Introduction to Reinforcement Learning

CS 294-112: Deep Reinforcement Learning

Sergey Levine

Class Notes

1. Homework 1 is due next Wednesday!
   - Remember that Monday is a holiday, so no office hours
2. Remember to start forming final project groups
   - Final project assignment document and ideas document released

Today’s Lecture

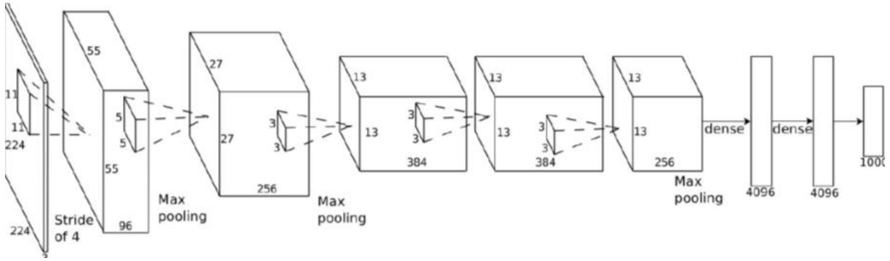
1. Definition of a Markov decision process
2. Definition of reinforcement learning problem
3. Anatomy of a RL algorithm
4. Brief overview of RL algorithm types

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

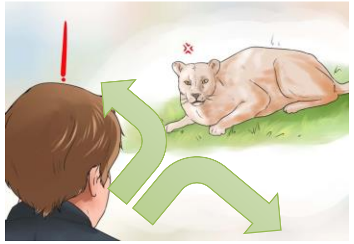
Definitions

Terminology & notation

$\mathbf{o}_t$



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

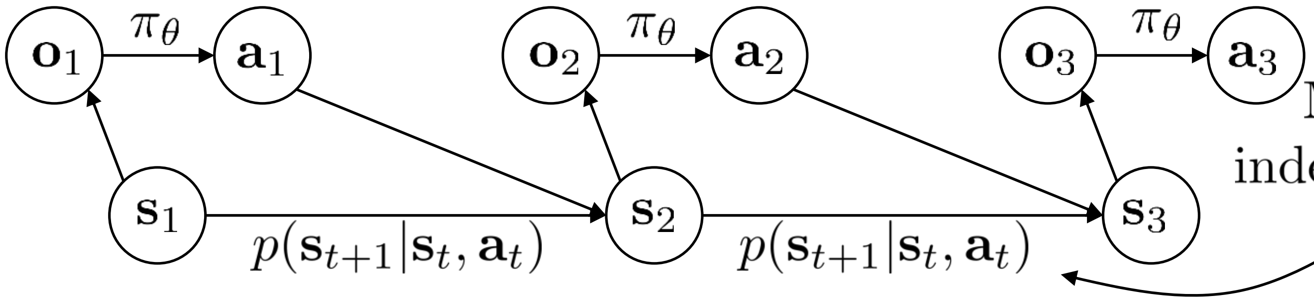


$\mathbf{a}_t$

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

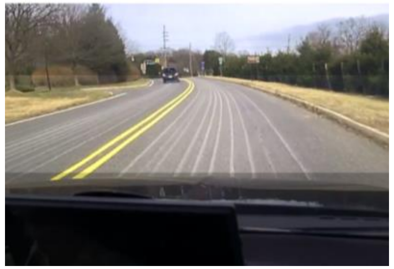
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy

