Introduction to Reinforcement Learning

CS 285: Deep Reinforcement Learning, Decision Making, and Control

Sergey Levine

Class Notes

1. Homework 1 is due next Monday!
2. Remember to start forming final project groups
    - Final project proposal due Sep 25
    - Final project ideas document coming soon!

Today's Lecture

1. Definition of a Markov decision process

MDP is the fundamental mathematical building blocks for the RL algos we cover in this course

2. Definition of reinforcement learning problem
3. Anatomy of a RL algorithm
4. Brief overview of RL algorithm types

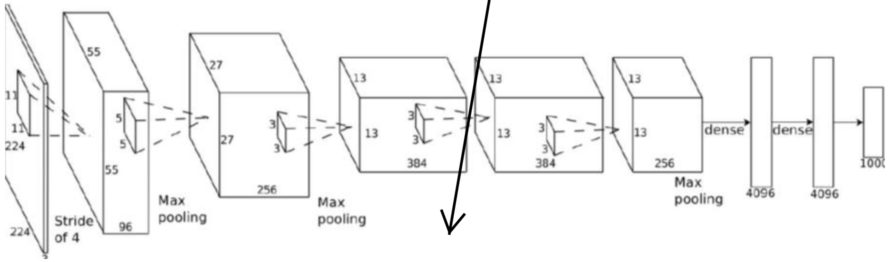
- Goals:
  - Understand definitions & notation
  - Understand the underlying reinforcement learning objective
  - Get summary of possible algorithms

Definitions

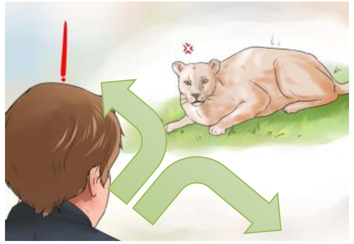
Terminology & notation

this is our model. This is what we're after! Our goal is to find theta

$\mathbf{o}_t$



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

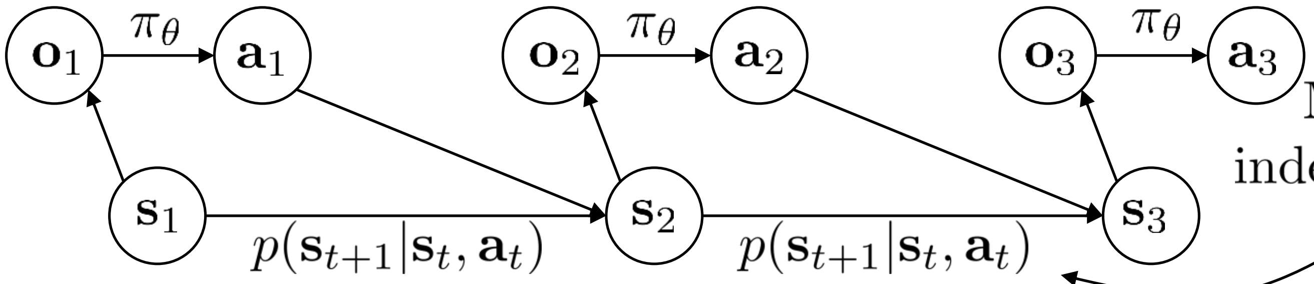


$\mathbf{a}_t$

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

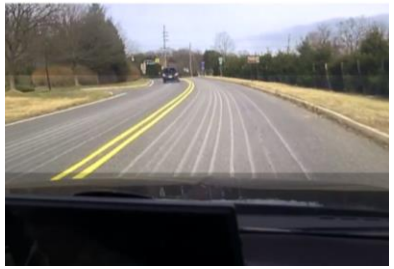
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy (partially observed)

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

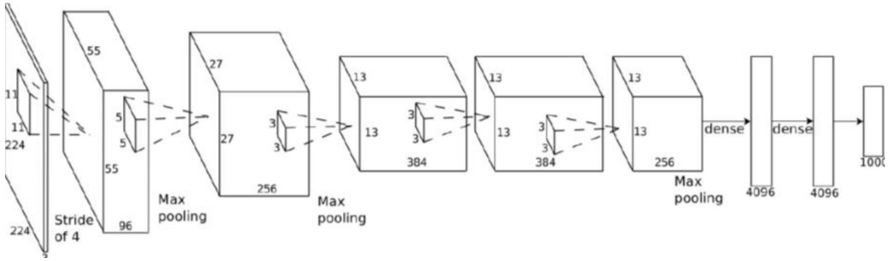


Markov property
independent of $\mathbf{s}_{t-1}$

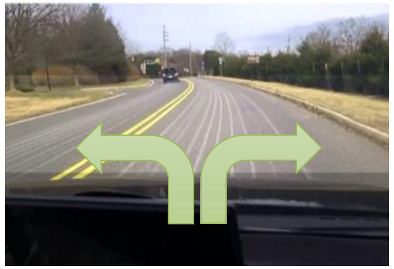
Imitation Learning



$\mathbf{o}_t$



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



$\mathbf{a}_t$

$\mathbf{o}_t$

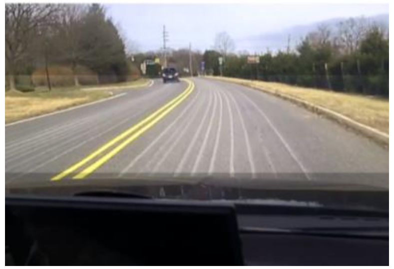
$\mathbf{a}_t$

→ training data → supervised learning → $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

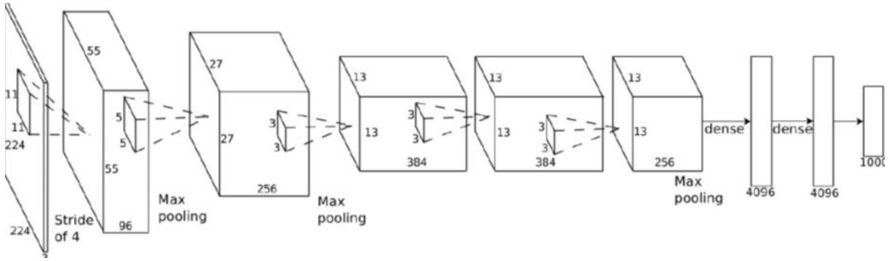
Images: Bojarski et al. '16, NVIDIA

Reward functions

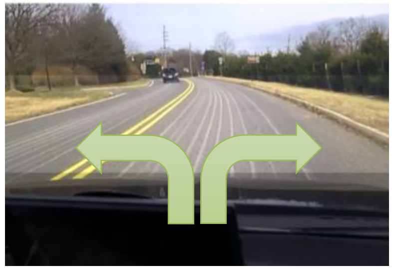
we use reward function to determine which actions are
good and bad



$\mathbf{o}_t$



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



$\mathbf{a}_t$

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

indicates the probability of being in any state at time t

if you have n states, this vector will be size n

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

T is an nxn matrix, where entry i,j is the probability of moving to state i given you're currently in state j

so we can use this to do linear algebra

$$\mathbf{s}_1 \xrightarrow{p(\mathbf{s}_{t+1}|\mathbf{s}_t)} \mathbf{s}_2 \xrightarrow{p(\mathbf{s}_{t+1}|\mathbf{s}_t)} \mathbf{s}_3$$

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

Notice we don't have actions a yet. So we cant formulate RL with this because there's no agency!

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!)

typo. Should be mu t+1

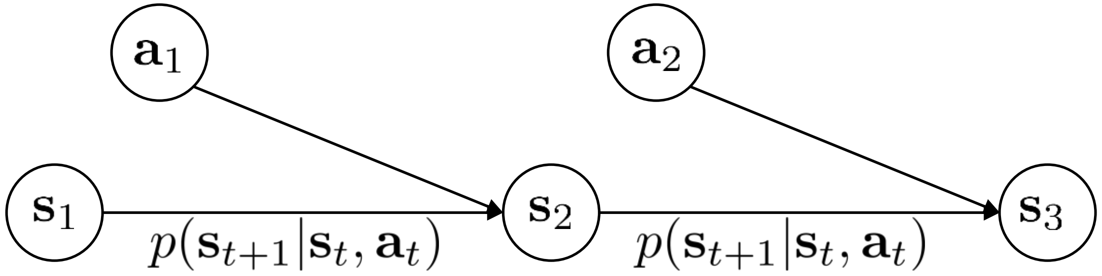
let $\mu_{t,j} = p(s_t = j)$

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

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

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

probability of taking an action of timestep k



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

$r(s_t, a_t)$ – 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)

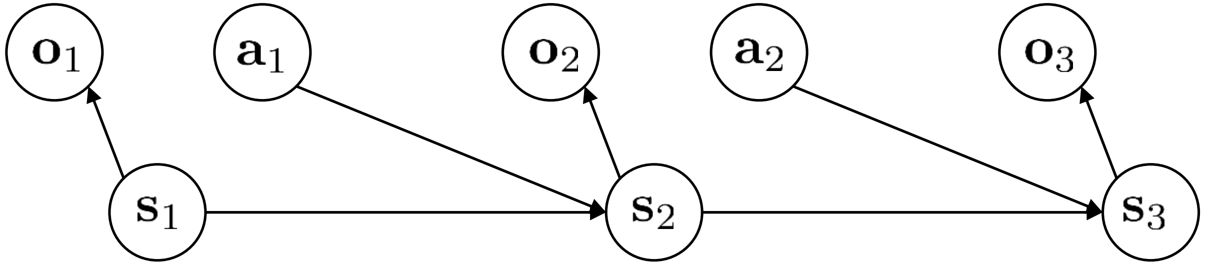
$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

