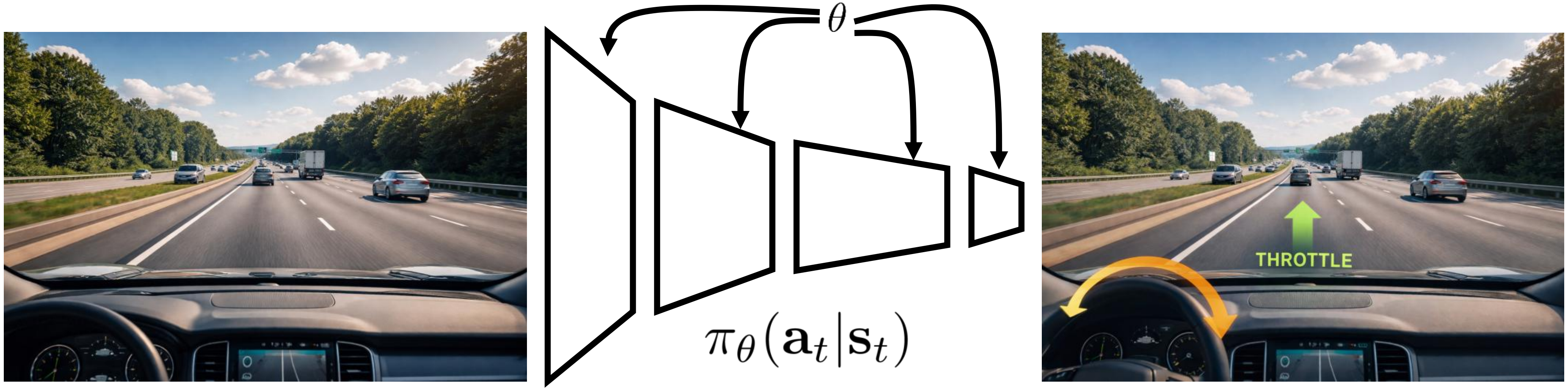
Reinforcement Learning Basics

CS 185/285

Instructor: Sergey Levine
UC Berkeley

Part 1:
The Markov decision process

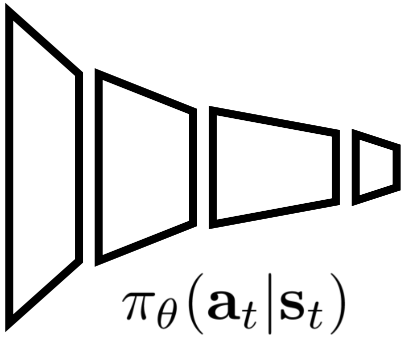


$\mathbf{s}_t$ $\mathbf{a}_t$

(we'll come back to partially observed settings later)

Imitation learning: $\arg\max_\theta \sum_{i=1}^{N} \sum_{t=1}^{H} \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})$

what if we don't have good demonstration data?



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

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

which action is better or worse?

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

tells us which states and actions are better

$\mathbf{s}_t$, $\mathbf{a}_t$, $r(\mathbf{s}_t, \mathbf{a}_t)$, and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ define Markov decision process

high reward

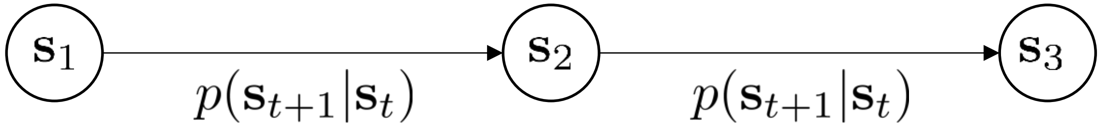
low reward

Basic definition: Markov chain

The most basic probabilistic model of a dynamical system

No actions yet, only states

Andrey Markov



Markov property

$\mathbf{s}_{t+1} \perp \mathbf{s}_{t-1} | \mathbf{s}_t$

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

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

$\mathcal{T}$ – transition operator $p(\mathbf{s}_{t+1}|\mathbf{s}_t)$

why "operator"?

$$p(\mathbf{s}_{t+1}) = \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\mathbf{s}_t)p(\mathbf{s}_t)$$

current state marginal

$$\mu_t = \begin{bmatrix} p(\mathbf{s}_t = 1) \\ p(\mathbf{s}_t = 2) \\ \dots \end{bmatrix}$$

let $\mathcal{T}_{i,j} = p(\mathbf{s}_{t+1} = i | \mathbf{s}_t = j)$ $\mu_{t+1} = \mathcal{T}\mu_t$

This bit of linear algebra will be important later!

Basic definition: Markov decision process

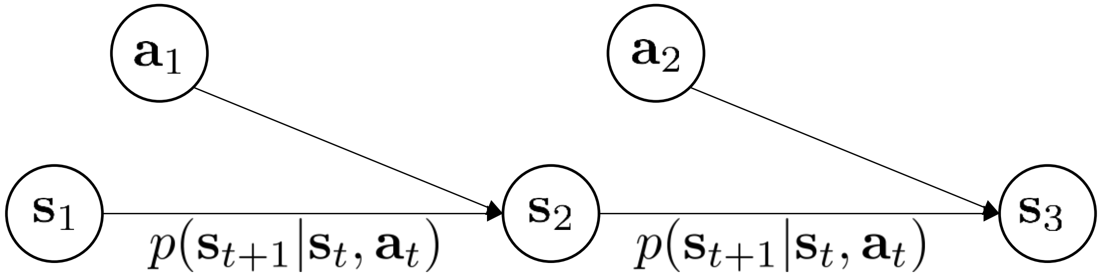
$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

states $\mathbf{s} \in \mathcal{S}$

actions $\mathbf{a} \in \mathcal{A}$

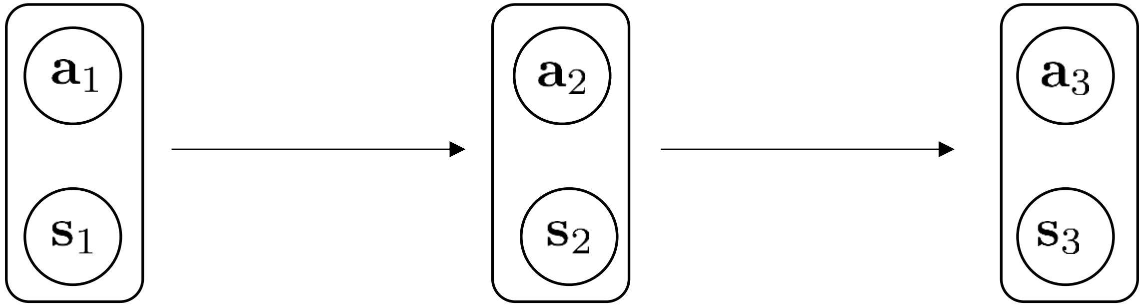
transition operator $\mathcal{T}$

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



Richard Bellman

Can we turn a Markov decision process into a Markov chain?