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

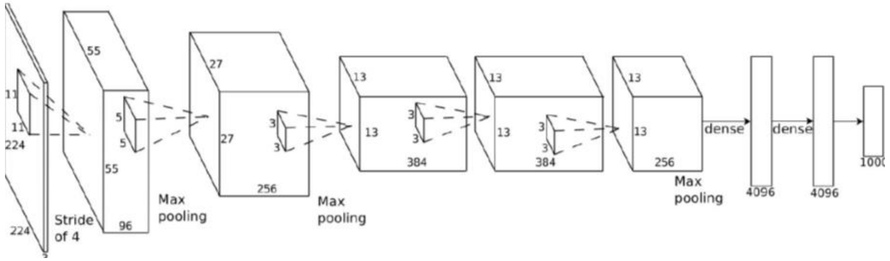


Markov property
independent of $\mathbf{s}_{t-1}$

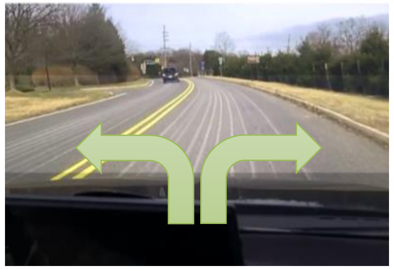
Imitation Learning



$\mathbf{o}_t$



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

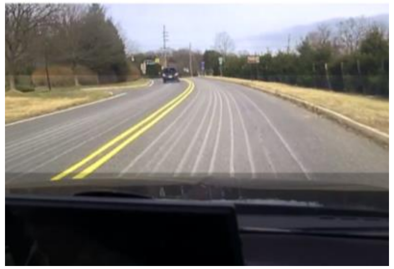


$\mathbf{a}_t$

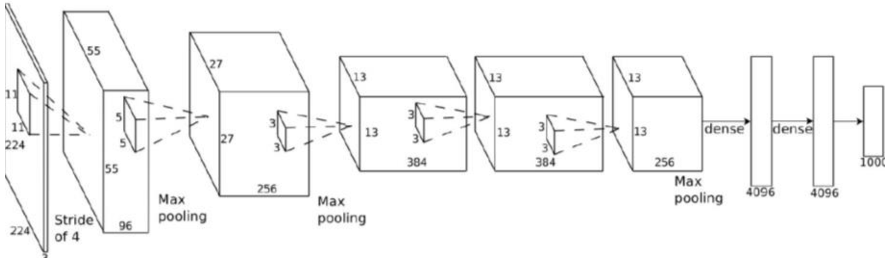
$\mathbf{o}_t$ $\mathbf{a}_t$ → training data → supervised learning → $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

Images: Bojarski et al. ‘16, NVIDIA

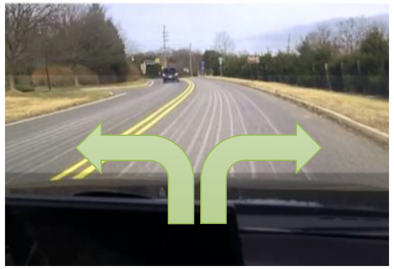
Reward functions



$\mathbf{o}_t$



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



$\mathbf{a}_t$

which action is better or worse?

$r(\mathbf{s}, \mathbf{a})$: reward function 最大化所有时间上reward

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 — $p(s_{t+1}|s_t)$

s_t的所有状态,和为1
[0.1, 0.2, 0.4, 0.3]
第一种状态概率为0.1

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

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

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

Definitions

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

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

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

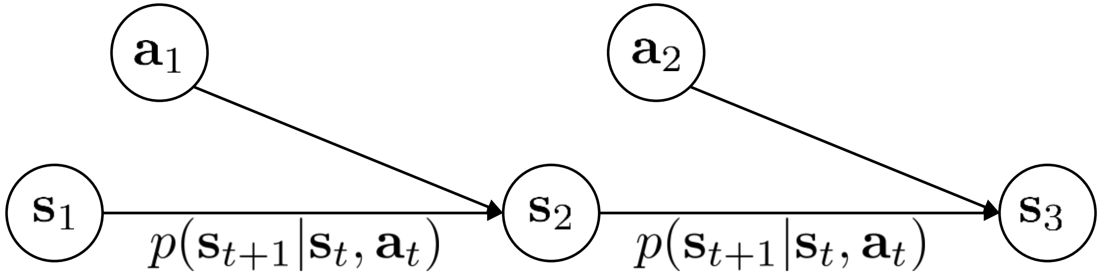
$\mathcal{T}$ – transition operator (now a tensor!)

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

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

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

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



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)

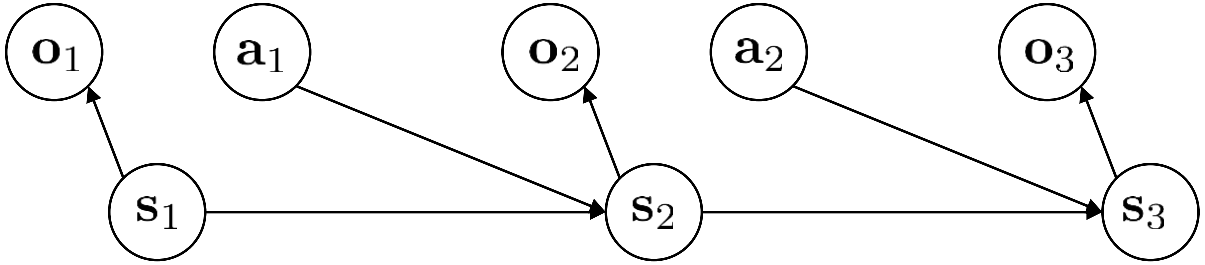
$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

