Introduction to Reinforcement Learning

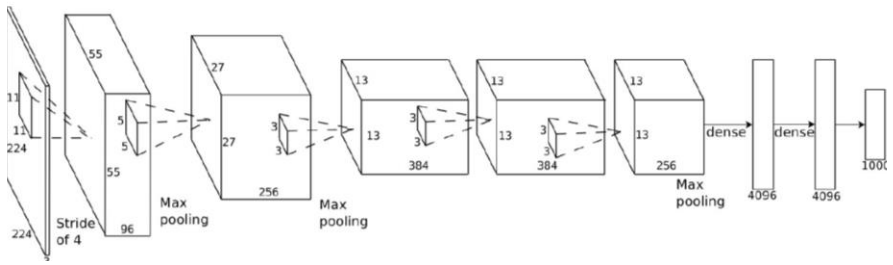
CS 285

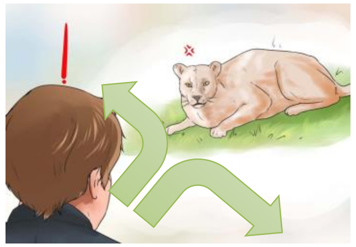
Instructor: Sergey Levine
UC Berkeley

Definitions

The policy can be represented as the distribution over actions conditioned on some observations. Policy pi, typically is represented with theta, to show that it depends on parameters (for DRL, params are part of a NN).

Terminology & notation





$$\begin{tikzcd}
\mathbf{o}_t & \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) & \mathbf{a}_t \arrow[ll, bend left]
\end{tikzcd}$$

$\mathbf{s}_t$ – 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)

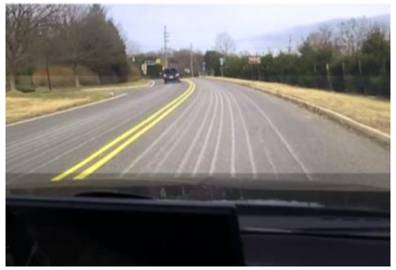
select one action over the distribution of actions

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

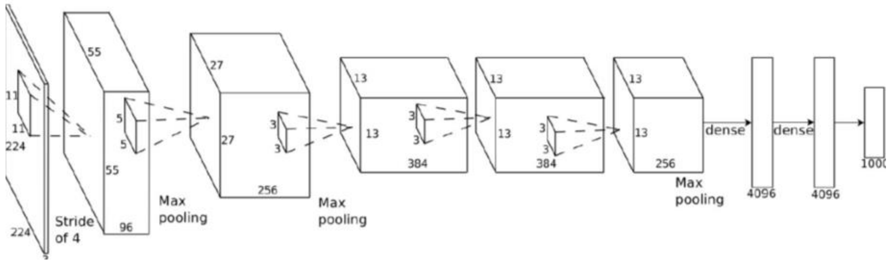
Markov property independent of $\mathbf{s}_{t-1}$

This satisfies the Markov property! s_t+1 is independent of s_t-1.
The observation is a stochastic representation of the state that may or may not contain all the information necessary to infer the full state.
- Fully observed RL (access to state)
- Partially observed RL (access to obs only)

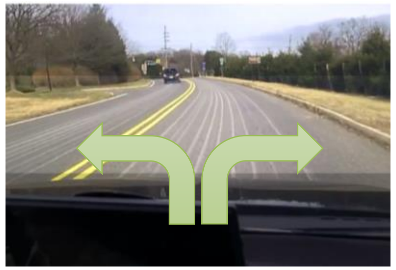
Imitation Learning



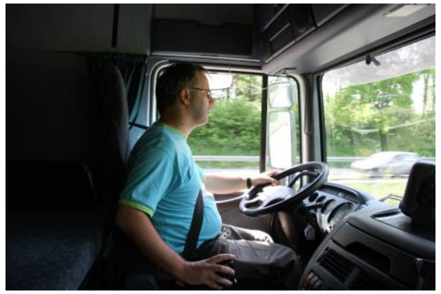
$\mathbf{o}_t$



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

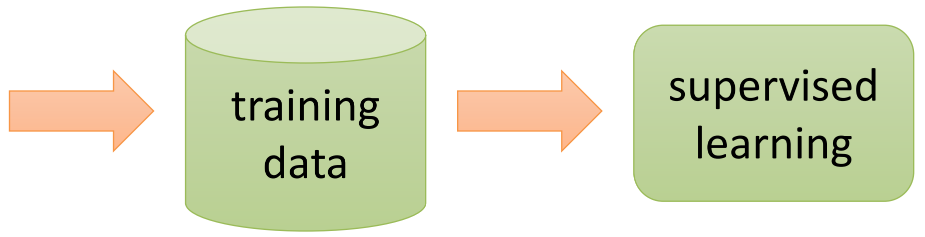


$\mathbf{a}_t$



$\mathbf{o}_t$

$\mathbf{a}_t$

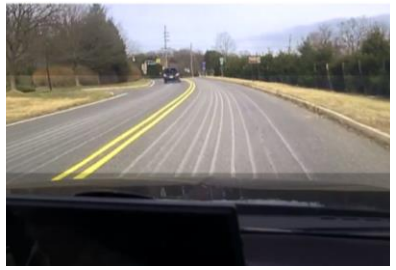


$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

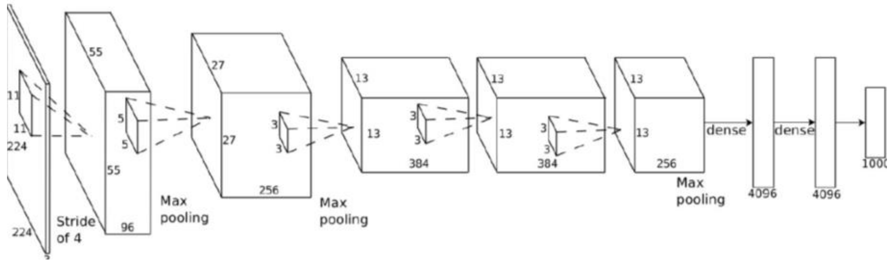
Images: Bojarski et al. '16, NVIDIA

The reward function formalizes the objective of the RL model. It is a scalar valued function of the state and action and tells us which states and actions are better! Although, it's possible for it to be a function of the state only. Goal: take actions that lead to a higher reward function. The heart of the RL problem is to choose rewards "later", not just right now.

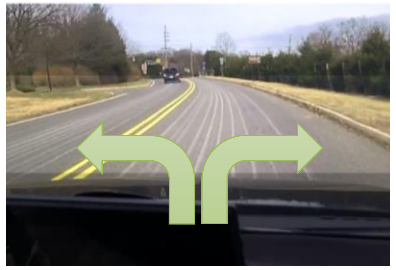
Reward functions



$\mathbf{o}_t$



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



$\mathbf{a}_t$

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

state

probability matrix of the probability of being in a state in t+1, given the state in t.

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

The Markov Chain by itself is not a decision-making problem because there is no notion of actions!

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

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

Definitions

Here, we include the notion of actions.
- we add action space
- we add reward function

Andrey Markov
Richard Bellman

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!)

because its 3D! The next state, the current state, and the current action

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

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

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

"probability of being in state i if you were previously in state j and took action k"

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

now s_t+1 is dependent on the action too!

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

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

reward is a function of both states and actions

$r(s_t, a_t)$ – reward

Richard Bellman

Definitions

partially observed Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \boxed{\mathcal{O}}, \mathcal{T}, \boxed{\mathcal{E}}, r\}$

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

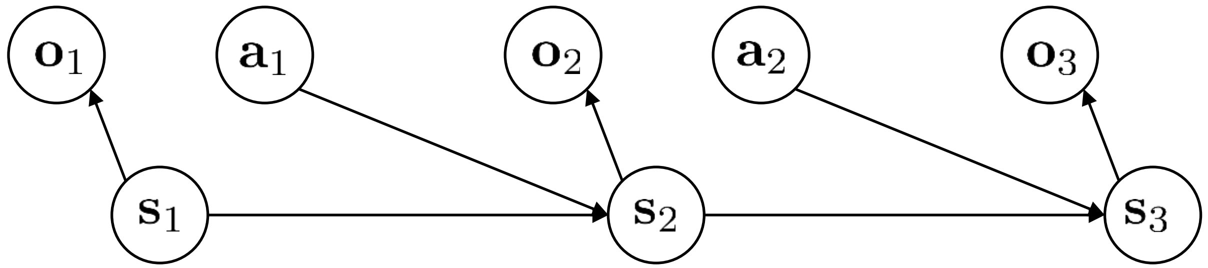
$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