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

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

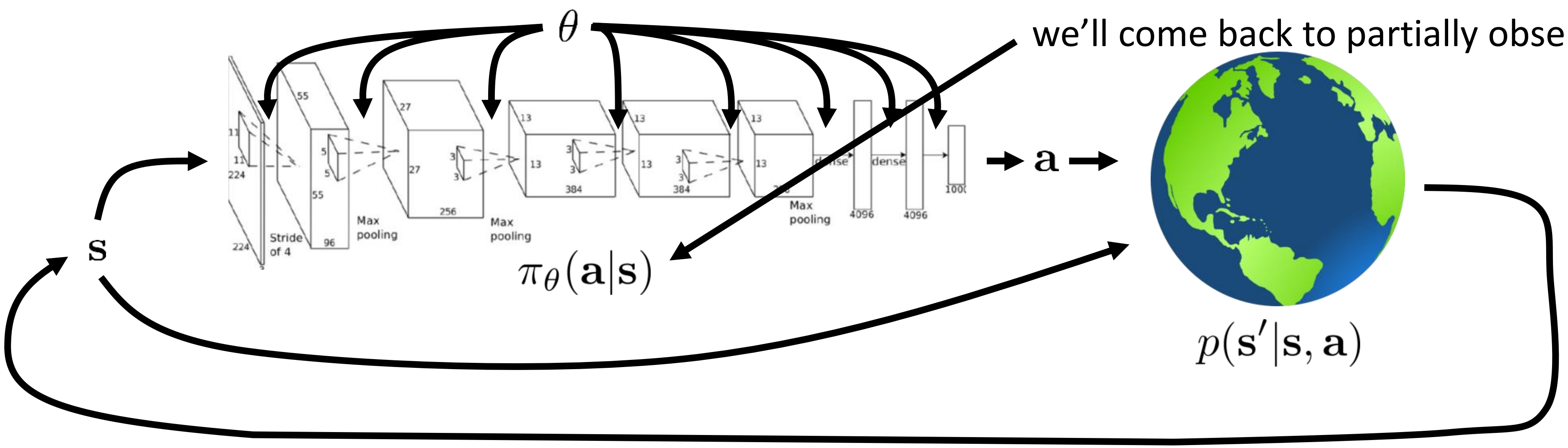
$\mathcal{E}$ – emission probability $p(o_t|s_t)$

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



The goal of reinforcement learning

Goal: Find a policy! We need to learn theta

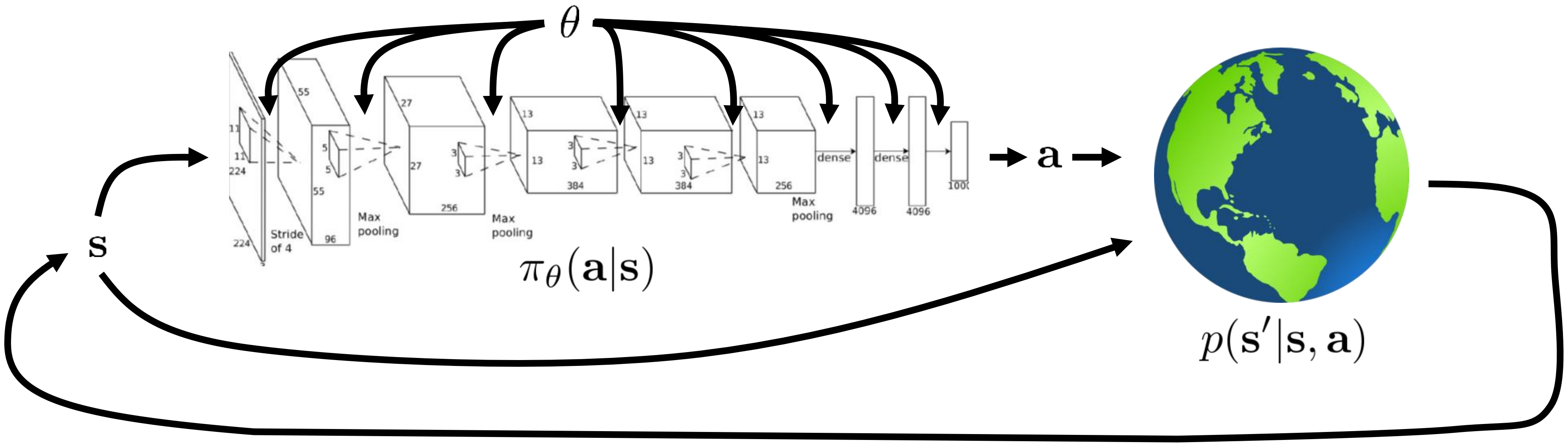


$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

Our objective is to get the best theta by maximizing reward over all time steps.

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

The goal of reinforcement learning

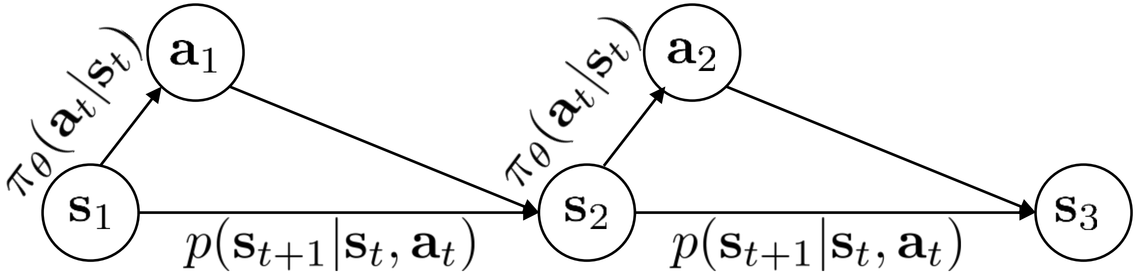


$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on }(\mathbf{s},\mathbf{a})}$$

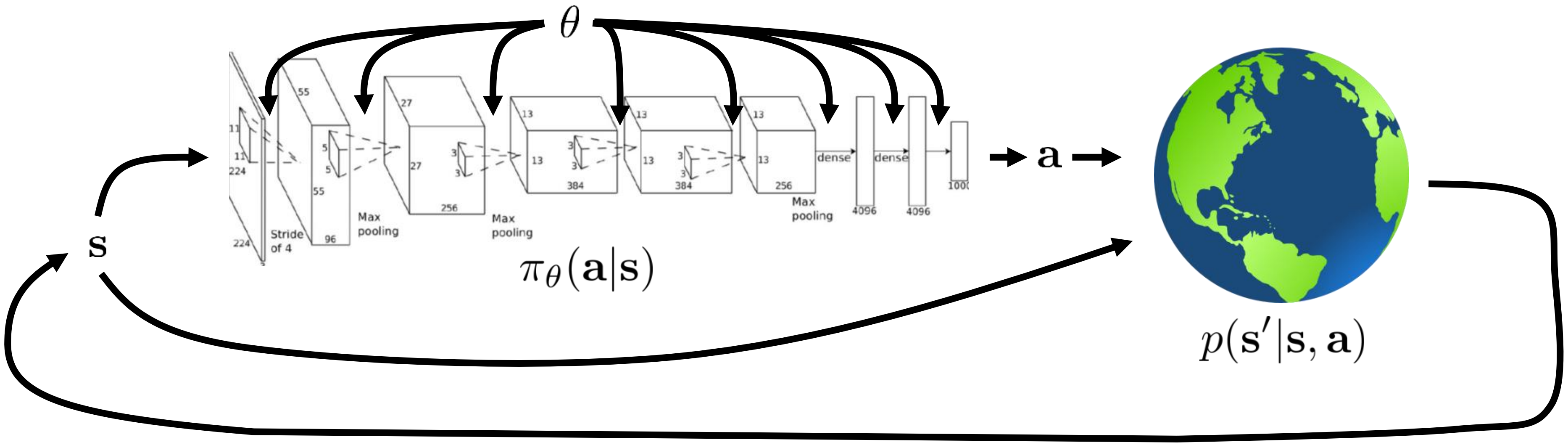
Three factors multiplied together
- Here we assume a finite horizon