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

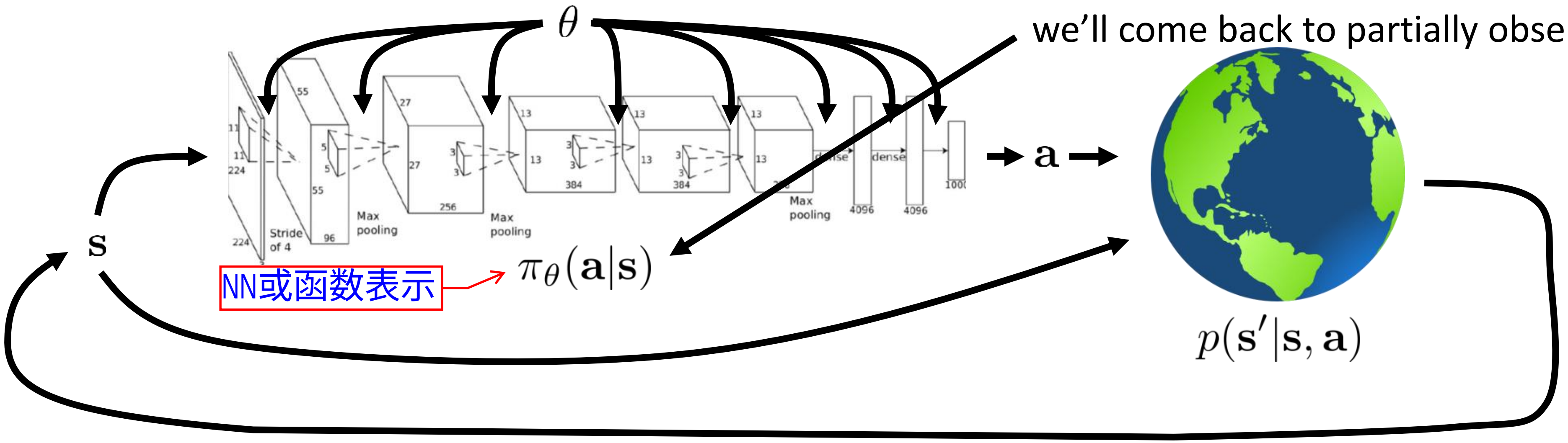
$\mathcal{T}$ – transition operator (like before)