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

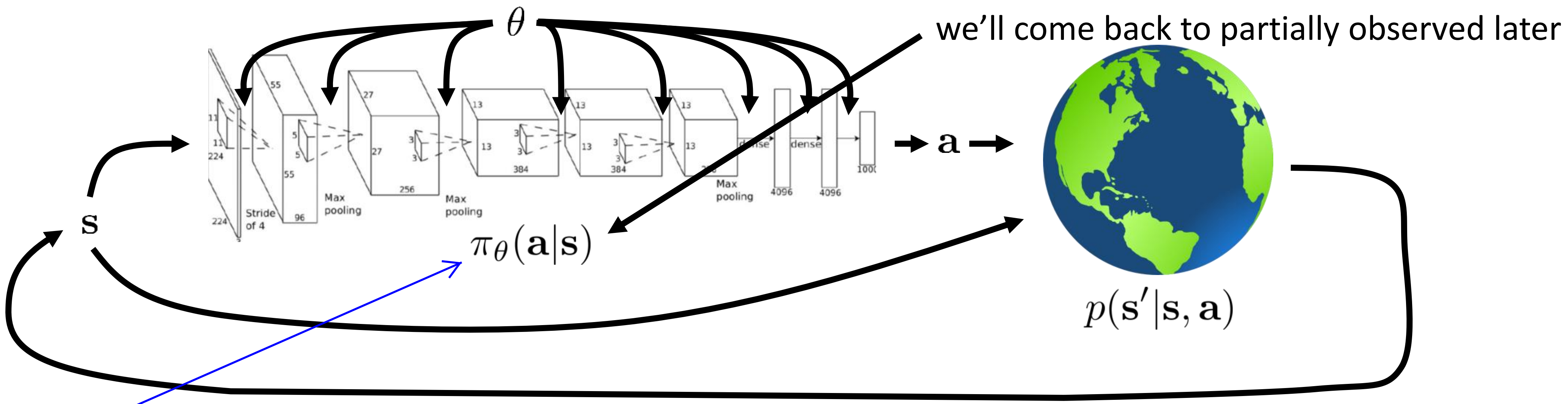
$\mathcal{T}$ – transition operator (like before)

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

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



The goal of reinforcement learning



θ
we'll come back to partially observed later
$\mathbf{s}$
$\mathbf{a}$
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s},\mathbf{a})$
Stride of 4
Max pooling
dense

the goal is to learn the policy! To find the policy we need to find the parameters!

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

Probability trajectory distribution: a sequence of states and actions

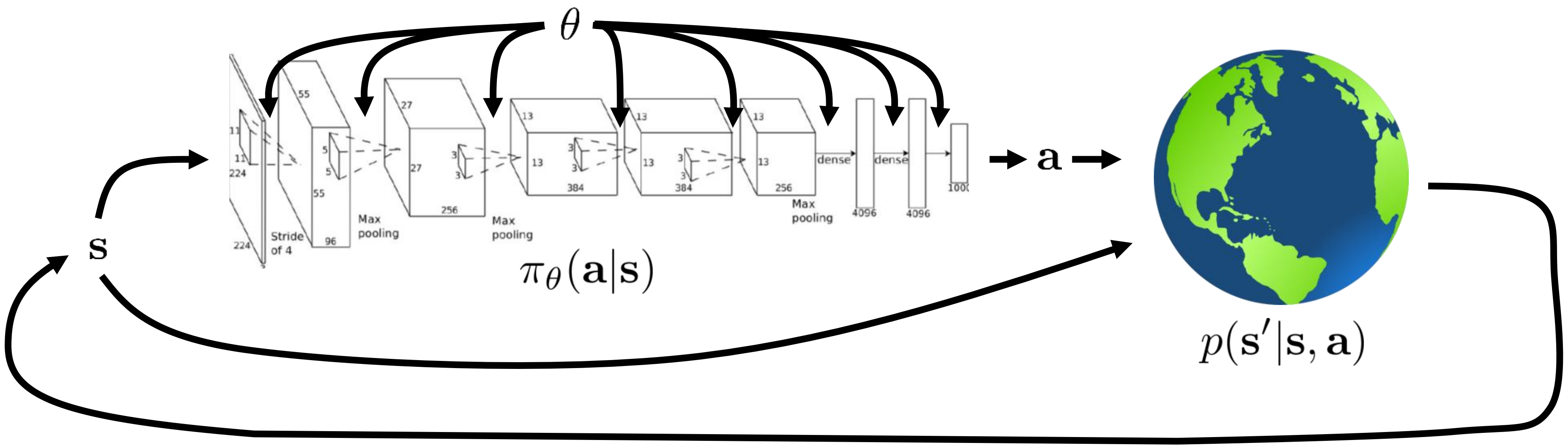
for Markov chains, we have a prob vector of initial states

probability of an action given the state

The objective of RL is to maximize the expected value of the sum of rewards under the trajectory distribution.

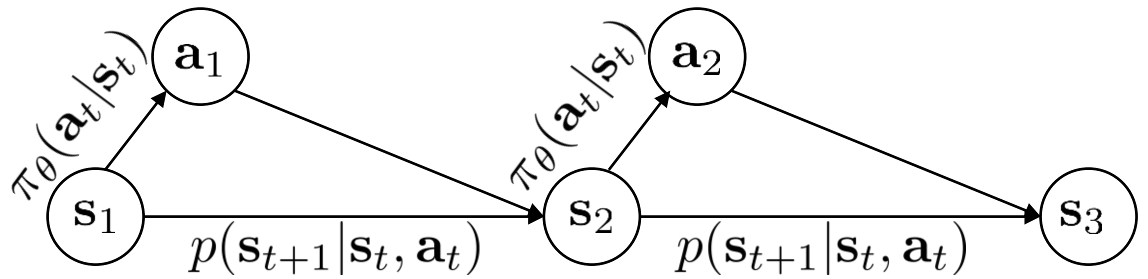
$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

The goal of reinforcement learning



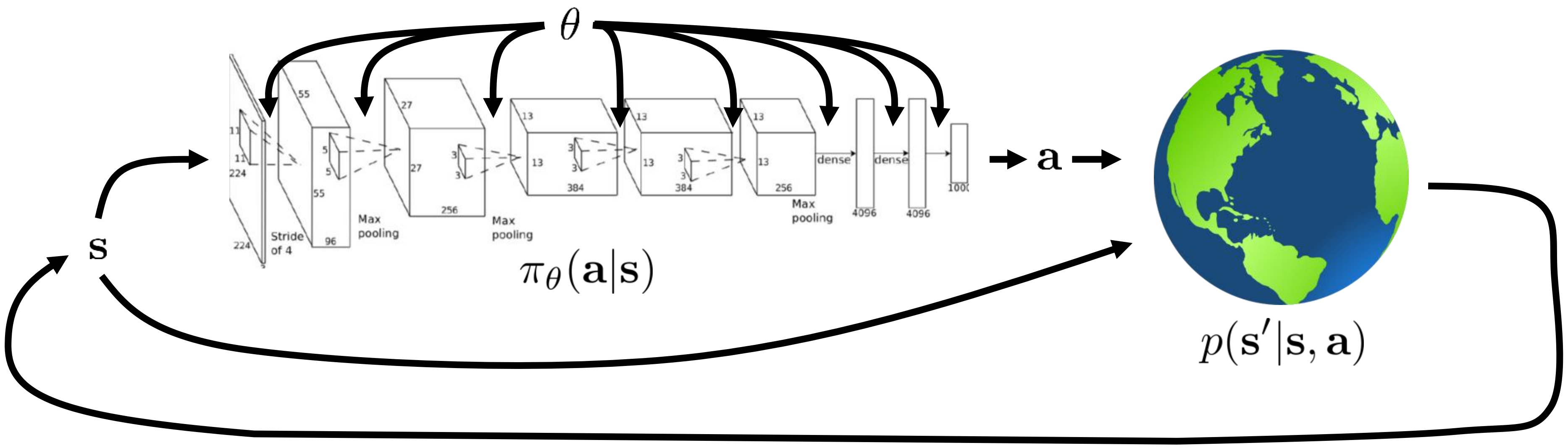
θ
$\mathbf{s}$
$\mathbf{a}$
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s},\mathbf{a})$
Stride of 4
Max pooling
dense

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



$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$

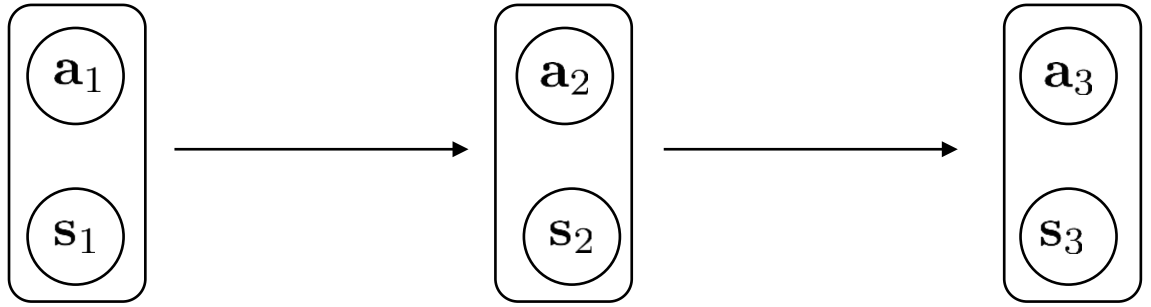
The goal of reinforcement learning



We can group the state and action together into an augmented state. So the augmented states actually form a markov chain! So the trajectory distribution is a product of the probability matrix and the policy