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

Partially observed Markov decision process

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

states $\mathbf{s} \in \mathcal{S}$

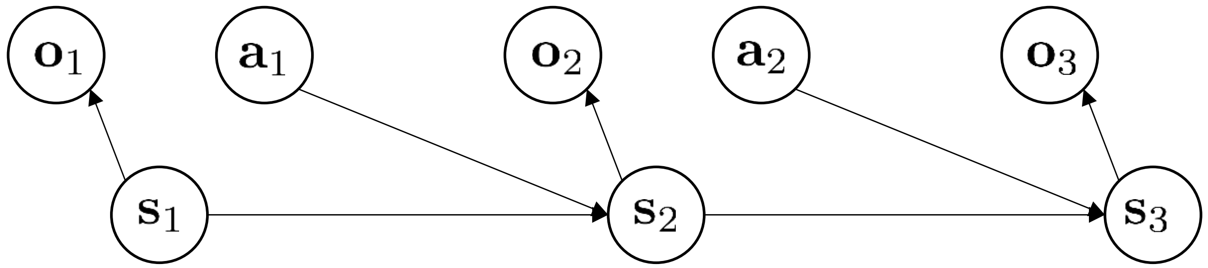
actions $\mathbf{a} \in \mathcal{A}$

observations $\mathbf{o} \in \mathcal{O}$

transition operator $\mathcal{T}$

emission operator $\mathcal{E}$

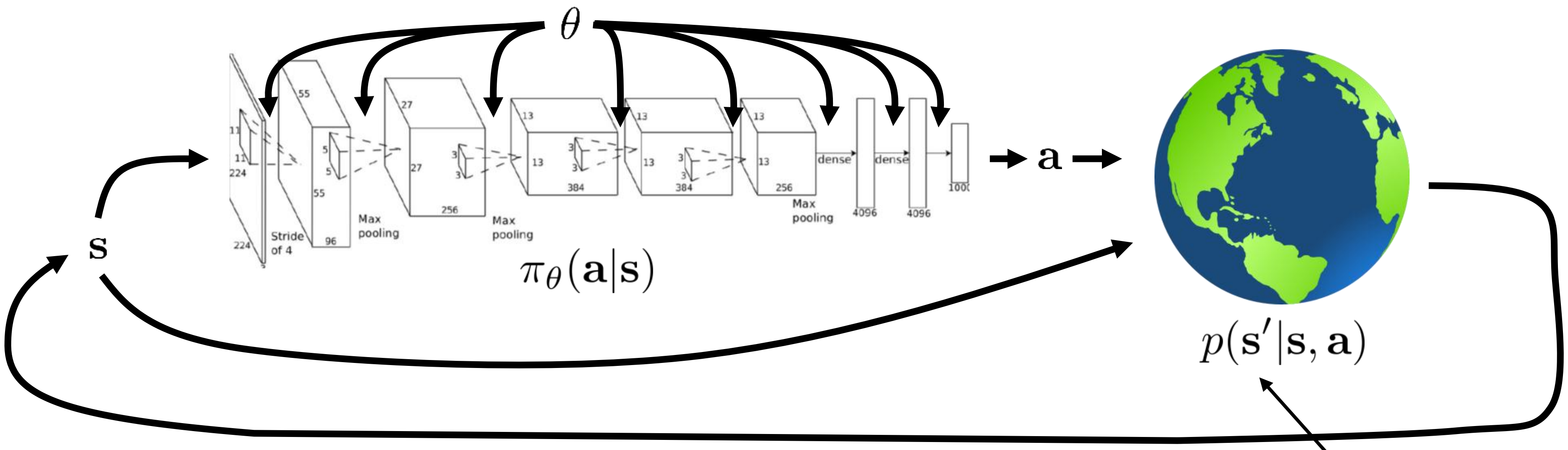
reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



Can you write this as a Markov chain?

For now we'll stick to the fully observed case

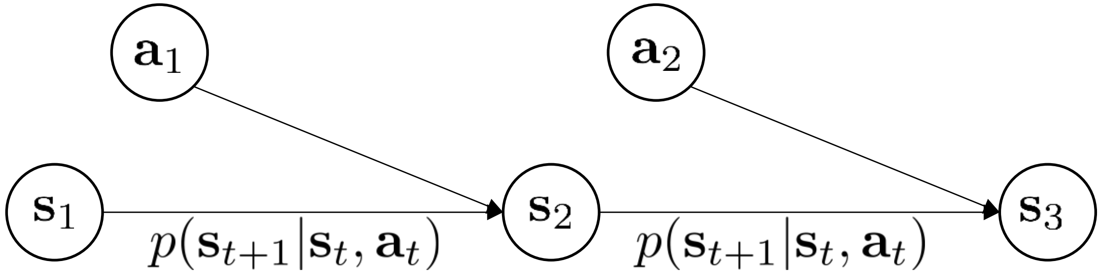
A few useful concepts



this is just shorthand so we don't need to write $t+1$

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

where did this come from??



A few useful concepts

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

"trajectory distribution"

$$\text{state marginal: } p_\theta(\mathbf{s}_t) = \sum_{\mathbf{a}_{1:H}, \mathbf{s}_{1:t-1}, \mathbf{s}_{t+1:H}} p(\mathbf{s}_1) \prod_{t'=1}^{t-1} \pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'}) p(\mathbf{s}_{t'+1}|\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

$$= \sum_{\mathbf{a}_{1:t-1}, \mathbf{s}_{1:t-1}} p(\mathbf{s}_1) \prod_{t'=1}^{t-1} \pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'}) p(\mathbf{s}_{t'+1}|\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

Question: what if we just want samples from the state marginal?

we'll learn about some other ways to estimate marginals later

Part 2:
Defining the objective

The objective of RL

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

tells us which states and actions are better

high reward

low reward

by the time you're about to hit someone, it's too late!

need to optimize for **long-term** reward

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

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

The Markov chain view

$$E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] = \sum_{t=1}^{H} E_{\mathbf{s}_1, \mathbf{a}_t \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)}\left[r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

state-action marginal

is there a simpler way to write this?

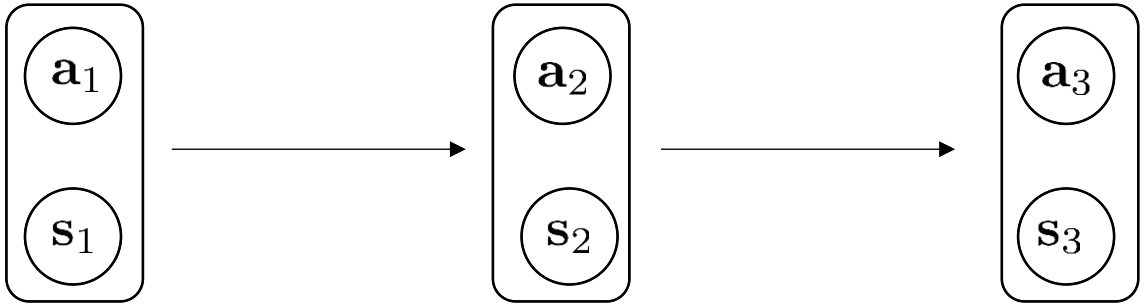
$$p(\mathbf{s}_t, \mathbf{a}_t) = \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \sum_{\mathbf{a}_{1:t-1}, \mathbf{s}_{1:t-1}} p(\mathbf{s}_1) \prod_{t'=1}^{t-1} \pi_\theta(\mathbf{a}_{t'} | \mathbf{s}_{t'}) p(\mathbf{s}_{t'+1} | \mathbf{s}_{t'}, \mathbf{a}_{t'})$$

$$\mu_{t+1} = \mathcal{T}_\theta \mu_t$$

$$\mu_t = \begin{bmatrix} p(\mathbf{s}_1 = 1, \mathbf{a}_1 = 1) \\ p(\mathbf{s}_1 = 1, \mathbf{a}_1 = 2) \\ \dots \\ p(\mathbf{s}_1 = 2, \mathbf{a}_1 = 1) \\ p(\mathbf{s}_1 = 2, \mathbf{a}_1 = 2) \\ \dots \end{bmatrix} \begin{matrix} 1^{\text{st}} \text{ tuple} \\ 2^{\text{nd}} \text{ tuple} \\ \\ \\ \\ \\ \end{matrix}$$