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

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



The goal of reinforcement learning



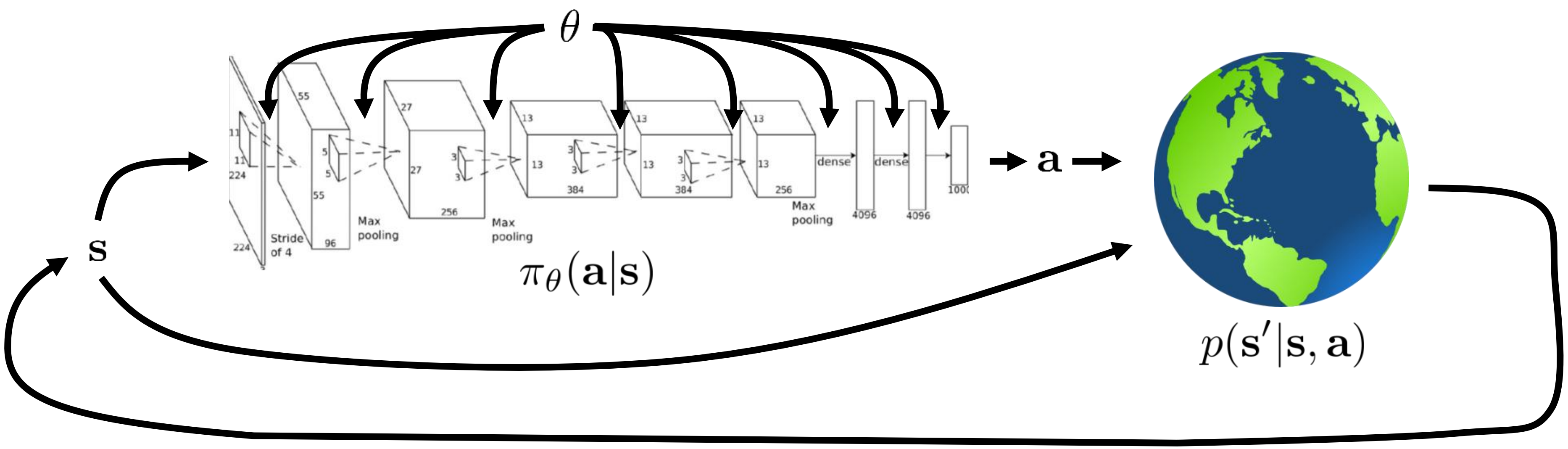
里面的sas...序列是trajectory

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

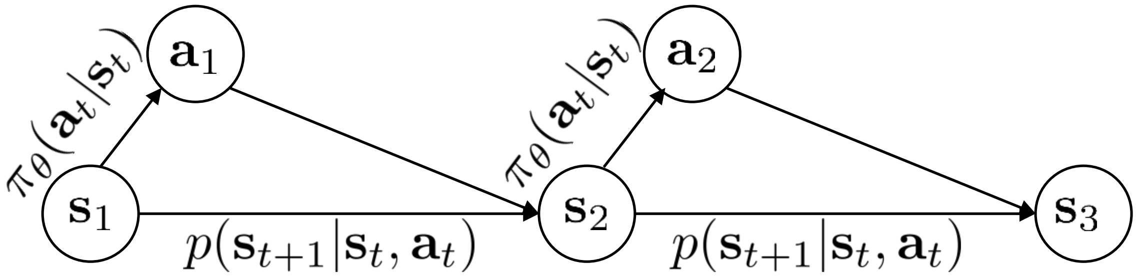
就是要学theta

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

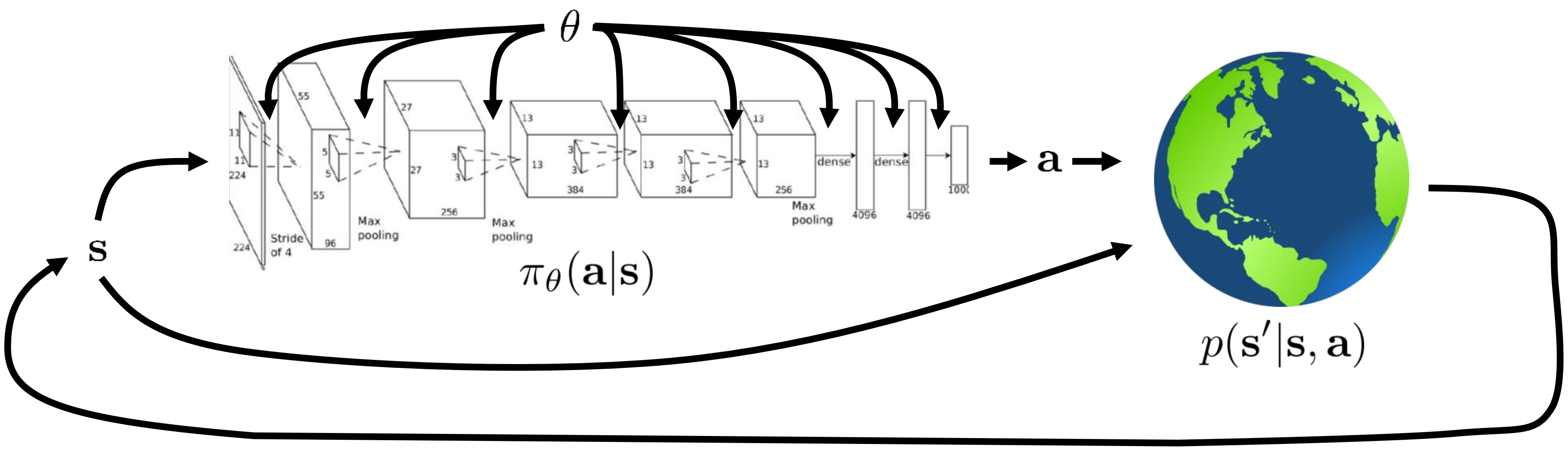
The goal of reinforcement learning



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

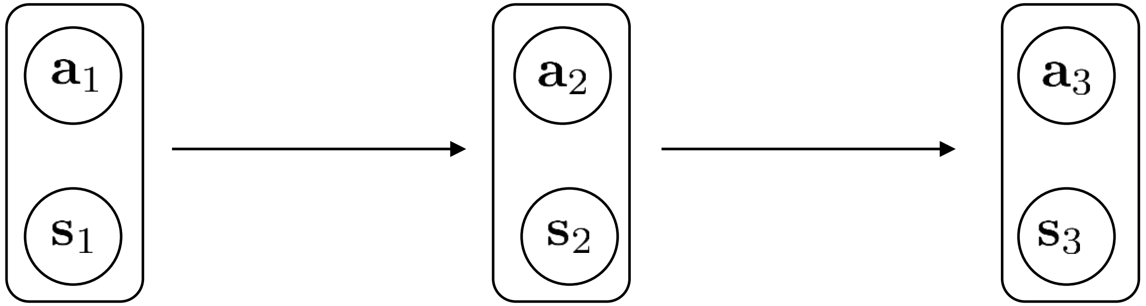


The goal of reinforcement learning



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

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



Finite horizon case: state-action marginal