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

$$p((\mathbf{s}_{t+1},\mathbf{a}_{t+1})|(\mathbf{s}_t,\mathbf{a}_t)) =$$
$$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



Finite horizon case: state-action marginal

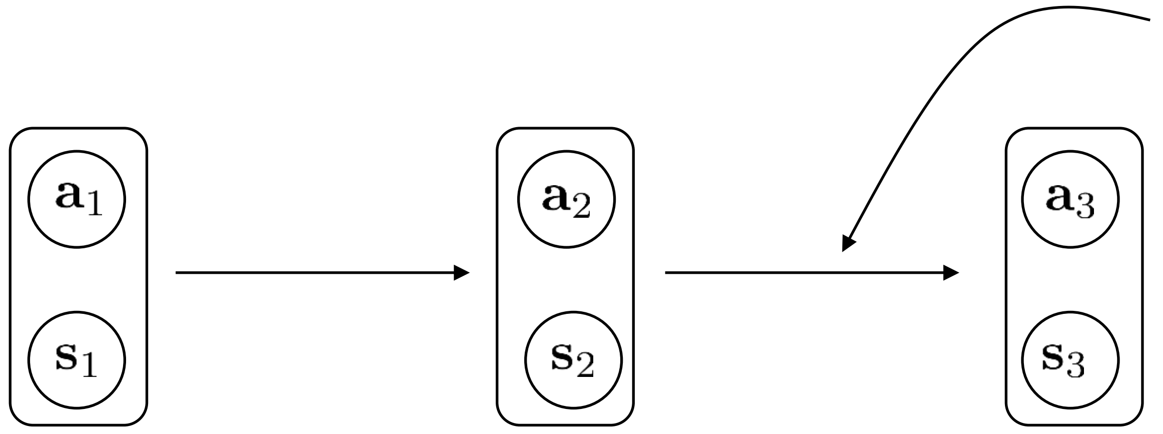
$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

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

$p_\theta(\mathbf{s}_t,\mathbf{a}_t)$ state-action marginal

This math trick is helpful in the infinite horizon case

$$p((\mathbf{s}_{t+1},\mathbf{a}_{t+1})|(\mathbf{s}_t,\mathbf{a}_t)) = p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



Infinite horizon case: stationary distribution

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

what if $T = \infty$?

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

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

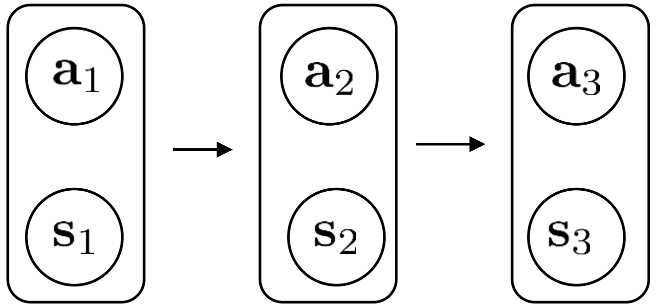
stationary = the same before and after transition

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

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

(always exists under some regularity conditions)

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



state-action transition operator

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

Infinite horizon case: stationary distribution

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

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

what if $T = \infty$?

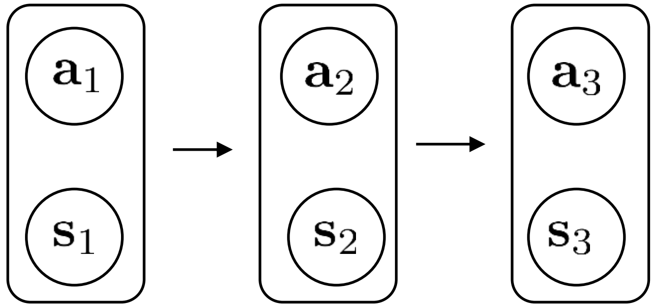
does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution?

$\mu = \mathcal{T}\mu$ $\quad (\mathcal{T} - \mathbf{I})\mu = 0 \quad \mu = p_\theta(\mathbf{s},\mathbf{a})$ stationary distribution

stationary = the same before and after transition

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

(always exists under some regularity conditions)



state-action transition operator

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

Expectations and stochastic systems

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

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

In RL, we almost always care about *expectations*

$r(\mathbf{x})$ – *not* smooth

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

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

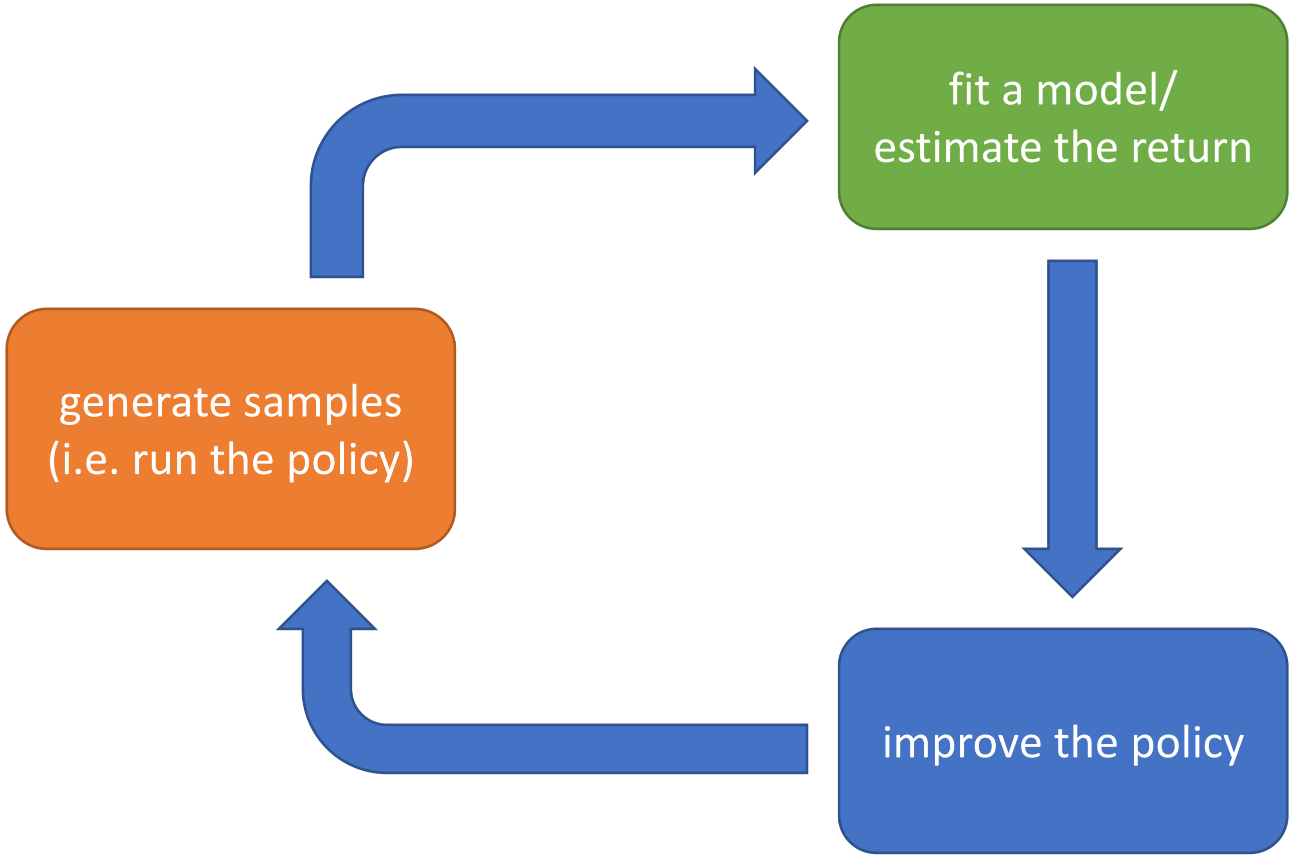
The reward function appears to be discontinuous, so you can't use gradient-based methods because the reward is not continuous or differentiable. But if you represent the action (fall off or don't fall off) as a distribution, then the exponential value of the reward is now smooth and differentiable in theta.

This is why RL algorithms can optimize seemingly discontinuous and non-differentiable reward functions using gradient-based methods (like NN).

Algorithms

The anatomy of a reinforcement learning algorithm

Pretty much all the algorithms have these three parts.