$$\mathcal{T}_{\theta,i,j} = p(\mathbf{s}' = s_i | \mathbf{s} = s_j, \mathbf{a} = a_j) \pi_\theta(\mathbf{a}' = a_i | \mathbf{s}' = s_i)$$

j^{th} state-action tuple



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

The Markov chain view

$$\sum_{t=1}^{H} E_{\mathbf{s}_1,\mathbf{a}_t \sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] = \sum_{t=1}^{H}\sum_{\mathbf{s}_t,\mathbf{a}_t} p_\theta(\mathbf{s}_t,\mathbf{a}_t) r(\mathbf{s}_t,\mathbf{a}_t)$$

$$= \sum_{t=1}^{H}\sum_{i} \mu_{t,i}\vec{r}_i$$

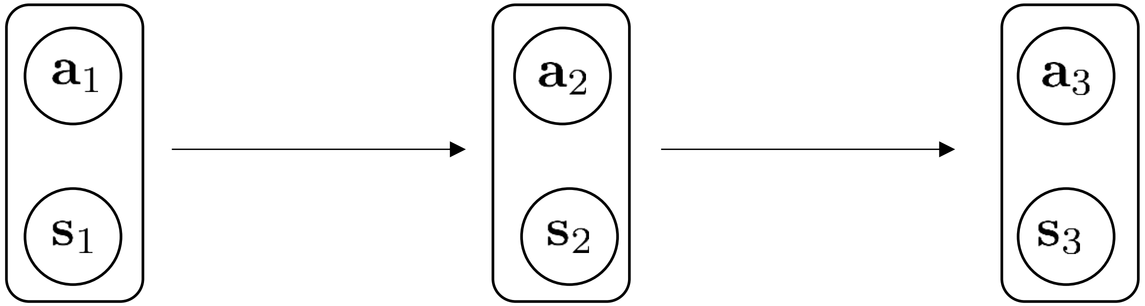
$$= \sum_{t=1}^{H} \mu_t^T \vec{r}$$

$$= \left[\sum_{t=1}^{H} \mathcal{T}_\theta^{t-1}\mu_1\right]^T \vec{r}$$

$$\vec{r} = \begin{bmatrix} r(\mathbf{s}=1,\mathbf{a}=1) \\ r(\mathbf{s}=1,\mathbf{a}=2) \\ \dots \\ r(\mathbf{s}=2,\mathbf{a}=1) \\ r(\mathbf{s}=2,\mathbf{a}=2) \\ \dots \end{bmatrix} \quad \mu_t = \begin{bmatrix} p(\mathbf{s}_1=1,\mathbf{a}_1=1) \\ p(\mathbf{s}_1=1,\mathbf{a}_1=2) \\ \dots \\ p(\mathbf{s}_1=2,\mathbf{a}_1=1) \\ p(\mathbf{s}_1=2,\mathbf{a}_1=2) \\ \dots \end{bmatrix}$$

$$\mu_{t+1} = \mathcal{T}_\theta \mu_t$$

$$\mu_t = \mathcal{T}_\theta^{t-1}\mu_1$$



What if the horizon is *infinite?*

$$\lim_{H\to\infty} \frac{1}{H} \sum_{t=1}^{H} E_{\mathbf{s}_1,\mathbf{a}_t \sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)} [r(\mathbf{s}_t,\mathbf{a}_t)] = E_{\mathbf{s}_1,\mathbf{a}_t \sim p(\mathbf{s}_t,\mathbf{a}_t)} [r(\mathbf{s}_t,\mathbf{a}_t)] = \bar{\mu}^T \vec{r}$$

↑ stationary distribution $\bar{\mu}$

We usually don't solve for the stationary distribution directly in RL algorithms

But understanding how we can manipulate the RL objective with linear algebra is very useful for understanding RL algorithms theoretically

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

$\bar{\mu} = \mathcal{T}_\theta \bar{\mu}$

stationary = the same before and after transition

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

$\bar{\mu}$ is eigenvector of $\mathcal{T}_\theta$ with eigenvalue 1!

(always exists under some regularity conditions)

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

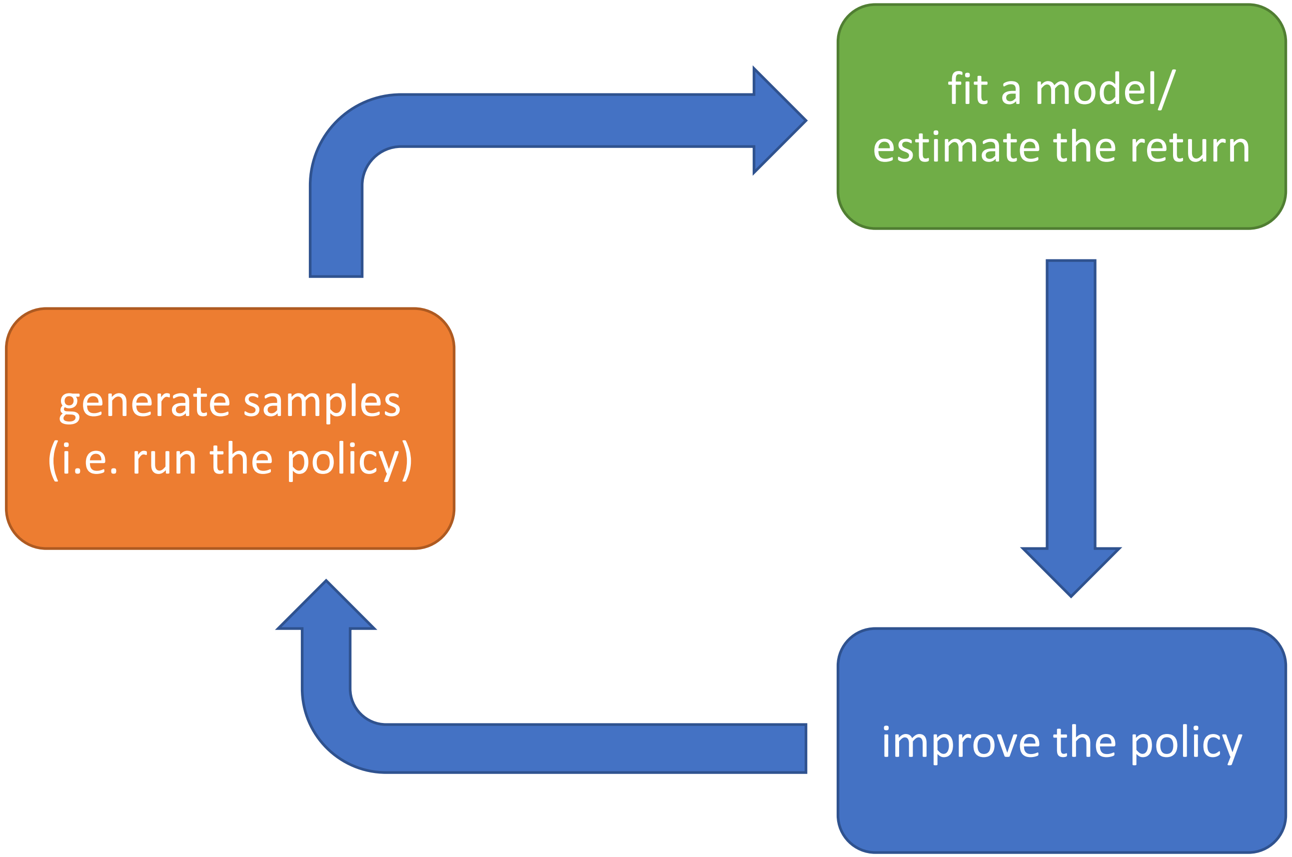
$\pi_\theta(\mathbf{a} = \text{fall}) = \theta$