- You exist for T timesteps
- Multiply initial state probability times probability of the action given the state at a given timestep, times the probability of transitioning to the next state s' given the current state and action

short hand...meaning probability of a certain trajectory

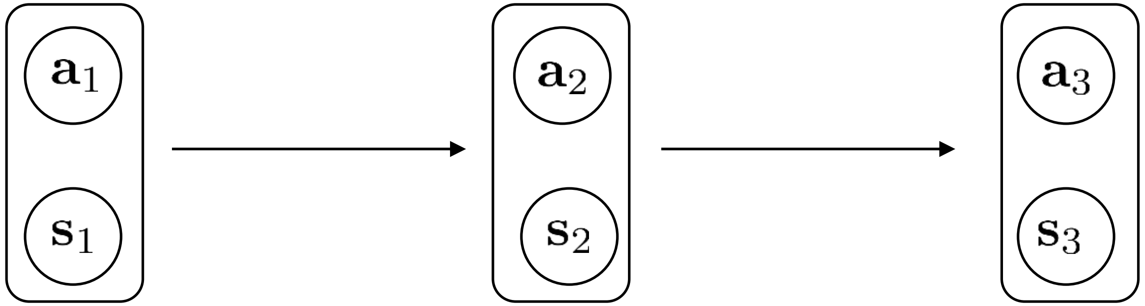


The goal of reinforcement learning



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

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



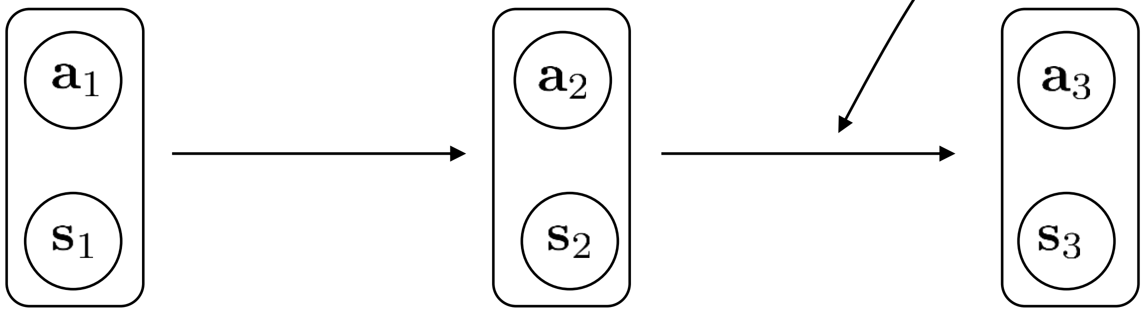
Finite horizon case: state-action marginal

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

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

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

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



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?

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

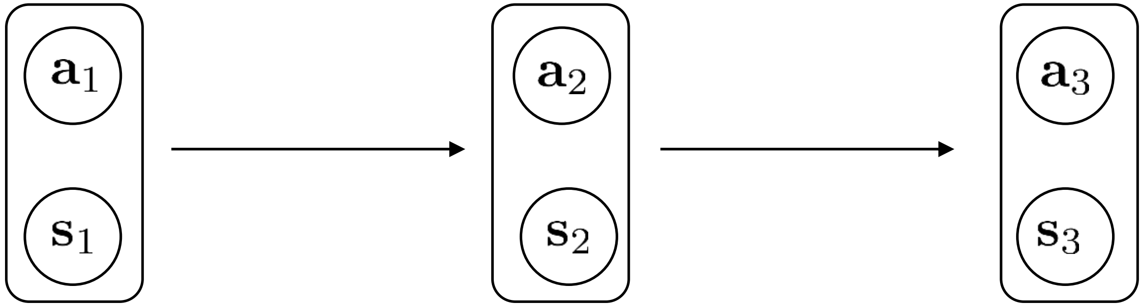
stationary = the same before and after transition

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

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?

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

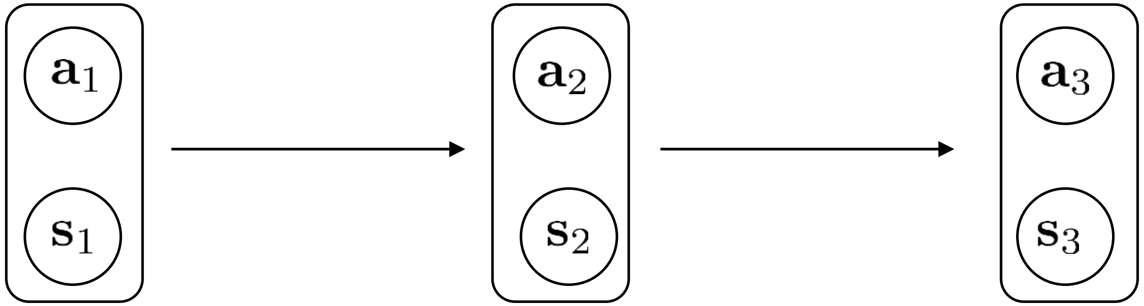
stationary = the same before and after transition

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

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

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

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!

Algorithms

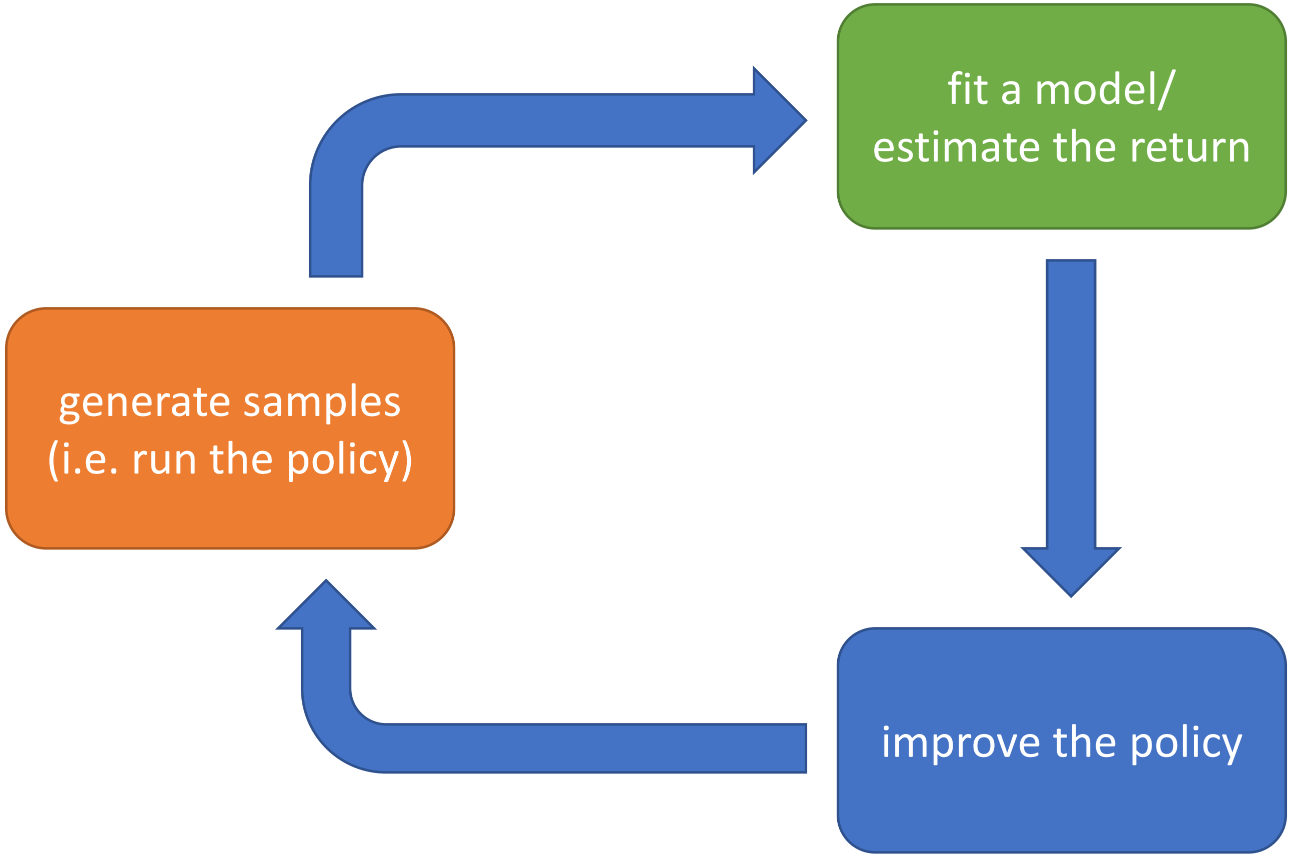
The anatomy of a reinforcement learning algorithm

Every algorithm has 3 parts!