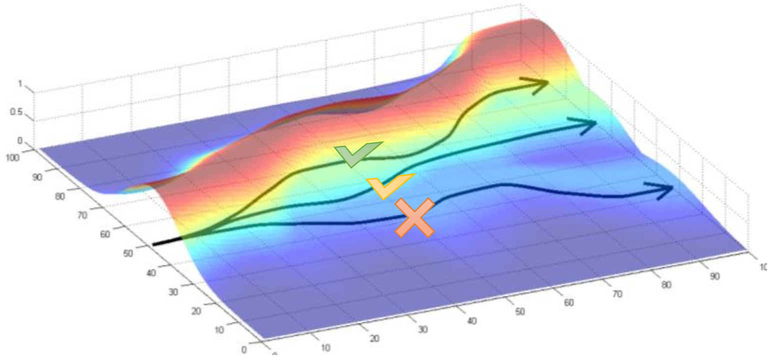

fit a model/ estimate the return
improve the policy
generate samples (i.e. run the policy)

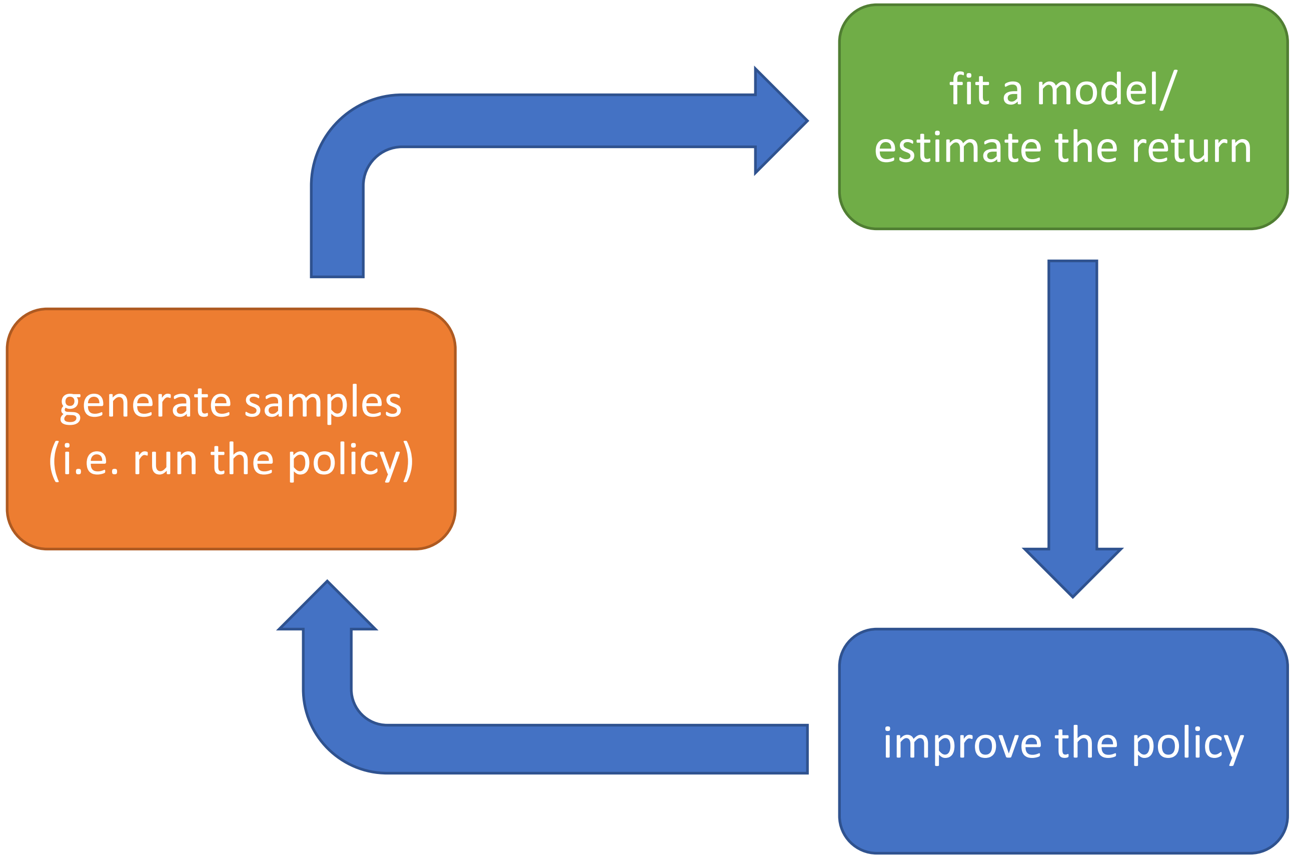
We need to learn some type of model. It could an explicit model, such as the dynamics, which is what we'd do in model-based RL, or it could be implicit, such as learning the value function. So we want to estimate something about the policy, such as how it's doing,what kind of rewards it's attaining

We must learn through trial & error, so we have to run our policy in our environment to get samples. These samples form trajectories

A simple example

This is a very simple Policy Gradient algorithm




fit a model/ estimate the return
improve the policy
generate samples (i.e. run the policy)

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

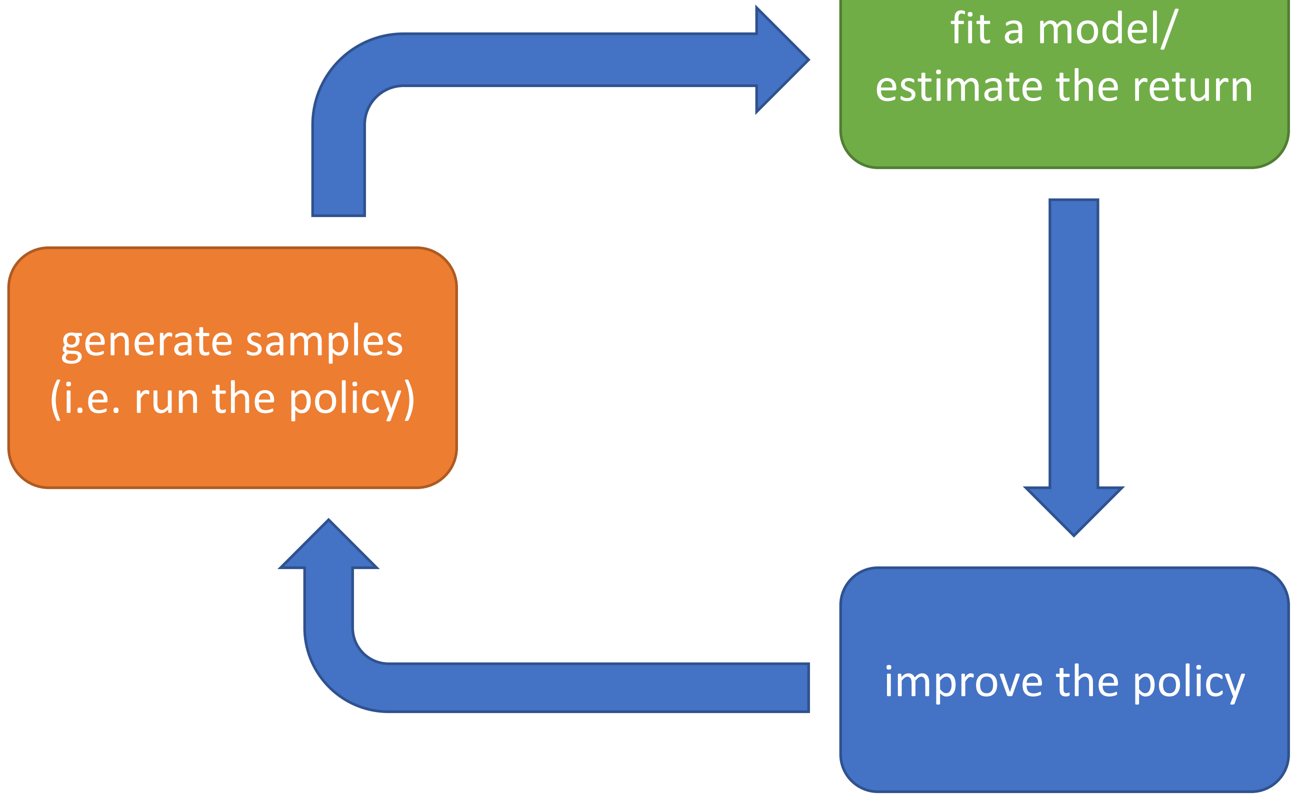
sum up the rewards. This tells us how good our policy is.

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

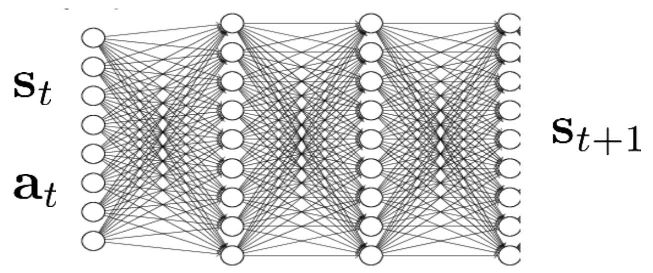
run gradient descent

Another example: RL by backprop

this is what we do in model-based RL



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



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

Which parts are expensive?