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

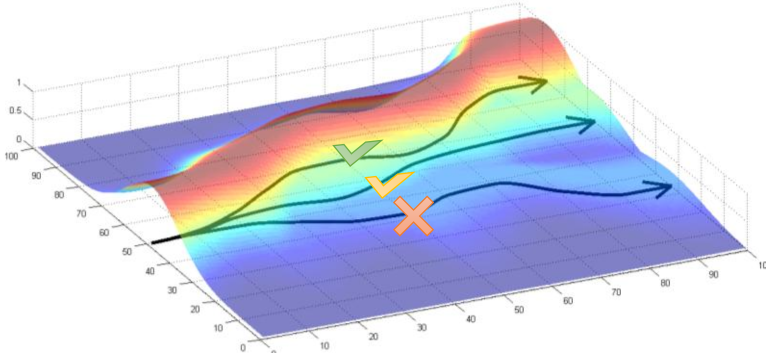
Intermission

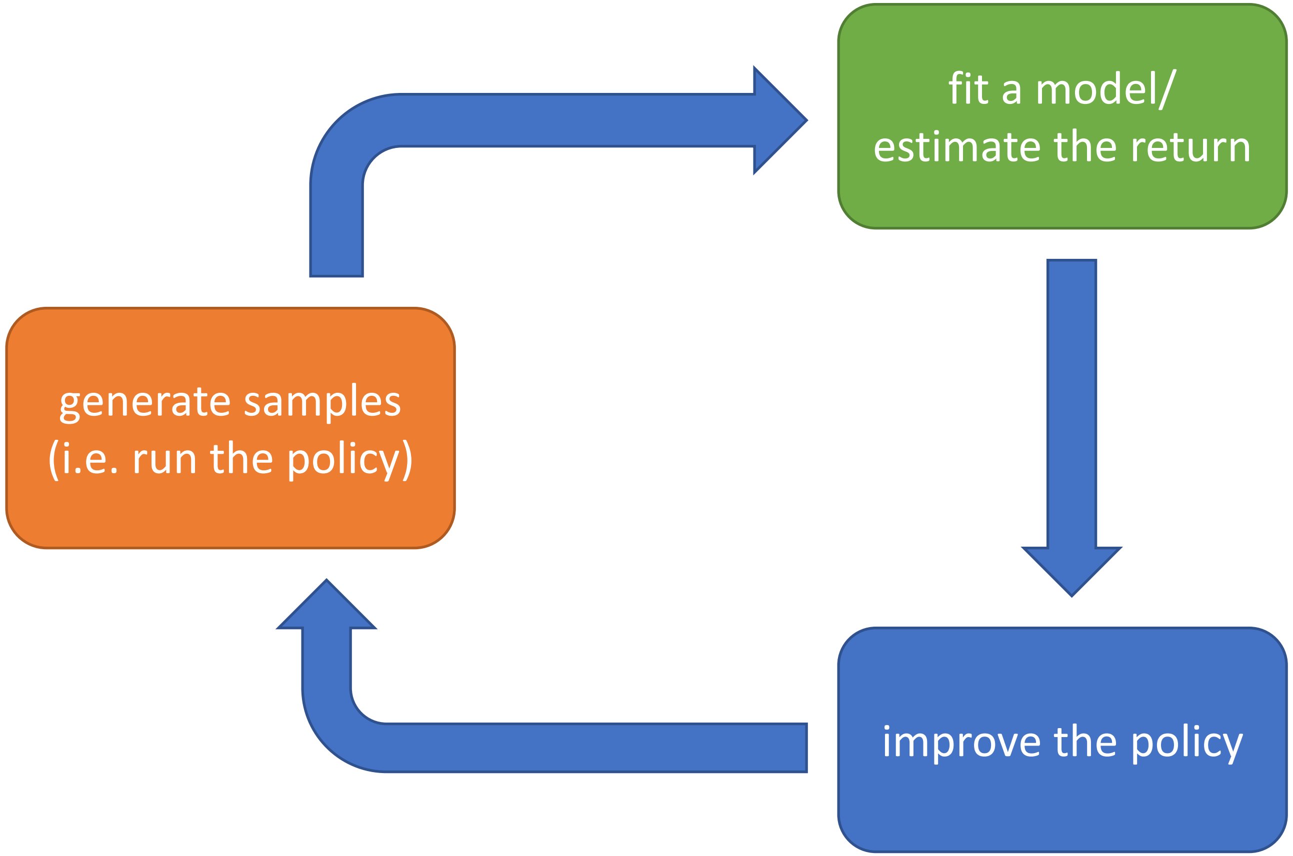
Part 3:
Anatomy of an RL algorithm

The anatomy of a reinforcement learning algorithm



A simple example

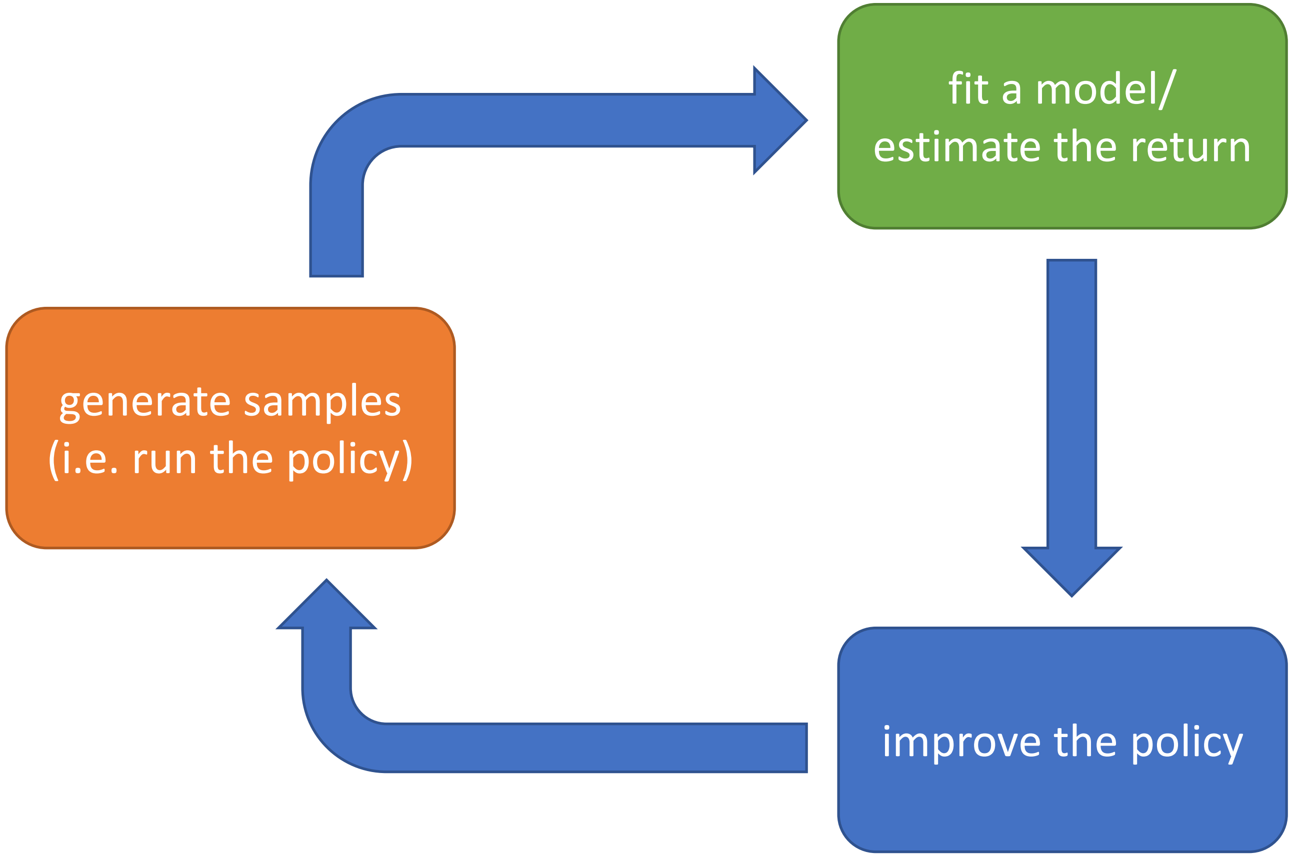




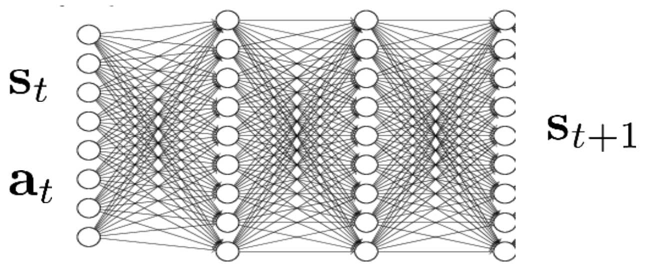
$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

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

Another example: RL by backprop



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

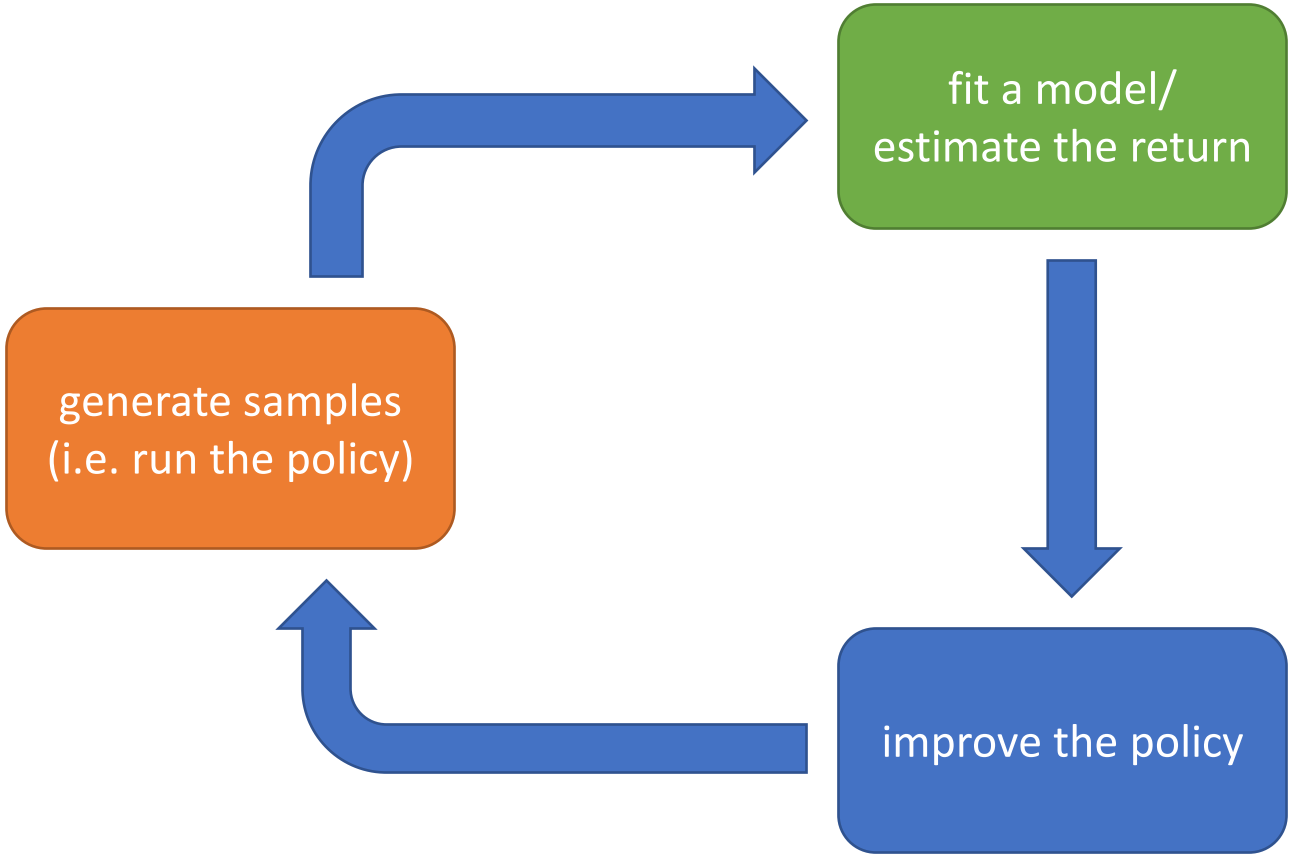


backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Which parts are expensive?

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

trivial, fast



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

fast simulator:
up to 10000x real time

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

Part 4:
Value functions and Q-functions

How do we deal with all these expectations?

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

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

what if we knew this part?