Where RL differentiates with optimal control is that RL actually interacts with the world to gather data

some sort of evaluation. Do something that doesn't change your policy, but helps us evaluate our policy.

might include...
- Fitting model
- Estimate return
- Evaluate reward



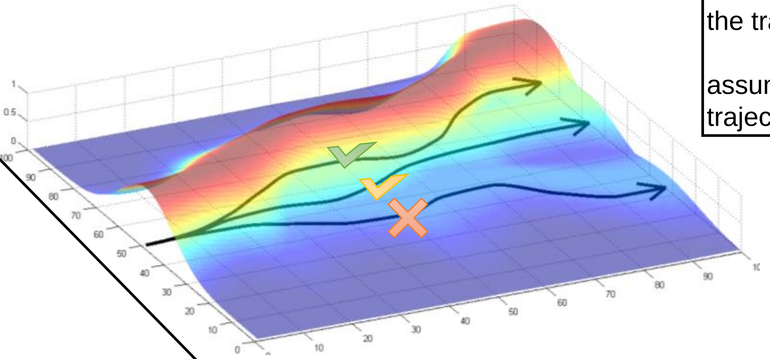
change policy using estimates from green box, to make the policy better

A simple example

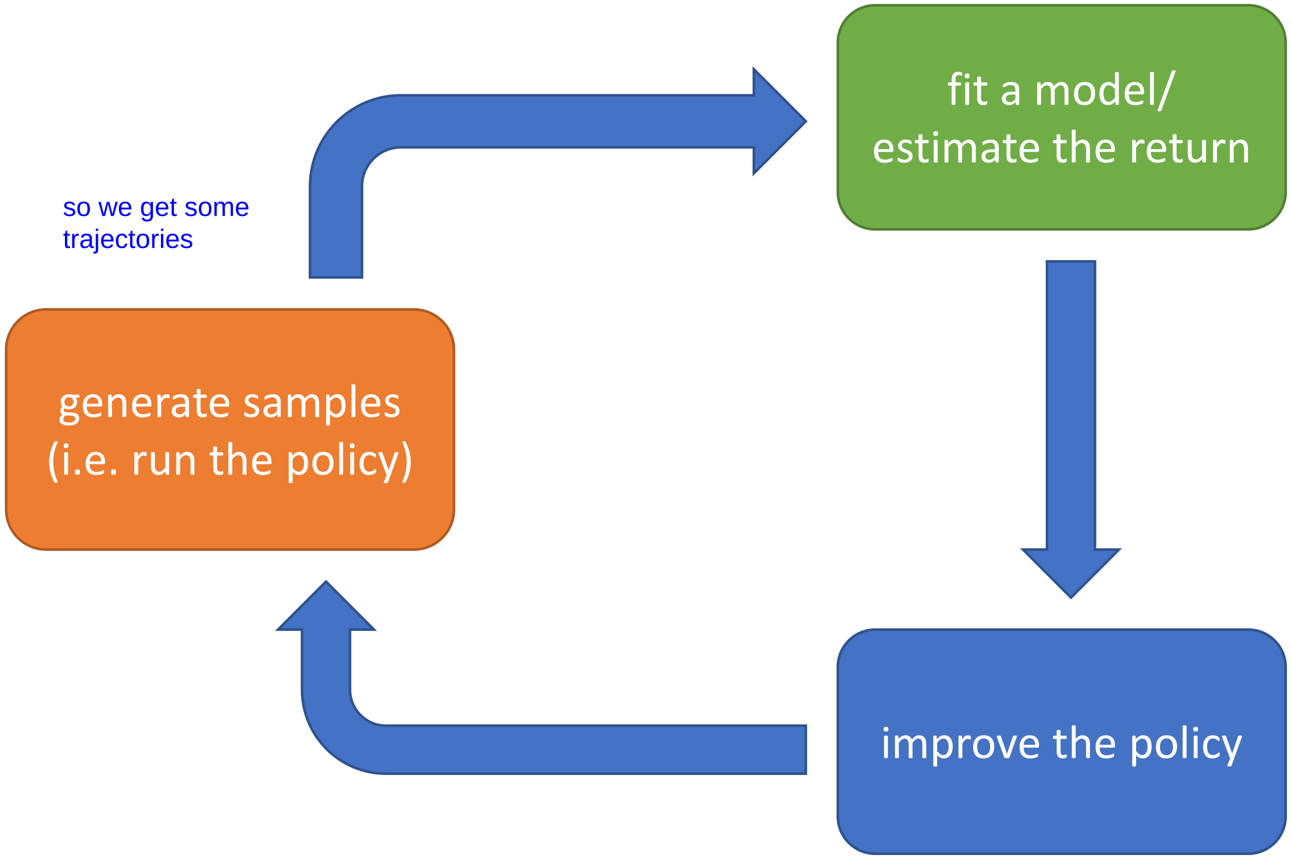
J(theta) is the expected total rewards for our policy

where i is the index of the trajectory

assume we ran N trajectories



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



so we get some trajectories

take each trajectory and evaluate the reward for each step along that trajectory and sum it. So we get the total reward for that trajectory.

Then we take the trajectories that seem good and increase its probability, and we take the trajectories that seem bad and decrease its probability

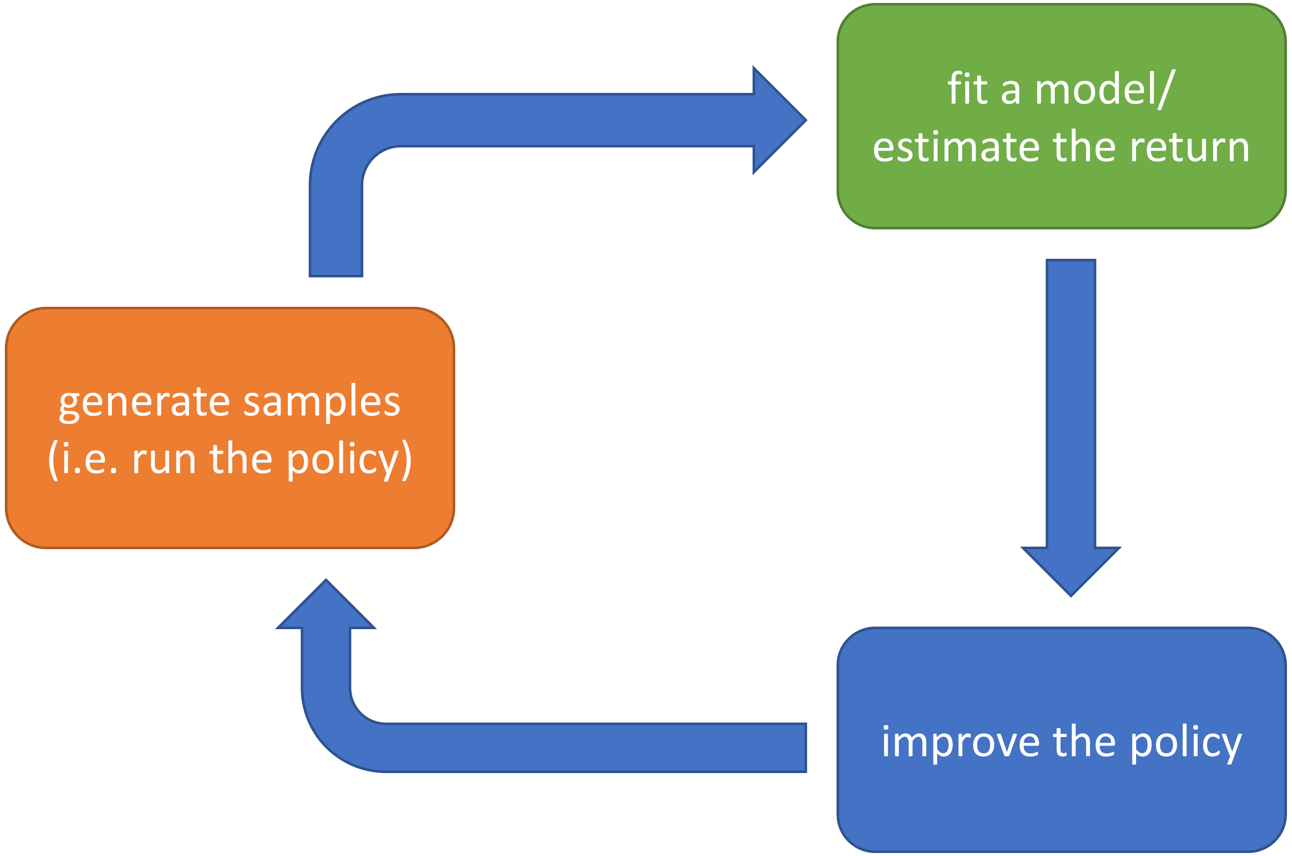
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

improvement done by gradient descent

figure out the derivative of the total reward along each trajectory and increase the ones where the derivative is positive, and decrease the ones where the derivatives is negative

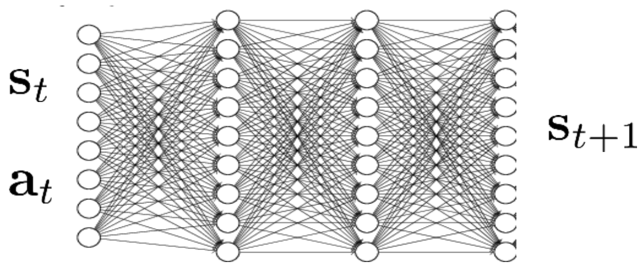
Another example: RL by backprop

for example if
you're doing
model-based RL



using supervised learning, fit a model to predict the next state given the current state and action

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?