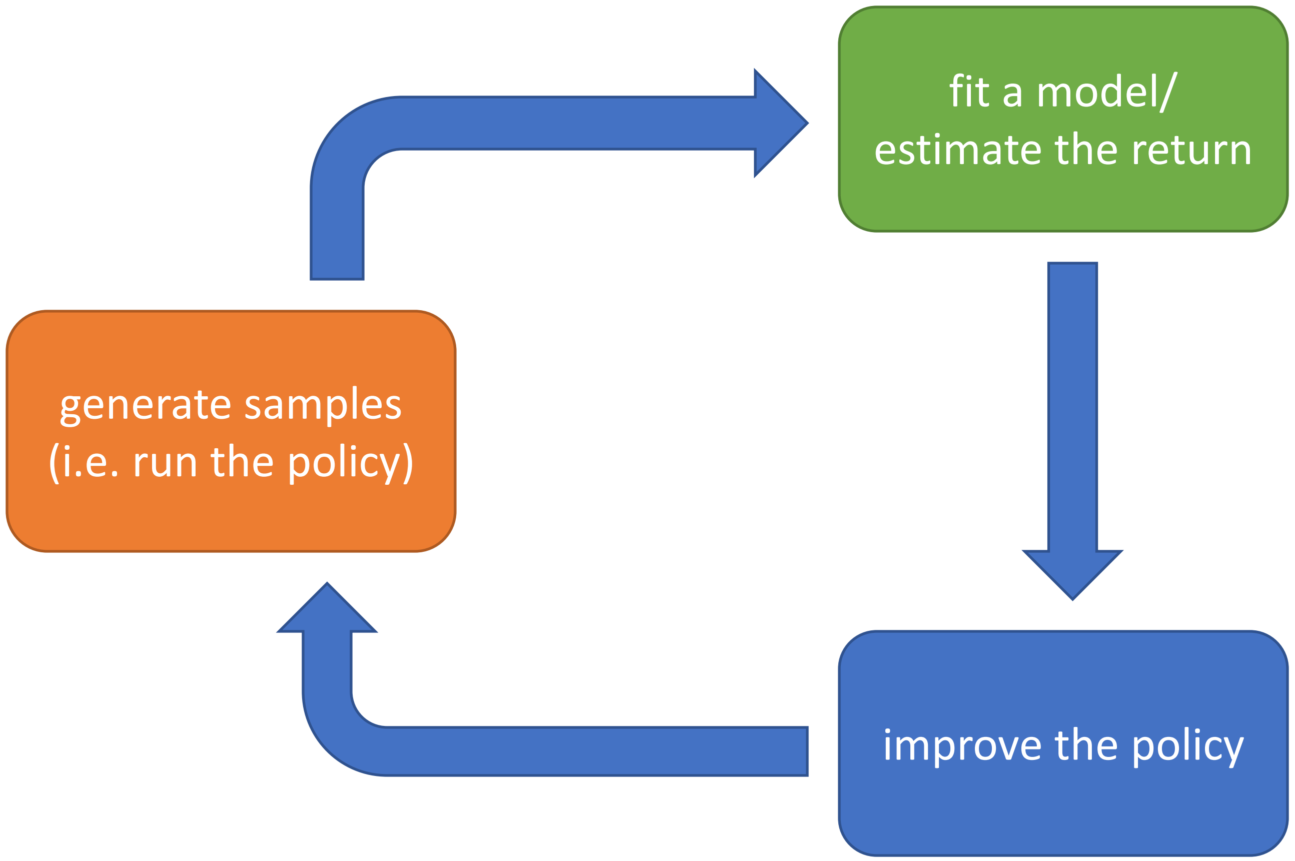
The algorithm you select will likely depend on which parts are cheap and which are expensive

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

trivial, fast

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

MuJoCo simulator:
up to 10000x real time


fit a model/ estimate the return
improve the policy
generate samples (i.e. run the policy)

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

expensive

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

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

Value Functions

How do we deal with all these expectations?

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

$$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} \left[E_{\mathbf{a}_1 \sim \pi(\mathbf{a}_1|\mathbf{s}_1)} \left[\underbrace{r(\mathbf{s}_1, \mathbf{a}_1) + E_{\mathbf{s}_2 \sim p(\mathbf{s}_2|\mathbf{s}_1,\mathbf{a}_1)} \left[E_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2|\mathbf{s}_2)} \left[r(\mathbf{s}_2, \mathbf{a}_2) + \ldots|\mathbf{s}_2\right]|\mathbf{s}_1, \mathbf{a}_1\right]}_{\text{what if we knew this part?}} |\mathbf{s}_1\right]\right]$$

we can write this recursively!

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

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

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

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

Definition: Q-function

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

Q is the sum over all time steps of the expected value of the reward, condition on starting at s_t and a_t

Definition: value function

Value function is dependent on state only. It says if you start at s_t and then roll out your policy, what will your be your total expected value

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

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$ so this can be written as the expected value over actions of the q function

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

Using Q-functions and value functions

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

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

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

and it doesn't matter what π is

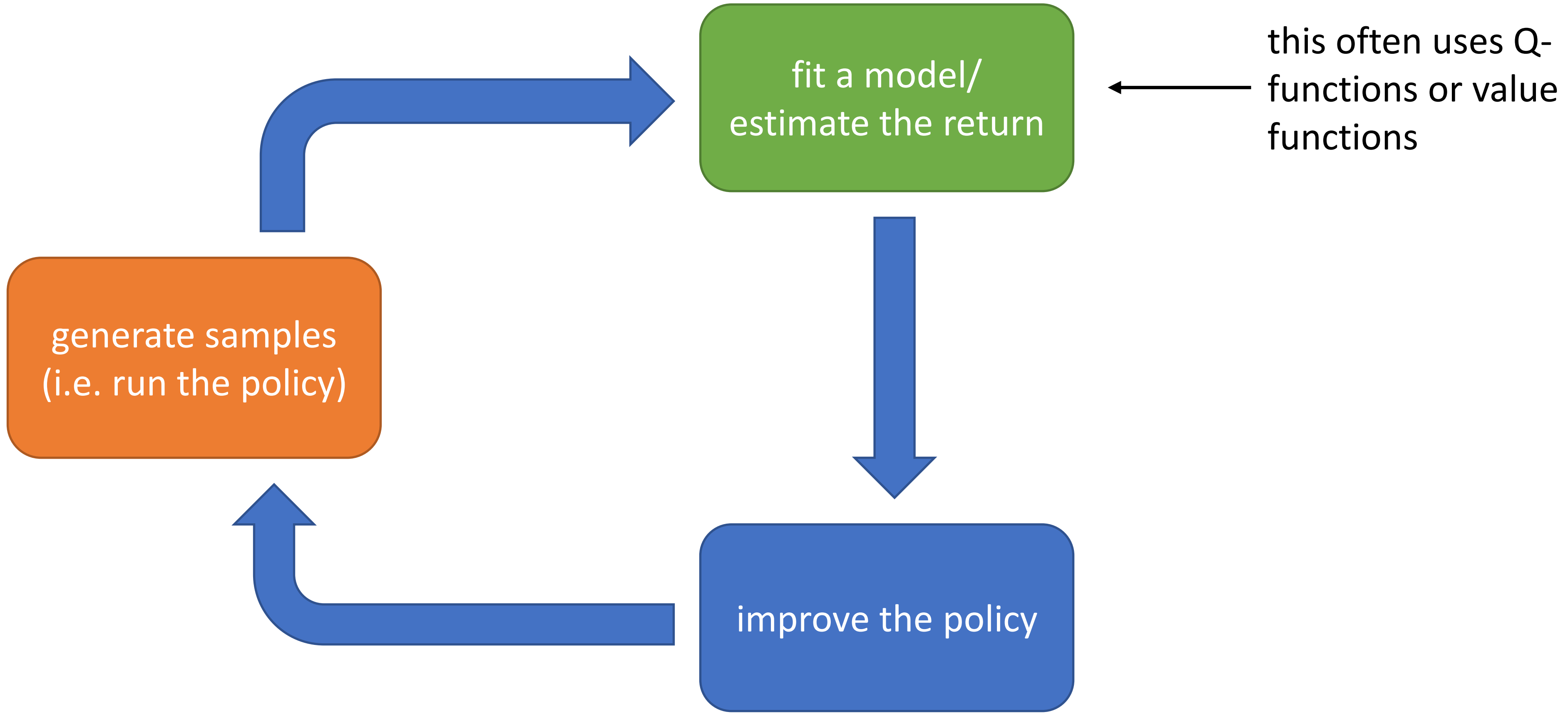
Idea 2: compute gradient to increase probability of good actions $\mathbf{a}$:

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

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

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

The anatomy of a reinforcement learning algorithm



Types of Algorithms

Types of RL algorithms

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

- Policy gradients: directly differentiate the above objective

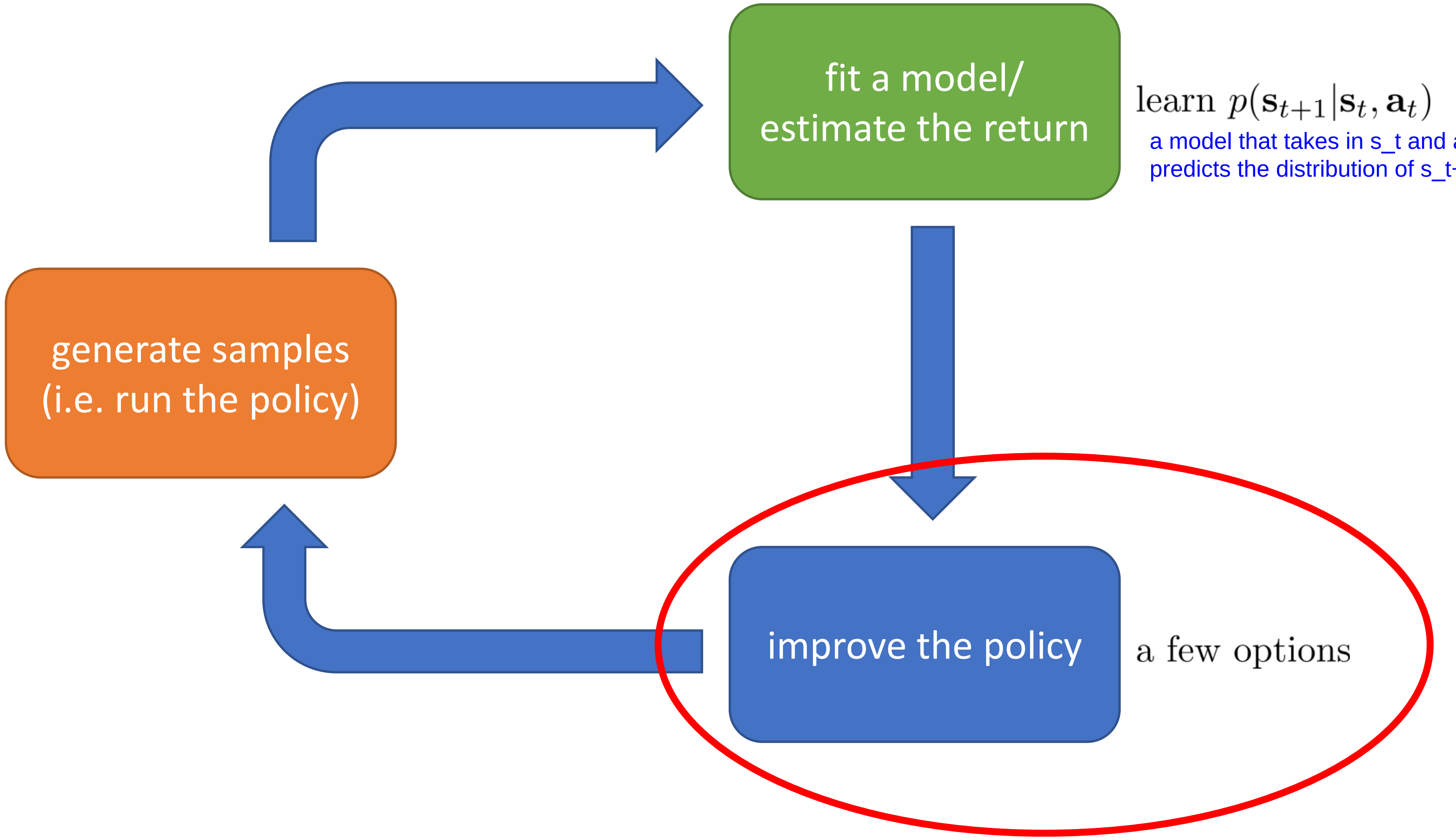
and then perform gradient decent using that derivative

- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)

which are represented by a function approximator like a NN, to improve the policy. Pure value-based functions don't even represent the policy directly, but do so indirectly, through say the argmax of a Q-function

- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then…
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else

Model-based RL algorithms



learn $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

a model that takes in s_t and a_t and predicts the distribution of s_t+1

a few options

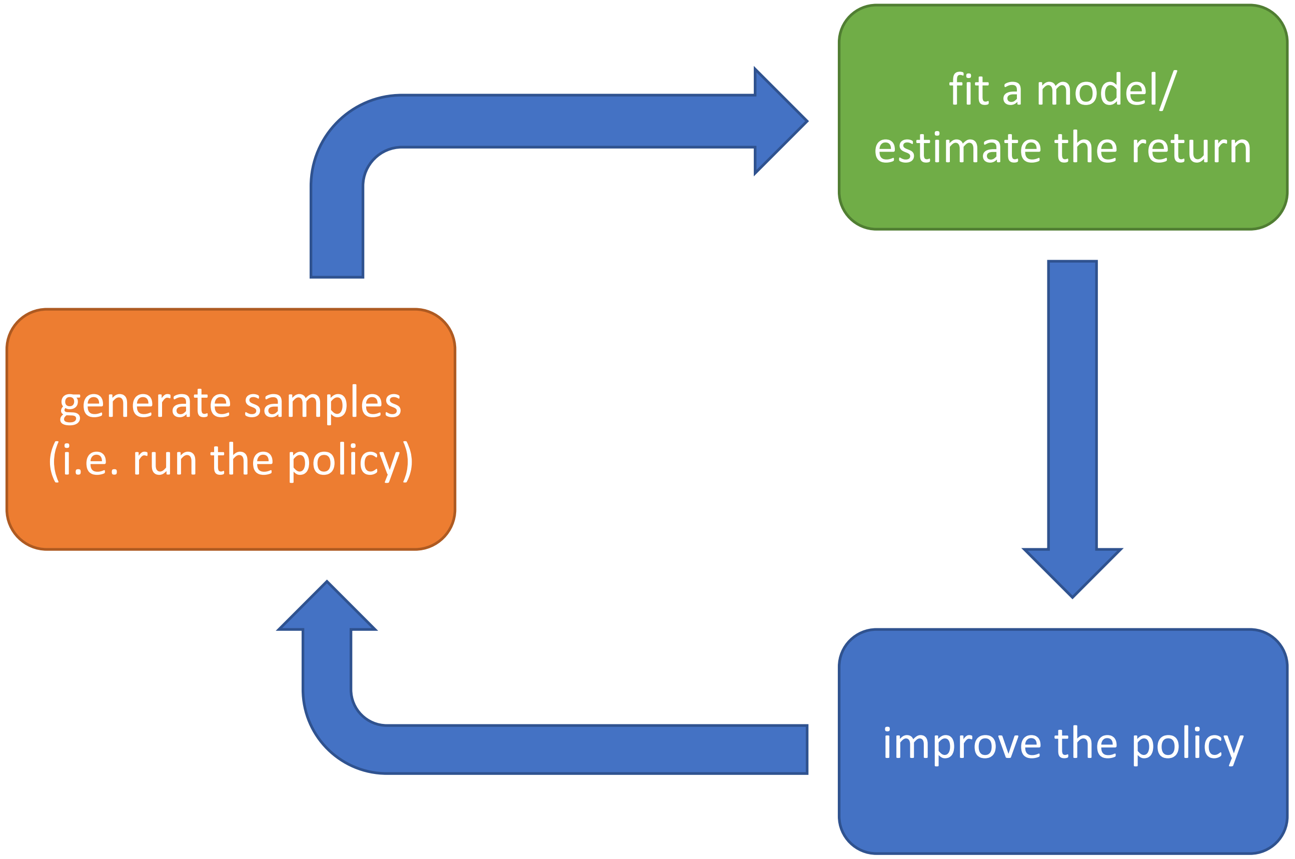
Model-based RL algorithms

improve the policy

a few options

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

Value function based algorithms



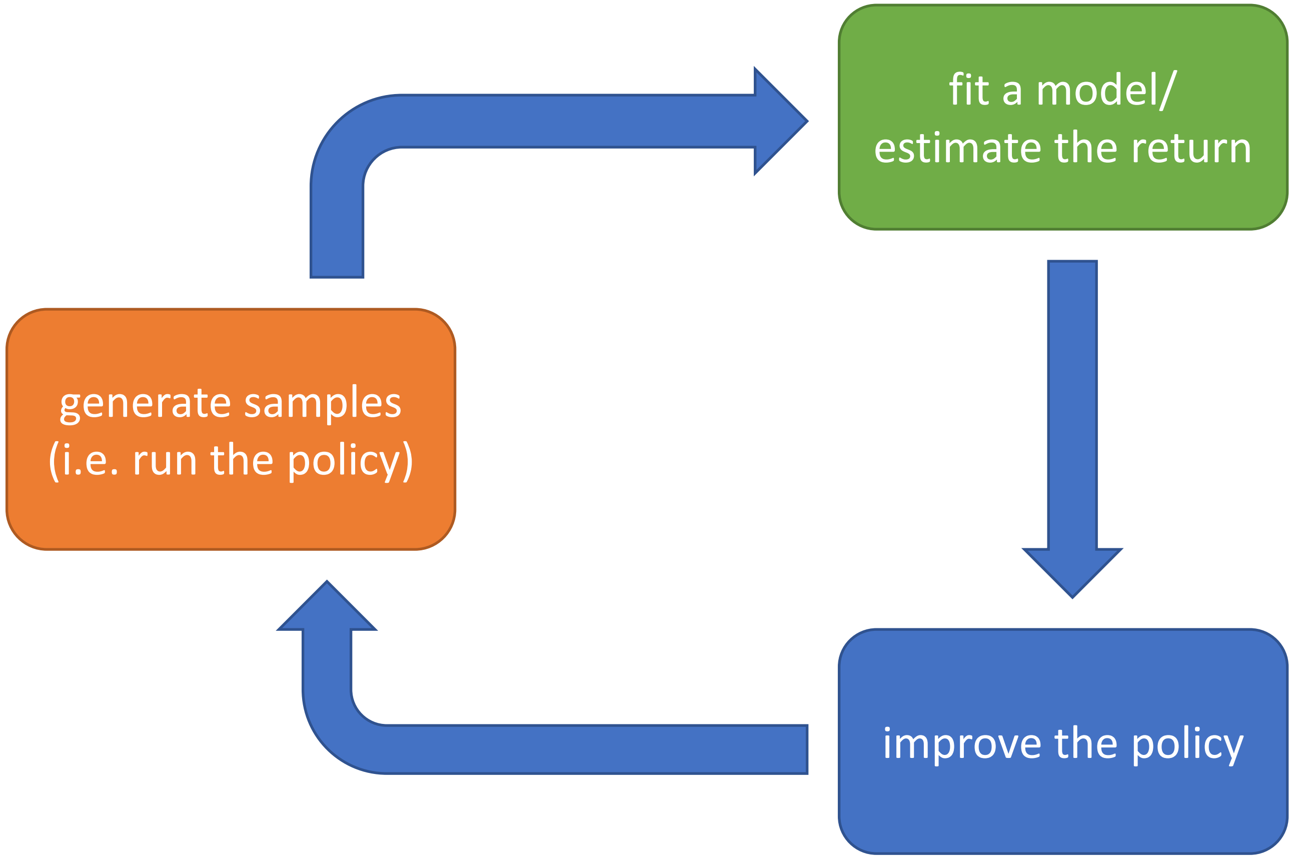
fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