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

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

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 $\mathbf{s}_t$

Definition: value function

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

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

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

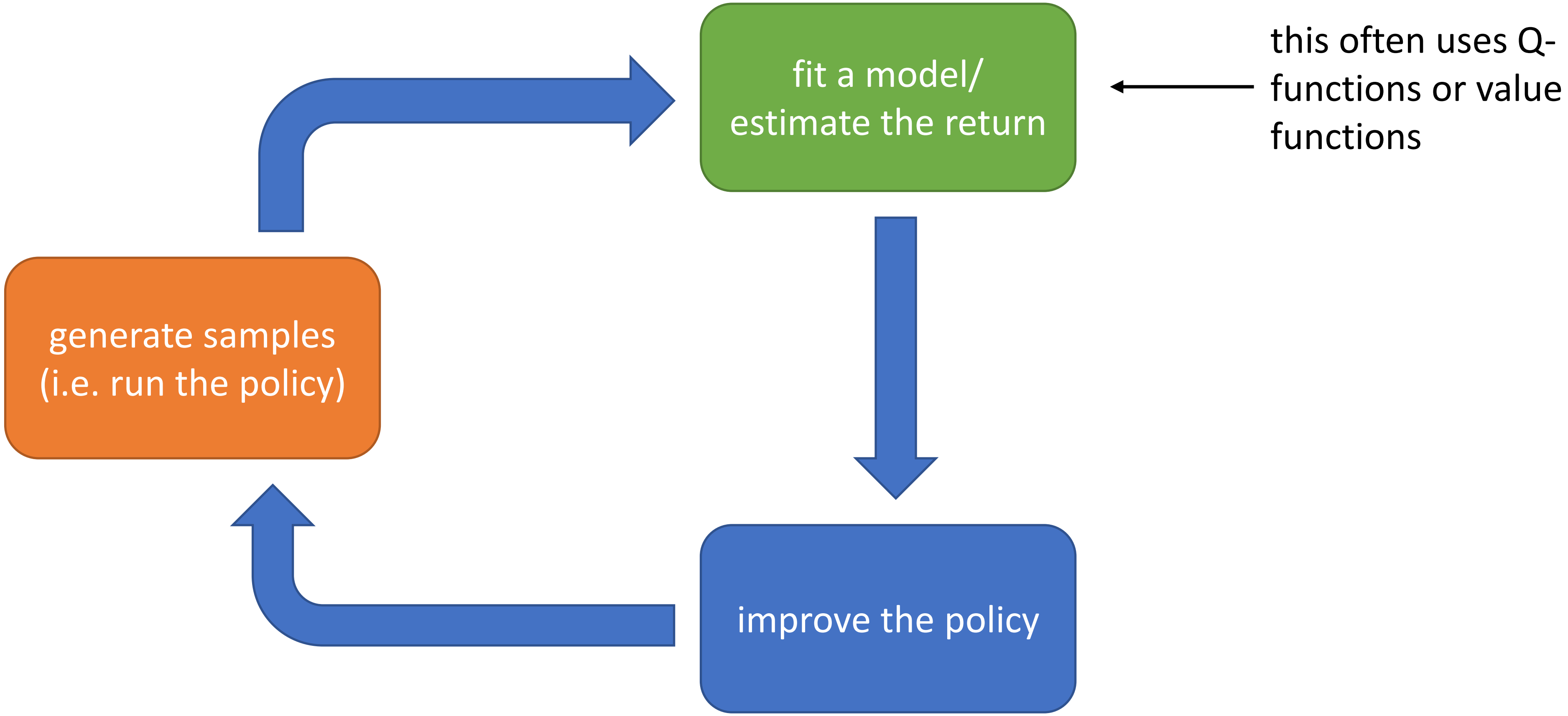
Idea 2: compute gradient to increase probability of good actions $\mathbf{a}$:

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



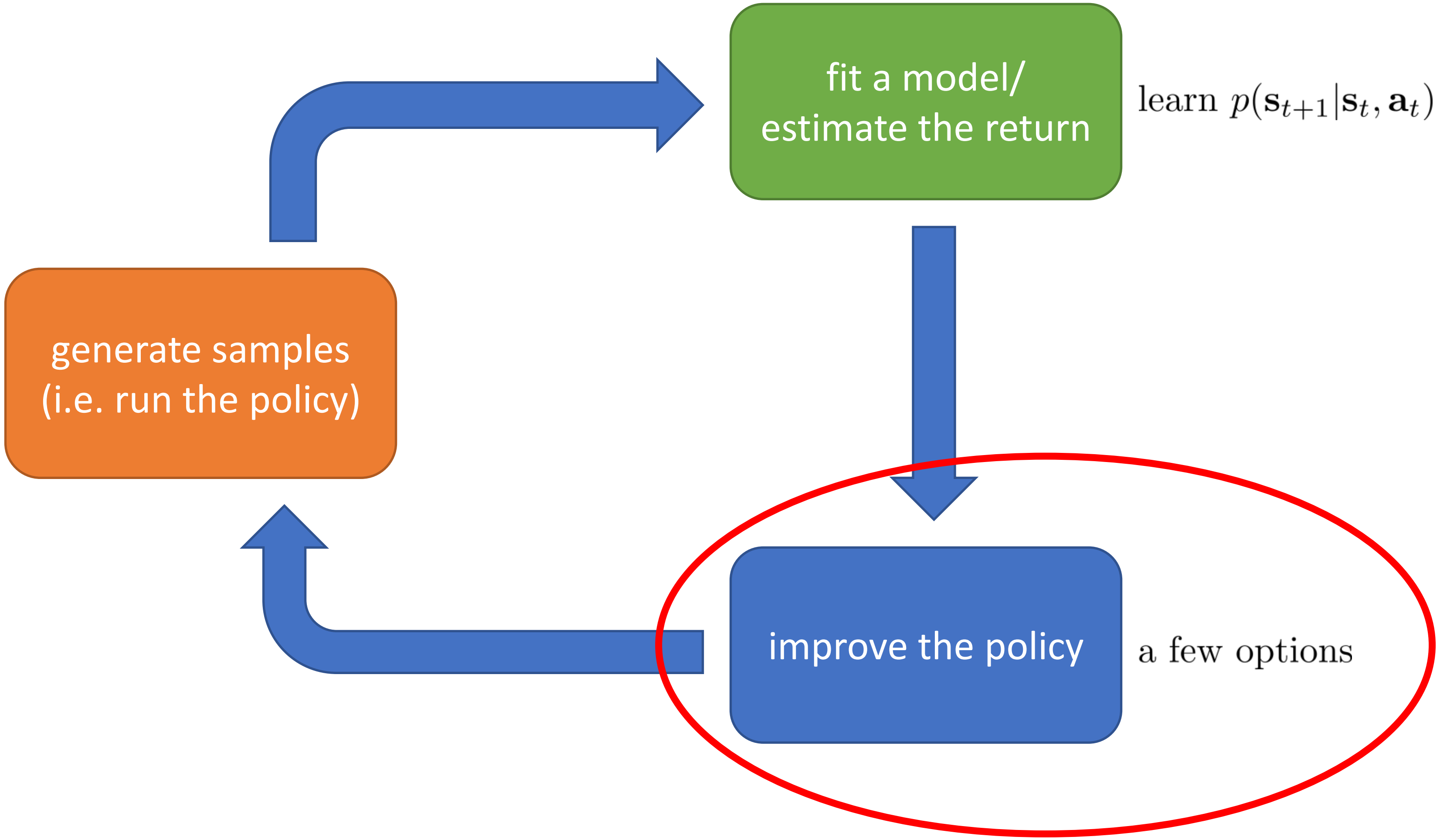
Part 5:
Types of RL algorithms

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

Model-based RL algorithms



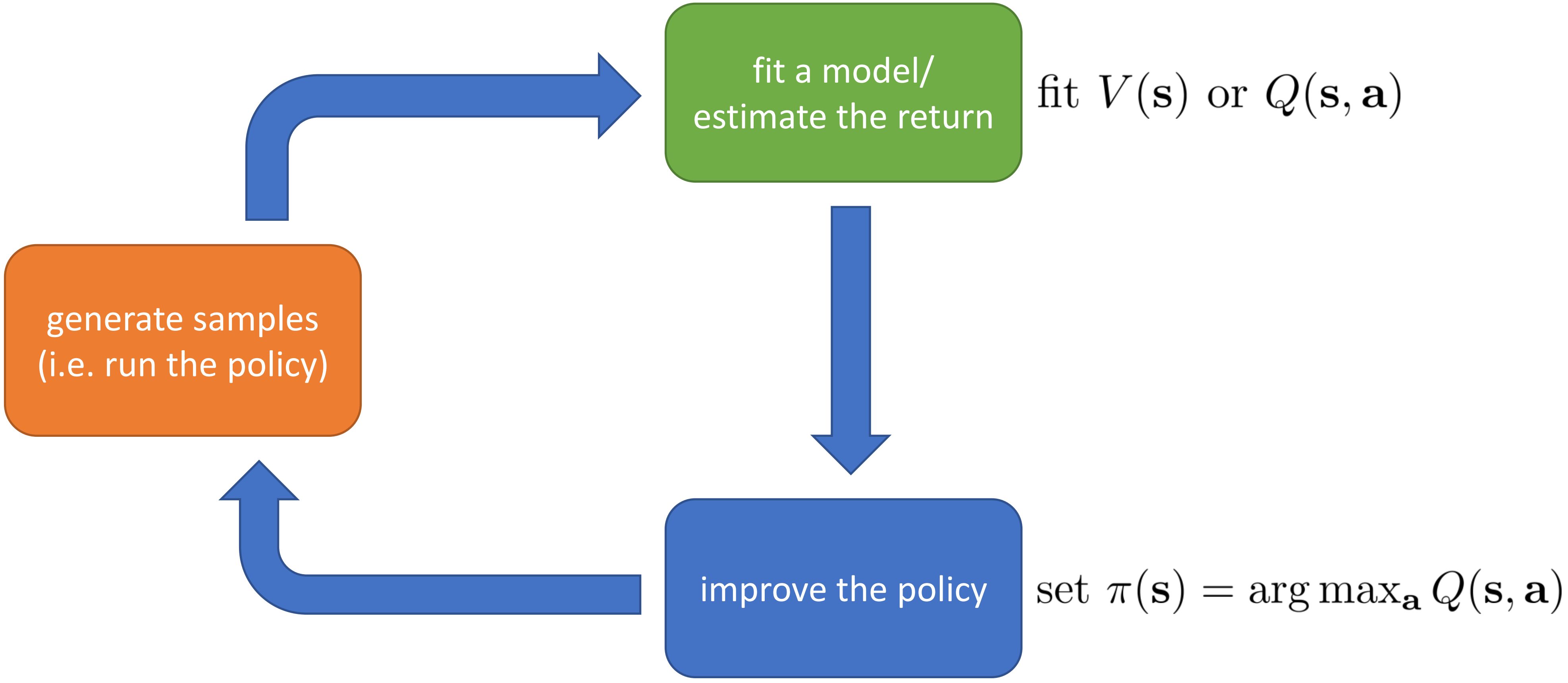
Model-based RL algorithms

improve the policy

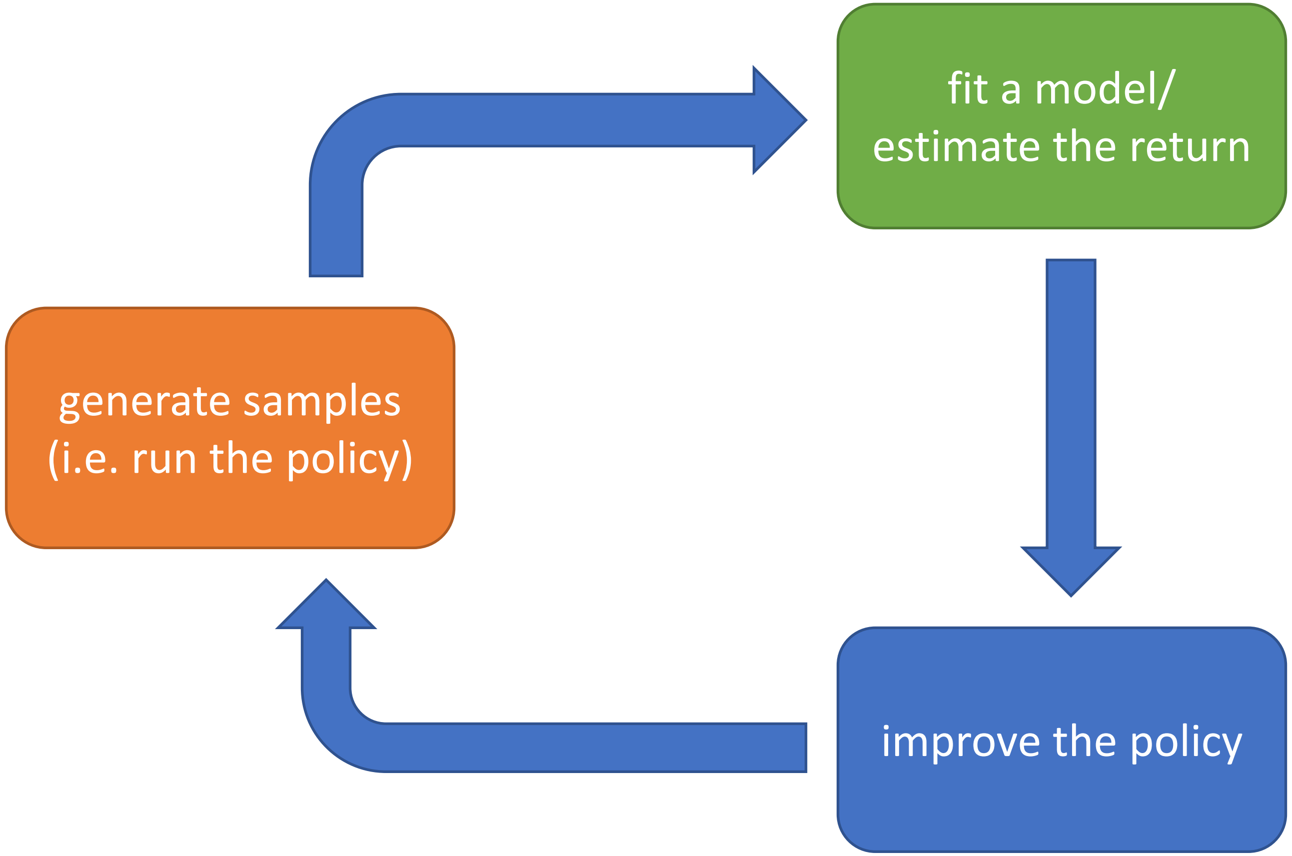
a few options

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

Value function based algorithms



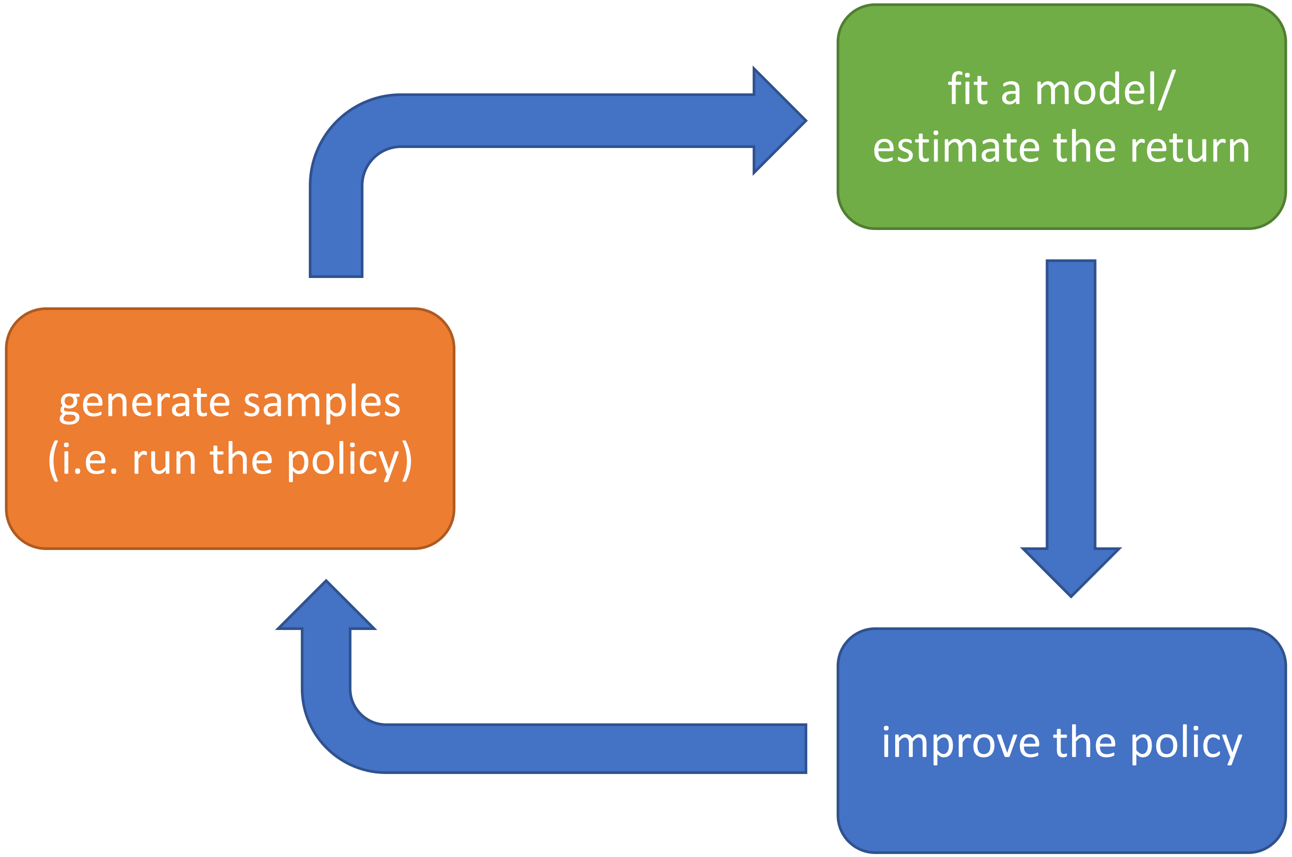
Direct policy gradients



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

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

Actor-critic: value functions + policy gradients