it depends on the problem

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

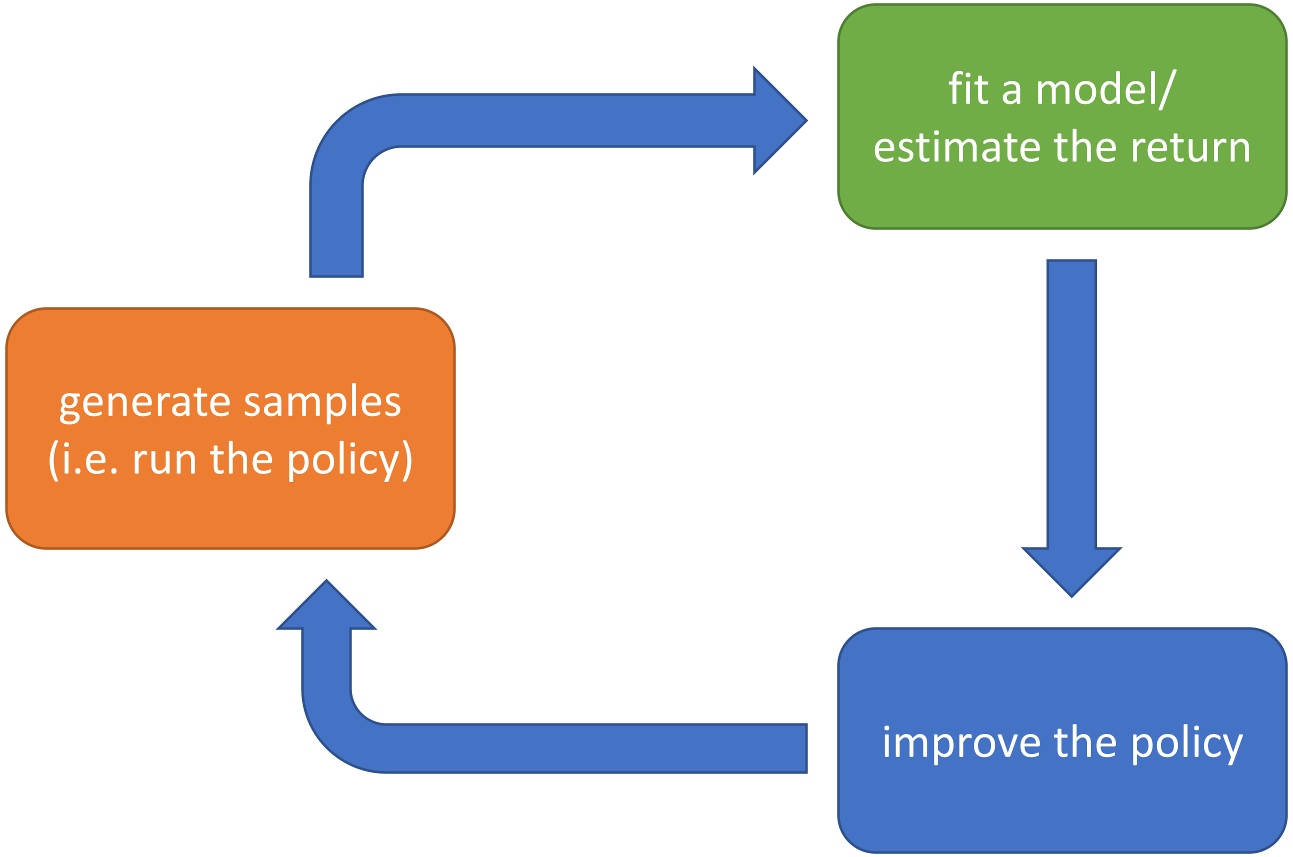
trivial, fast

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

expensive

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

MuJoCo simulator:
up to 10000x real time

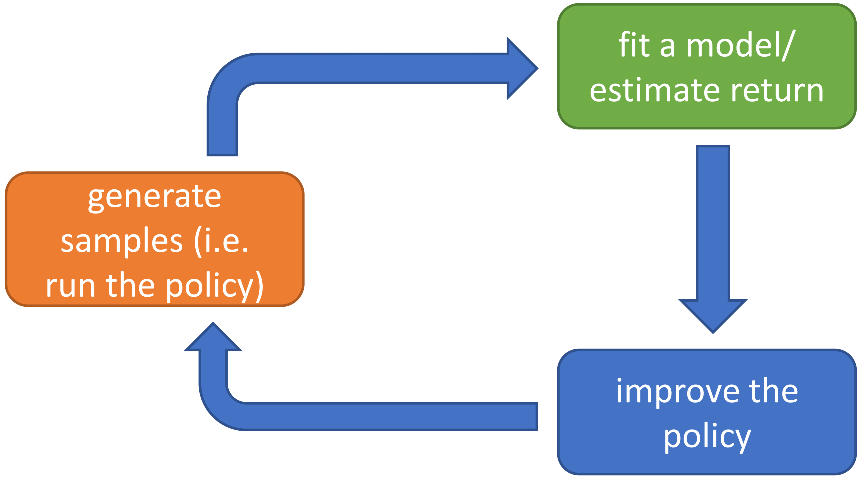


$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

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

Review

- Definitions
  - Markov chain
  - Markov decision process
- RL objective
  - Expected reward
  - How to evaluate expected reward?
- Structure of RL algorithms
  - Sample generation
  - Fitting a model/estimating return
  - Policy Improvement



Break

How do we deal with all these expectations?

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

$$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}$$

what if we knew this part?

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

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

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

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

typically we need to approximate this because
it becomes computationally intractable

Definition: value function

it's just a version of the Q-function except it doesn't take the
action as an input

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

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

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

the value function is just the average of the q-value. It's the
expected q value at a given state.

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

you could just take a new policy by changing the probability of taking that action given the state to 1

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

and it doesn't matter what π is

V(s) is just the average of Q(s)

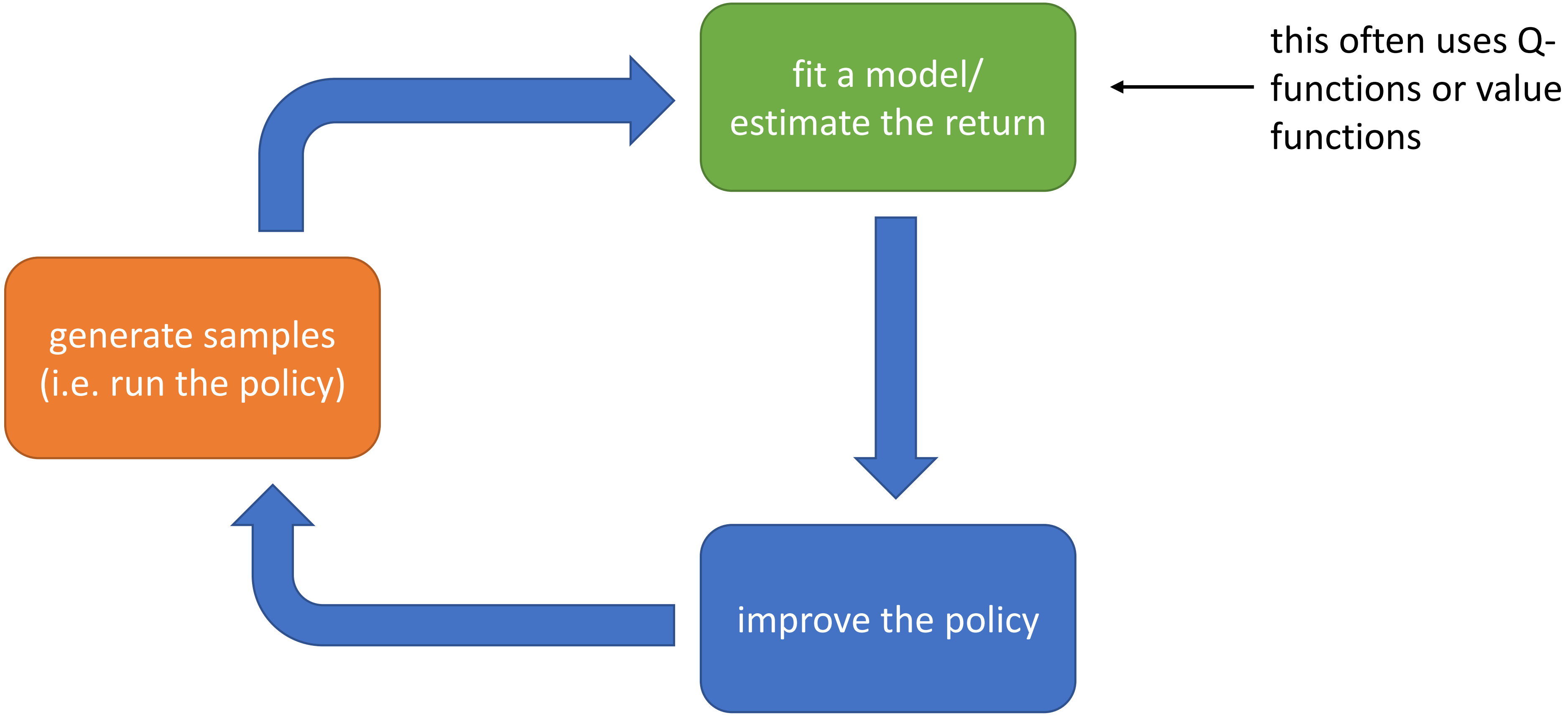
Idea 2: compute gradient to increase probability of good actions $\mathbf{a}$:

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

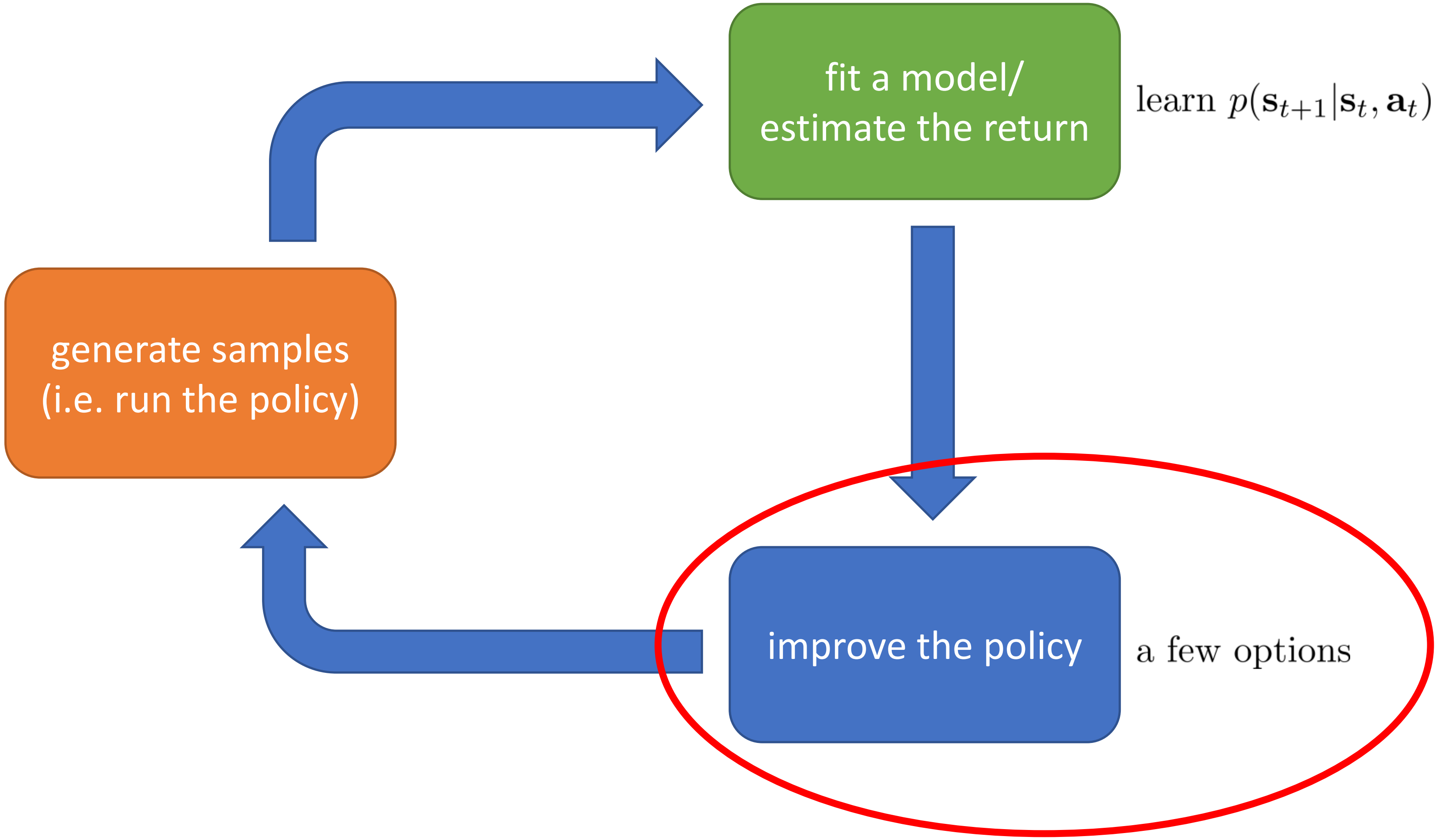


Types of RL algorithms

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

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy) doesn't keep track of pi_theta
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy does the combination of these two things
- Model-based RL: estimate the transition model, and then…
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else