using a NN,input either s or (s,a)
and output a return

set $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

choose the value as the
argmax over Q

Direct policy gradients



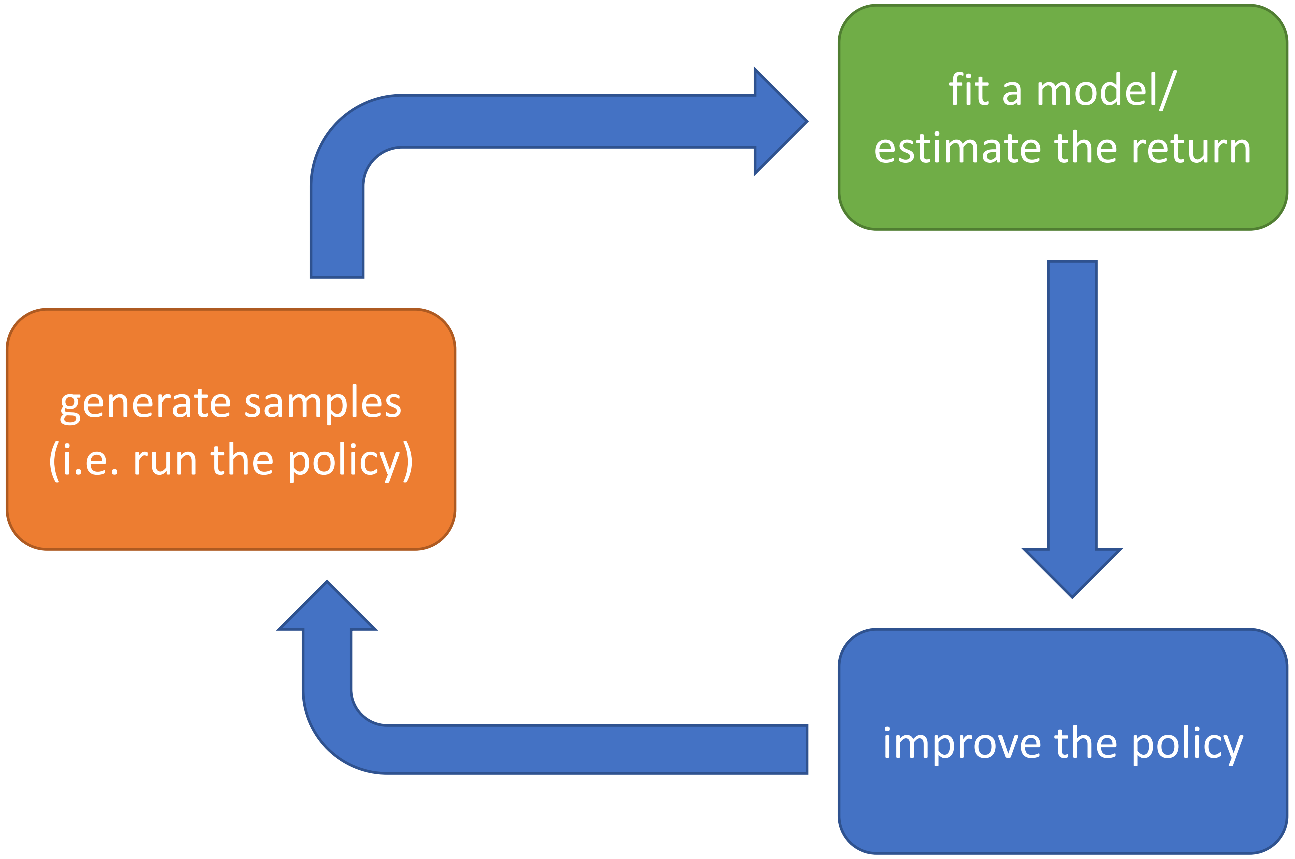
evaluate returns
$R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

simple: compute the rewards obtained by your rollout

$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$

take gradient ascent step on theta, using the gradient of the expected value of the reward

Actor-critic: value functions + policy gradients



fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

just like value-based methods

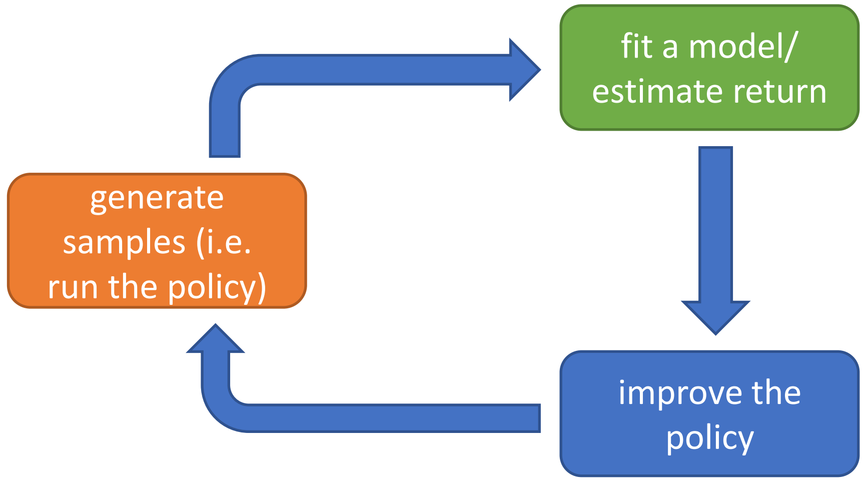
$\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(\mathbf{s}_t, \mathbf{a}_t)]$

just like policy gradient methods
and use the value or Q function
to obtain a more accurate
estimate of the gradient

Tradeoffs Between Algorithms

Why so many RL algorithms?

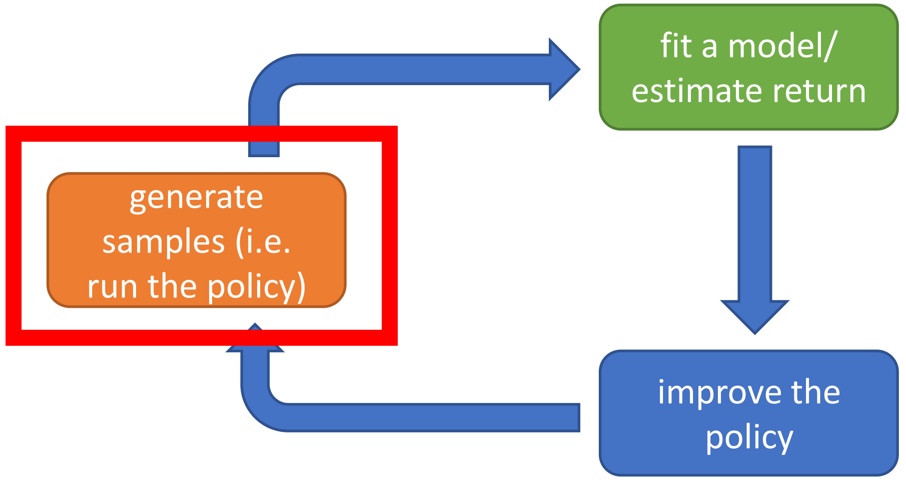
- Different tradeoffs
  - Sample efficiency — how many samples do you need to get a good policy
  - Stability & ease of use — some have more hyperparameters that are difficult to select
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?


generate samples (i.e. run the policy)
fit a model/ estimate return
improve the policy

