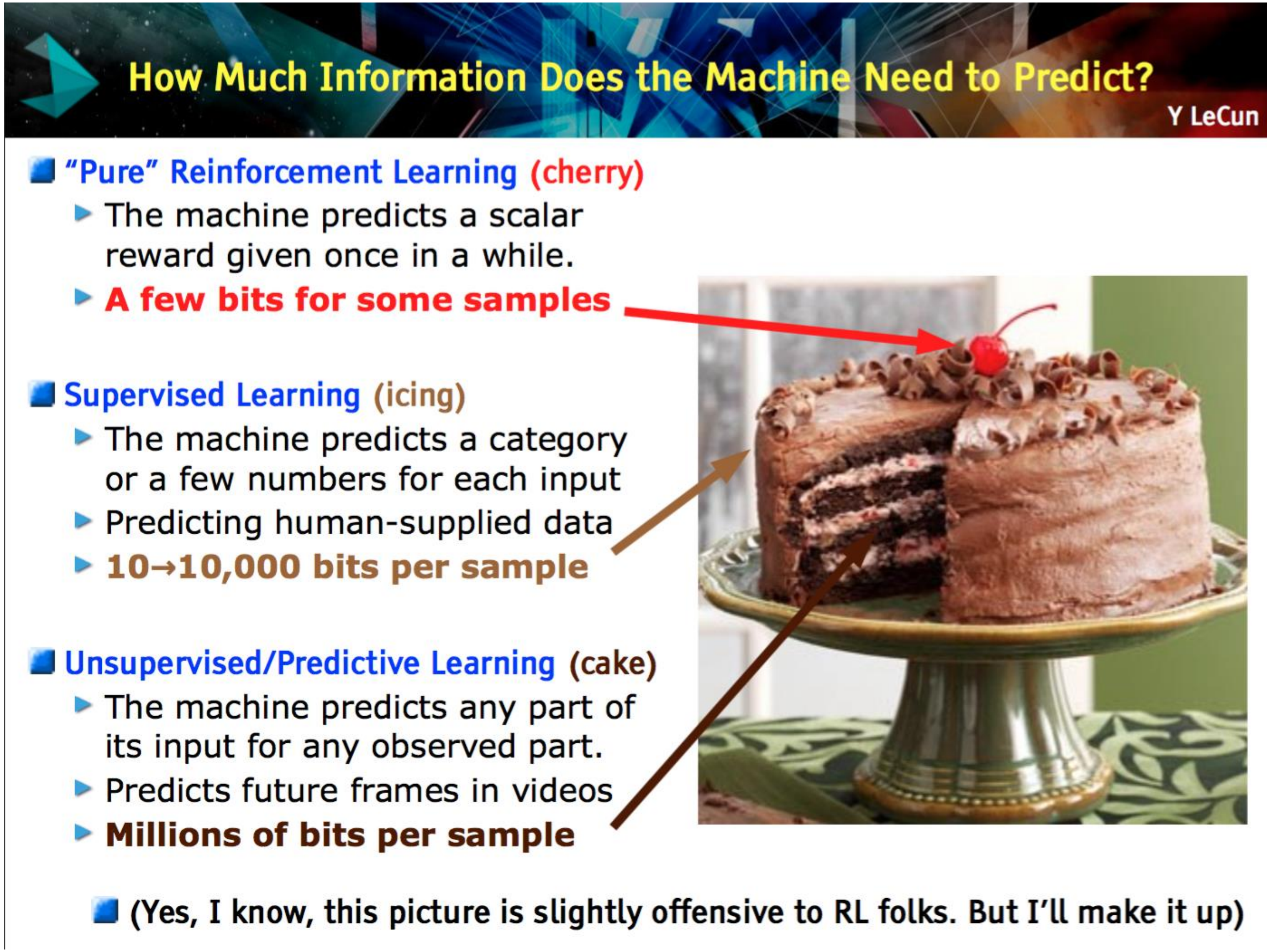
- Acquire high degree of proficiency in domains governed by simple, known rules
- Learn simple skills with raw sensory inputs, given enough experience
- Learn from imitating enough human-provided expert behavior

What has proven challenging so far?

- Humans can learn incredibly quickly
  - Deep RL methods are usually slow
- Humans can reuse past knowledge
  - Transfer learning in deep RL is an open problem
- Not clear what the reward function should be
- Not clear what the role of prediction should be



What is missing?



Where does the supervision 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!
- Pay attention to generative models, prediction
- Carefully understand the relationship between RL and other ML fields

to learn more: see rail.eecs.berkeley.edu/deeprlcourse

OpenAI Gym