Model-based RL algorithms



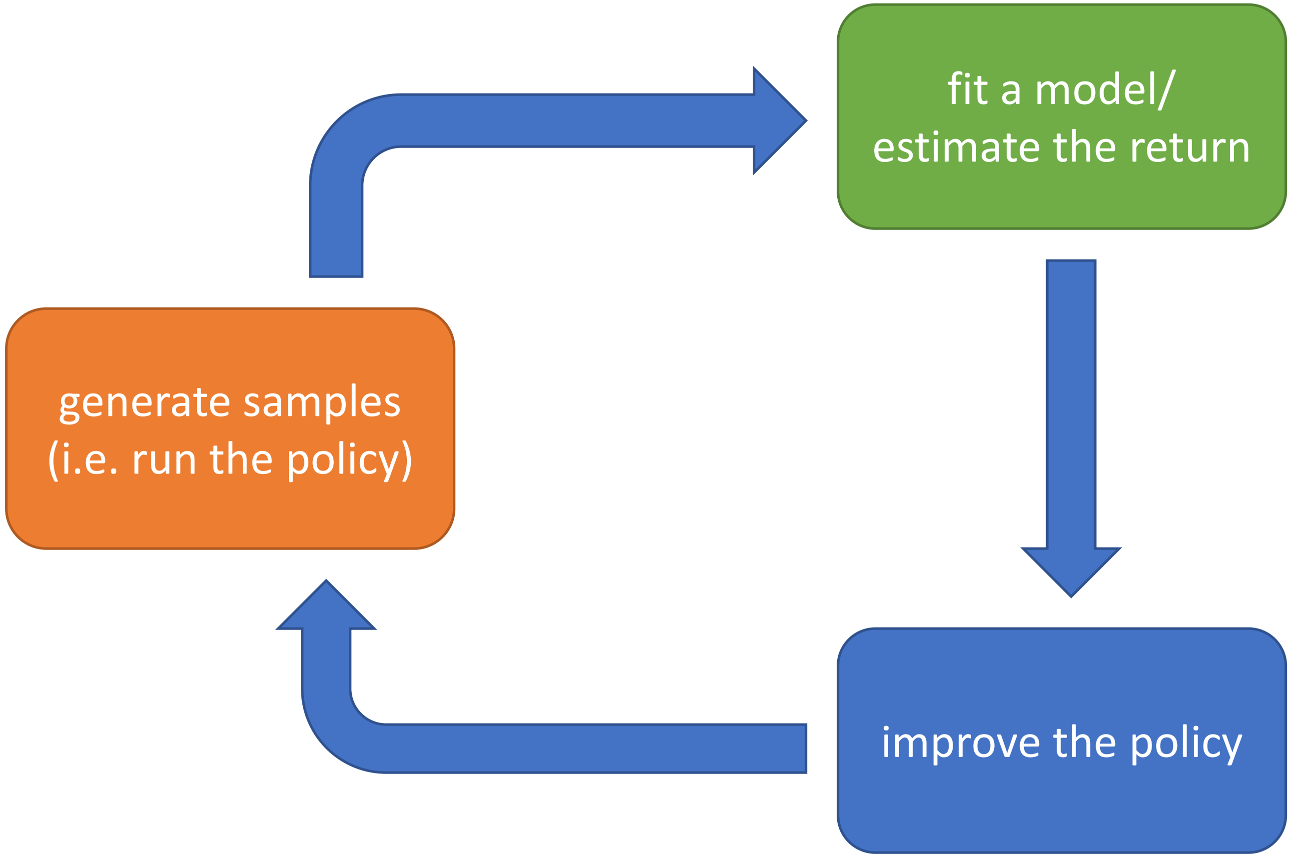
Model-based RL algorithms

improve the policy

a few options

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

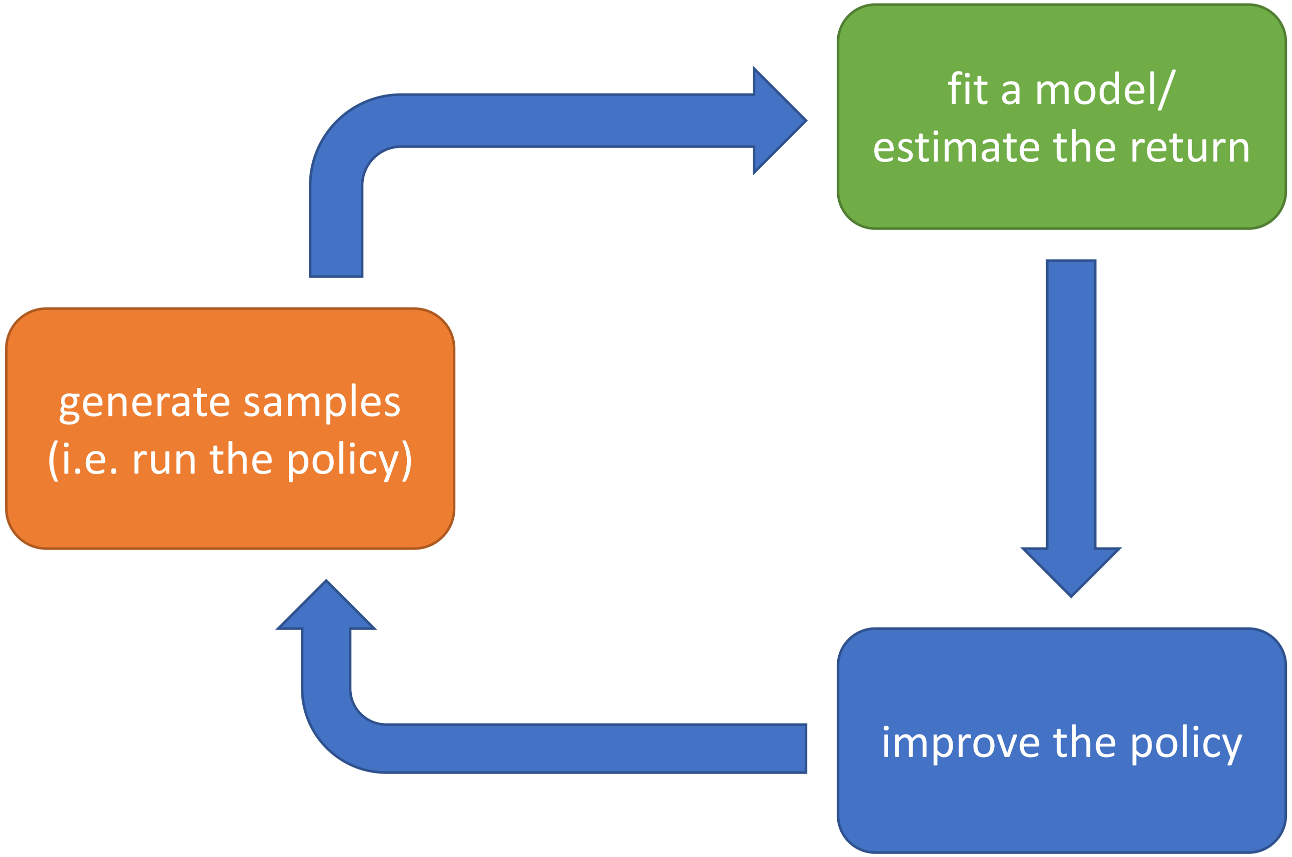
Value function based algorithms



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

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

Direct policy gradients



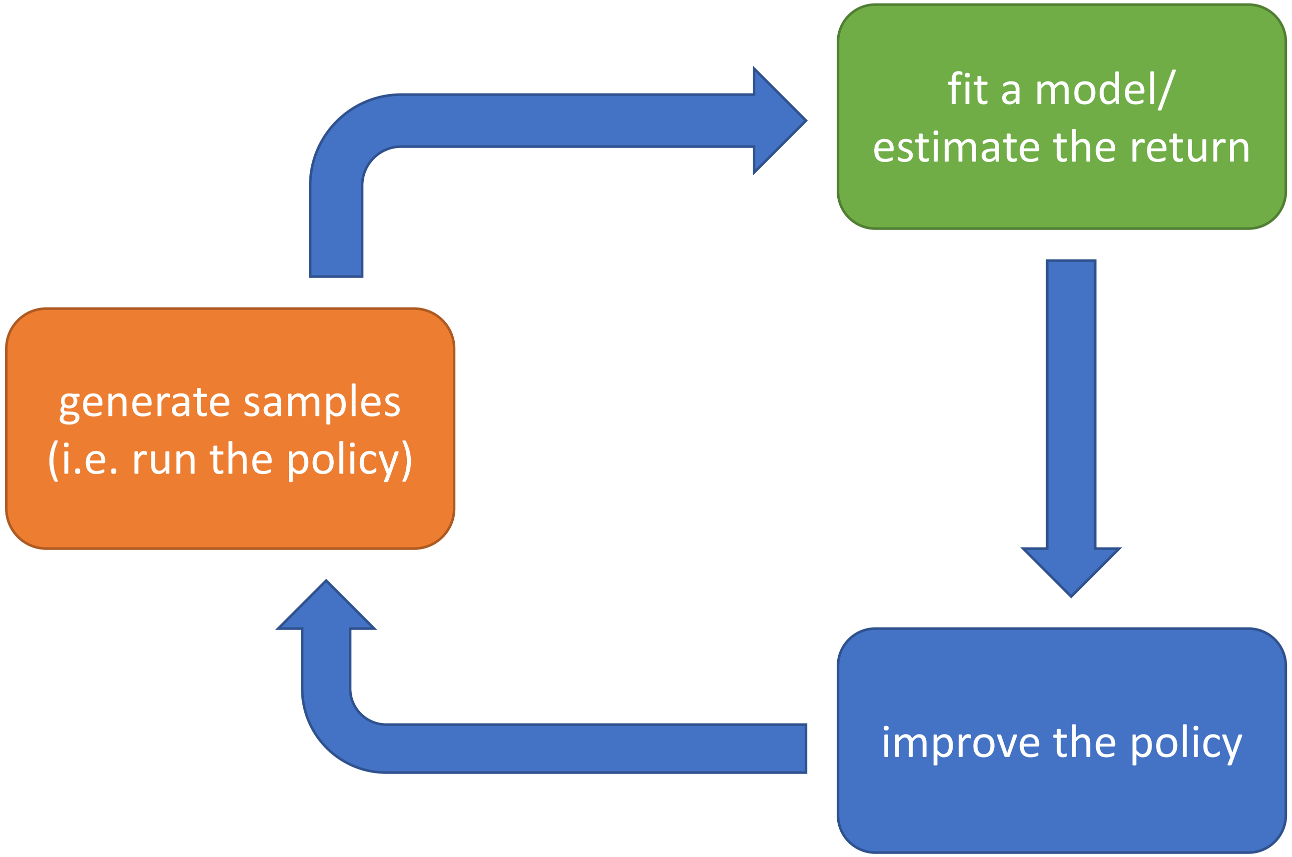
evaluate returns
$R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

trivial

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

gradient ascent

Actor-critic: value functions + policy gradients



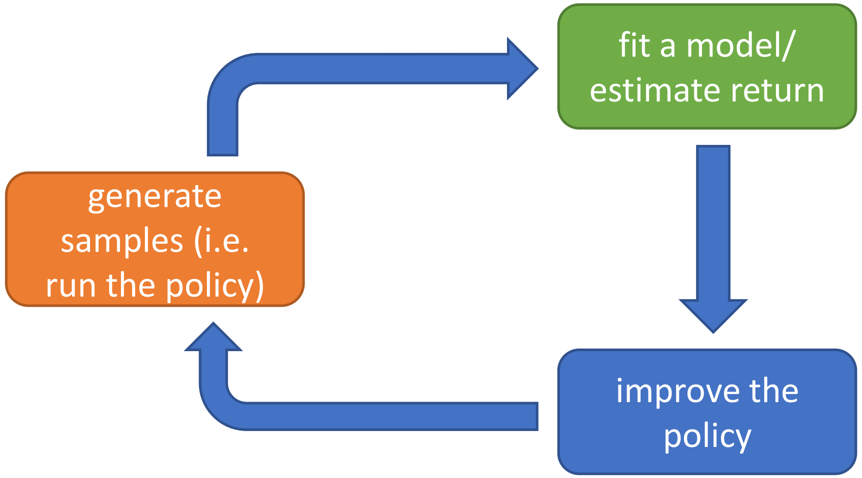
fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

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

Tradeoffs

Why so many RL algorithms?

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



Comparison: sample efficiency

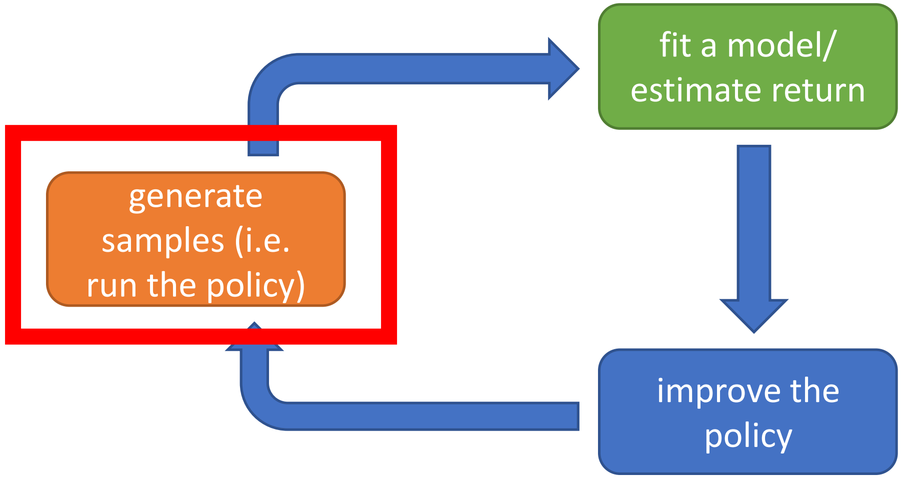
a sample means, take your policy, run it, and see what it does
could be in simulator or real world

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

as soon as the policy changes a little bit, you need to generate new samples

One of the most important factors in determining the efficiency of a RL algorithm is whether it's on or off policy.