Comparison: sample efficiency

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

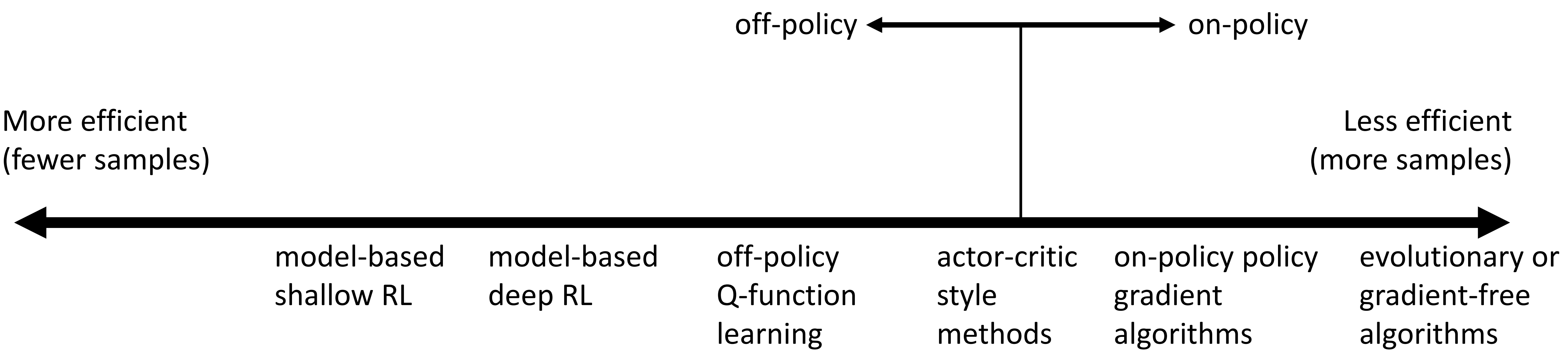
has to throw out all samples every time the policy changes, and then generate new samples. So each time it takes a gradient-step on the policy, it has to generate new samples.


generate samples (i.e. run the policy)
fit a model/ estimate return
improve the policy

$$\theta \leftarrow \theta + \alpha \nabla_\theta E[\textstyle\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$$

just one gradient step

Comparison: sample efficiency


off-policy
on-policy
More efficient (fewer samples)
Less efficient (more samples)
model-based shallow RL
model-based deep RL
off-policy Q-function learning
actor-critic style methods
on-policy policy gradient algorithms
evolutionary or gradient-free algorithms

Why would we use a *less* efficient algorithm?

Wall clock time is not the same as efficiency!

generating new samples could be very very fast!!

Comparison: stability and ease of use

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

Why is any of this even a question???

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

Unlike supervised learning, convergent algorithms in RL is a rare luxury! Many algos we use in practice are not guaranteed to converge! RL often is not pure gradient decent, many are fixed-point algos that are only guaranteed to converge under specific assumptions that often don't hold in practice

instead, it's optimized to be an accurate model. There's no guarantee that getting a better model will result in a better reward value

technically, it's gradient ascent

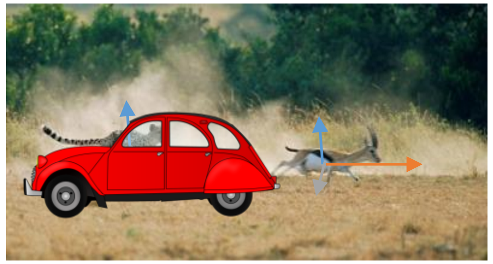
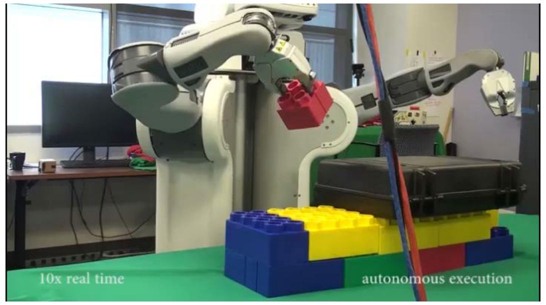
Comparison: stability and ease of use

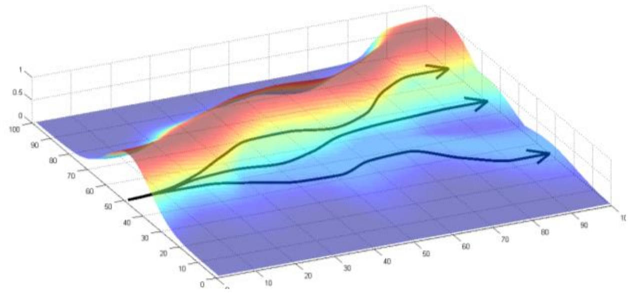
- Value function fitting
  - At best, minimizes error of fit (“Bellman error”)
    - Not the same as expected reward
  - At worst, doesn’t optimize anything
    - Many popular deep RL value fitting algorithms are not guaranteed to converge to *anything* in the nonlinear case
- Model-based RL
  - Model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

Comparison: assumptions

there's a trial, and then it resets, then there's another trial. This ability to "reset" is assumed.

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



Examples of Algorithms

Examples of specific algorithms

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

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

Example 1: Atari games with Q-functions

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

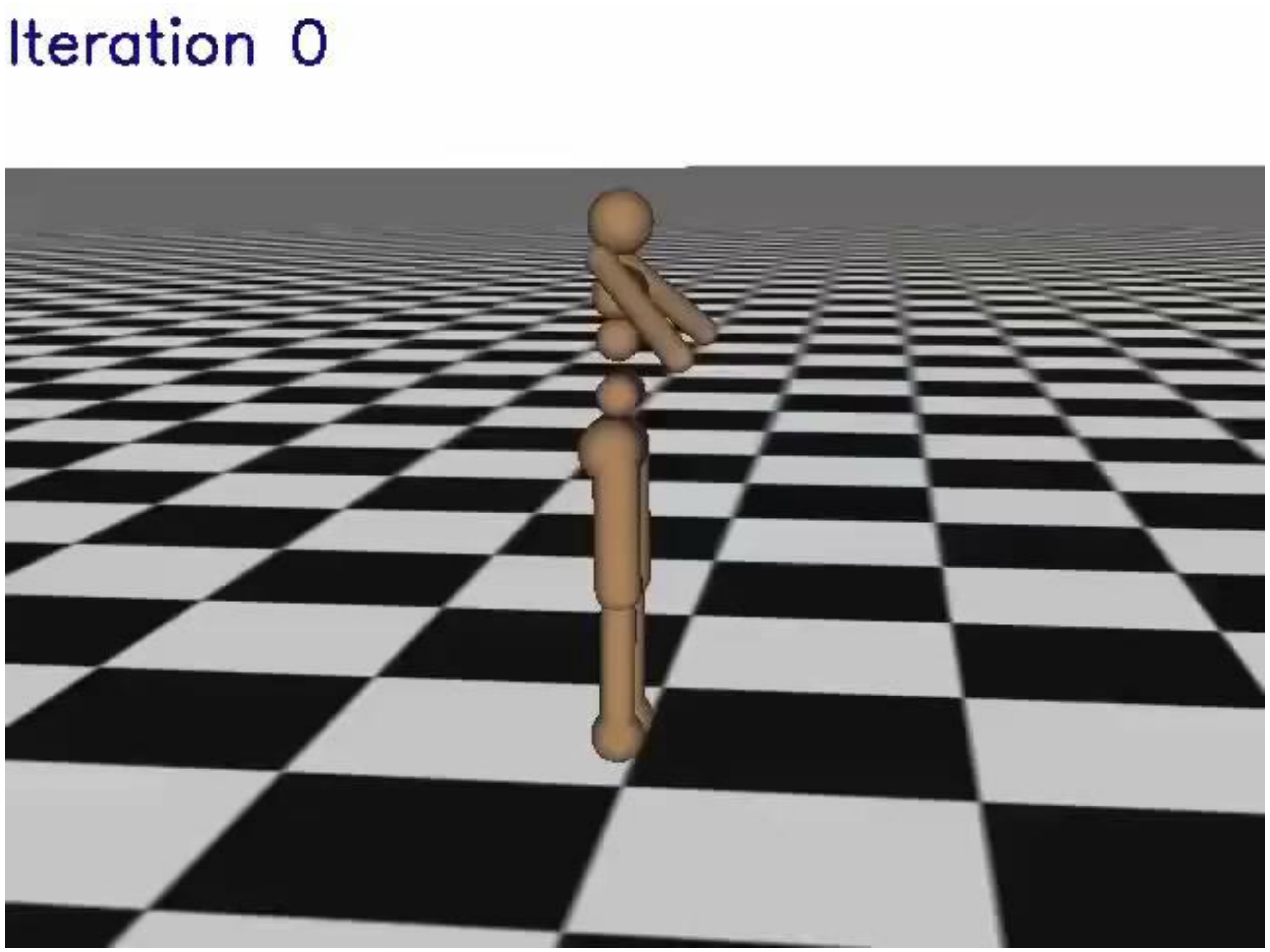
Example 2: robots and model-based RL

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

Various Experiments
Including the policy input

Example 3: walking with policy gradients

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



Example 4: robotic grasping with Q-functions

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