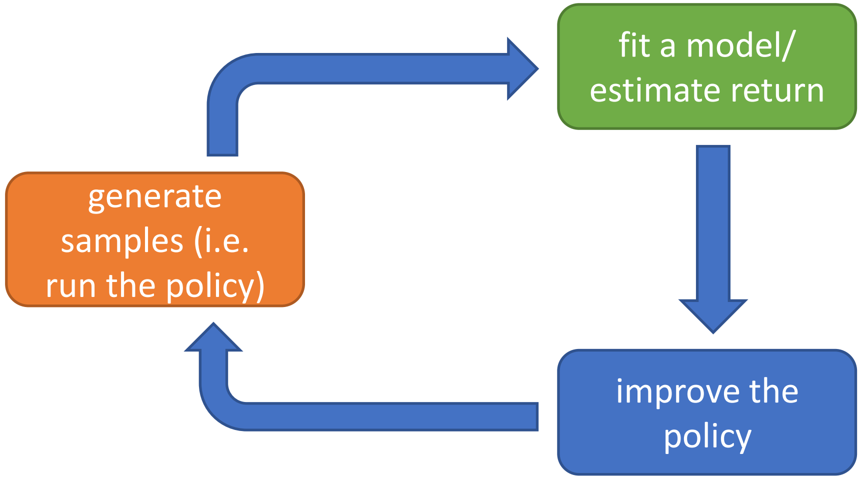


$$\text{fit } V(\mathbf{s}) \text{ or } Q(\mathbf{s}, \mathbf{a})$$

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

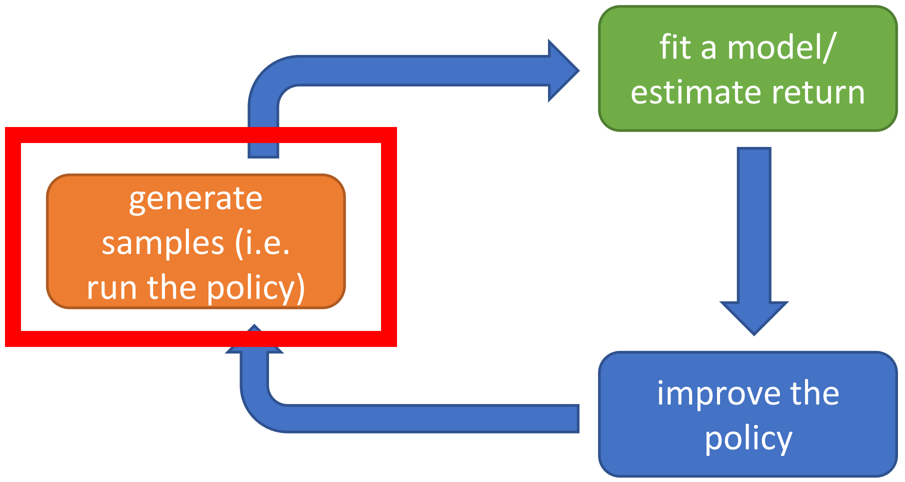
Why so many RL algorithms?

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



Comparison: sample efficiency

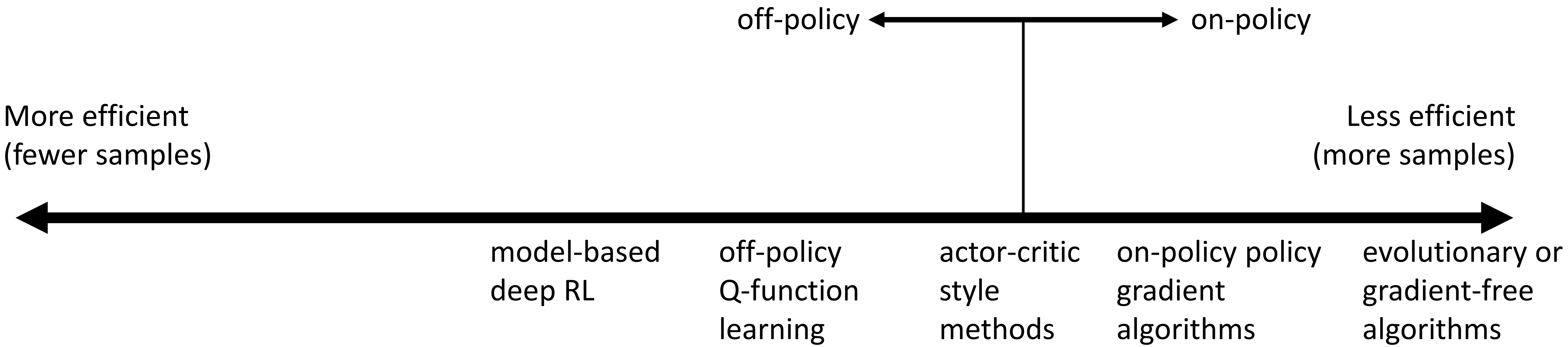
- Sample efficiency = how many samples do we need to get a good policy?
- Most important question: is the algorithm *off policy*?
  - Off policy: able to improve the policy without generating new samples from that policy
  - On policy: each time the policy is changed, even a little bit, we need to generate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



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

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

Examples of specific algorithms

- Value function methods
  - Q-learning, DQN
  - Temporal difference learning
  - Fitted value iteration
- Policy gradient methods
  - REINFORCE
  - Natural policy gradient
  - PPO
- Actor-critic algorithms
  - Asynchronous advantage actor-critic (A3C)
  - Soft actor-critic (SAC)
- Model-based RL algorithms
  - Dyna (+ “Dyna-like” methods such as MBPO)
  - muZero

We’ll learn about many 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: walking with policy gradients

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