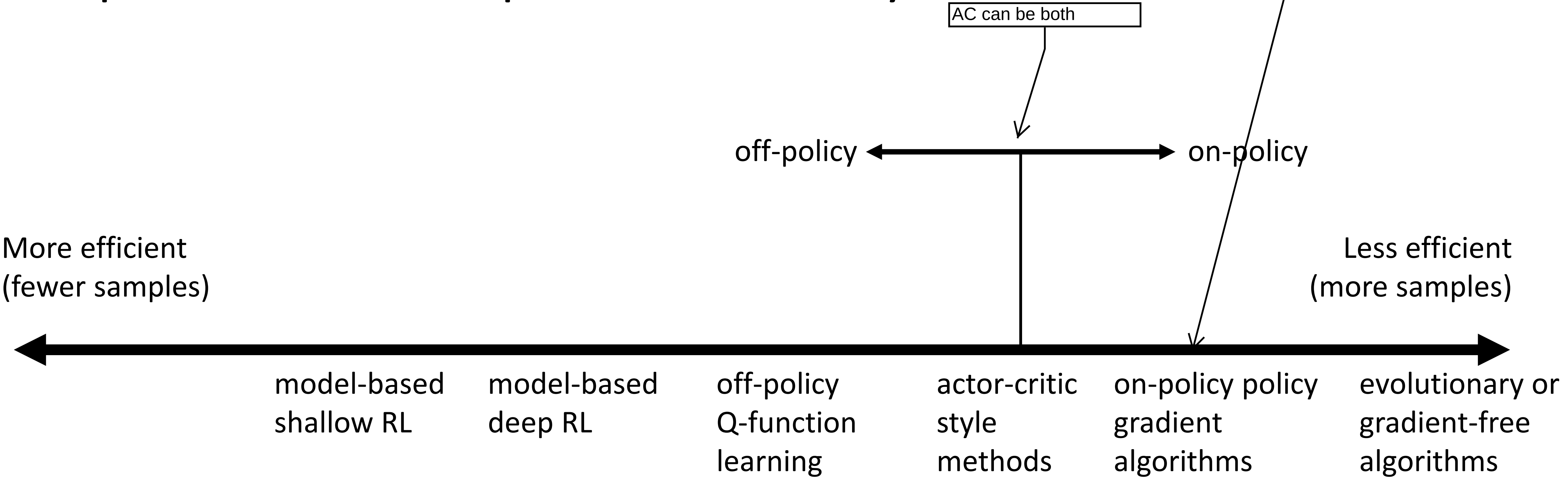
- Whether the algos can use data from other policies



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

just one gradient step

Comparison: sample efficiency



Why would we use a *less* efficient algorithm?

Wall clock time is not the same as efficiency!

Comparison: stability and ease of use

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

Why is any of this even a question???

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

Comparison: stability and ease of use

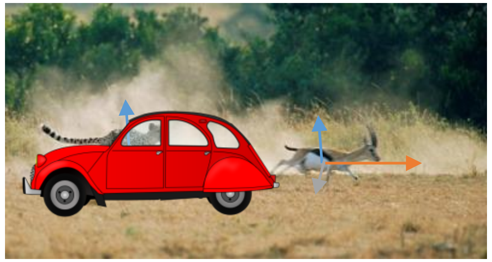
- Value function fitting
  - At best, minimizes error of fit (“Bellman error”)
    - Not the same as expected reward
  - At worst, doesn’t optimize anything
    - Many popular deep RL value fitting algorithms are not guaranteed to converge to *anything* in the nonlinear case
- Model-based RL
  - Model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

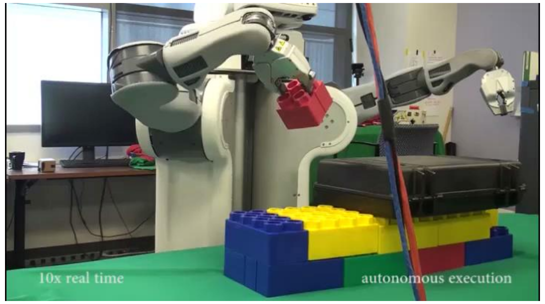
Comparison: assumptions

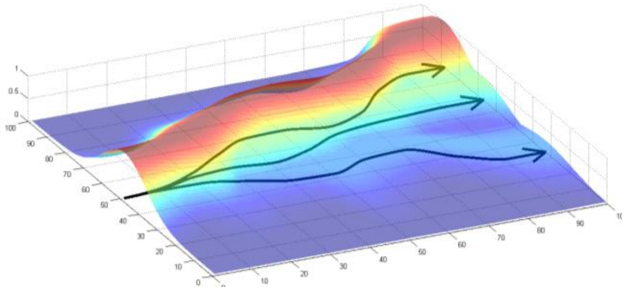
common assumptions that some algorithms will make

learning is separated into specific trials

- Common assumption #1: full observability
  - Generally assumed by value function fitting methods
    exploits Markov Property, so we don't include past
  - 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 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) DDPG also
- 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

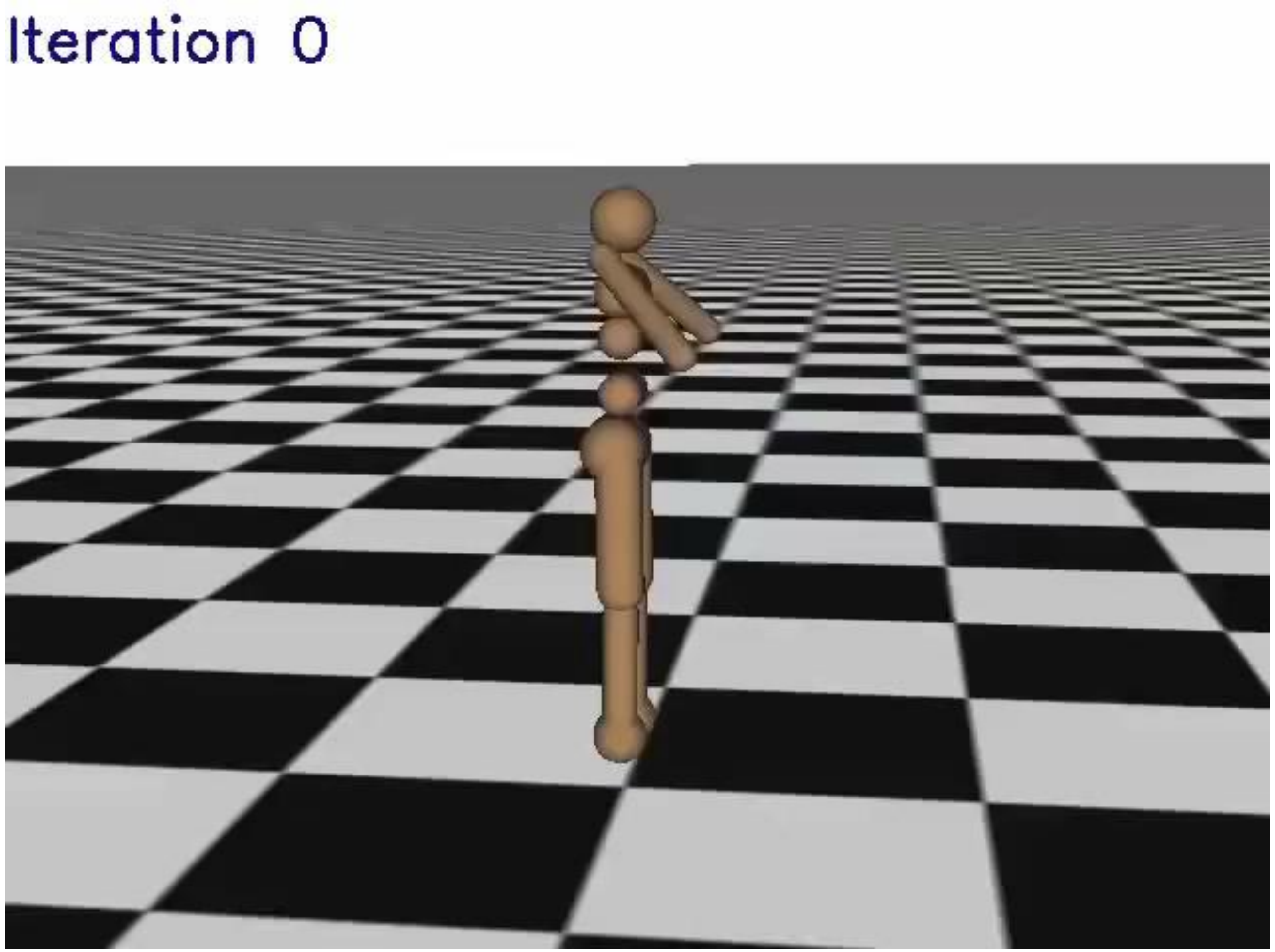
- Playing Atari with deep reinforcement learning, Mnih et al. ‘13
- Q-learning with convolutional neural networks

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

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