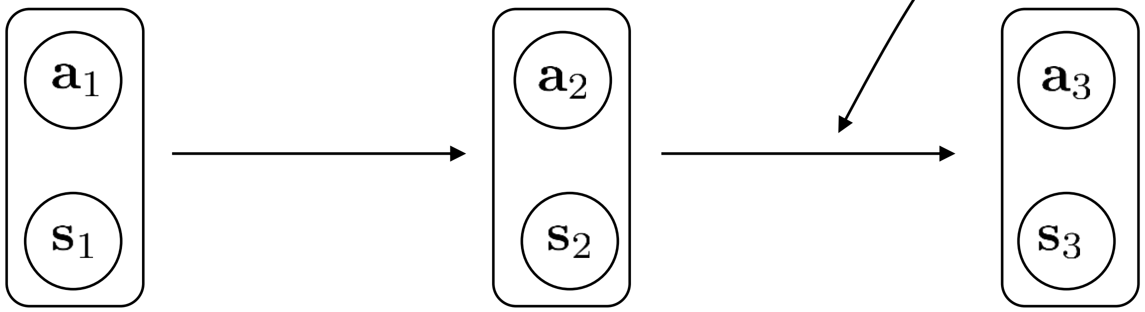
$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

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

把其他变量都求和，剩下s_t，a_t这个就叫marginal

$p_\theta(\mathbf{s}_t, \mathbf{a}_t)$ 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

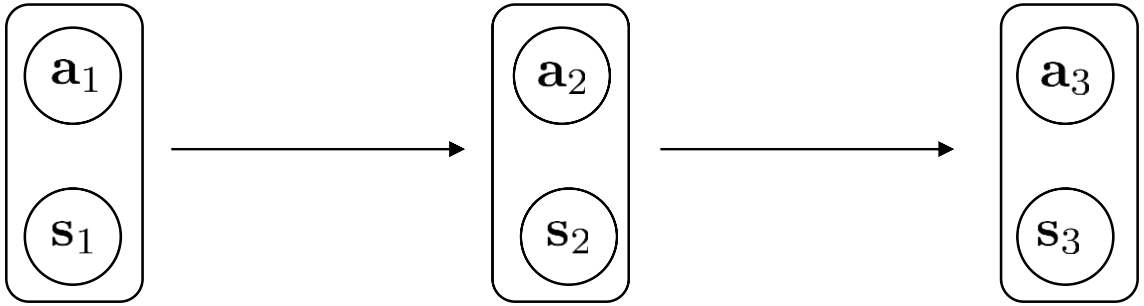
$(\mathcal{T} - \mathbf{I})\mu = 0$

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?

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

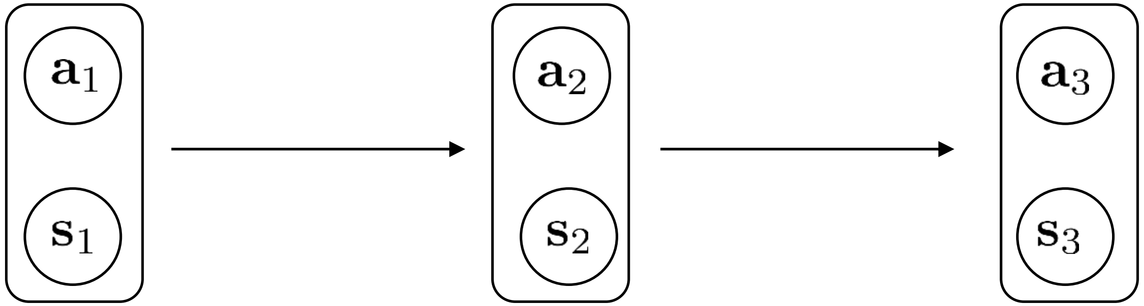
stationary = the same before and after transition

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

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*

$$p_\theta(\text{fall}) = \theta$$

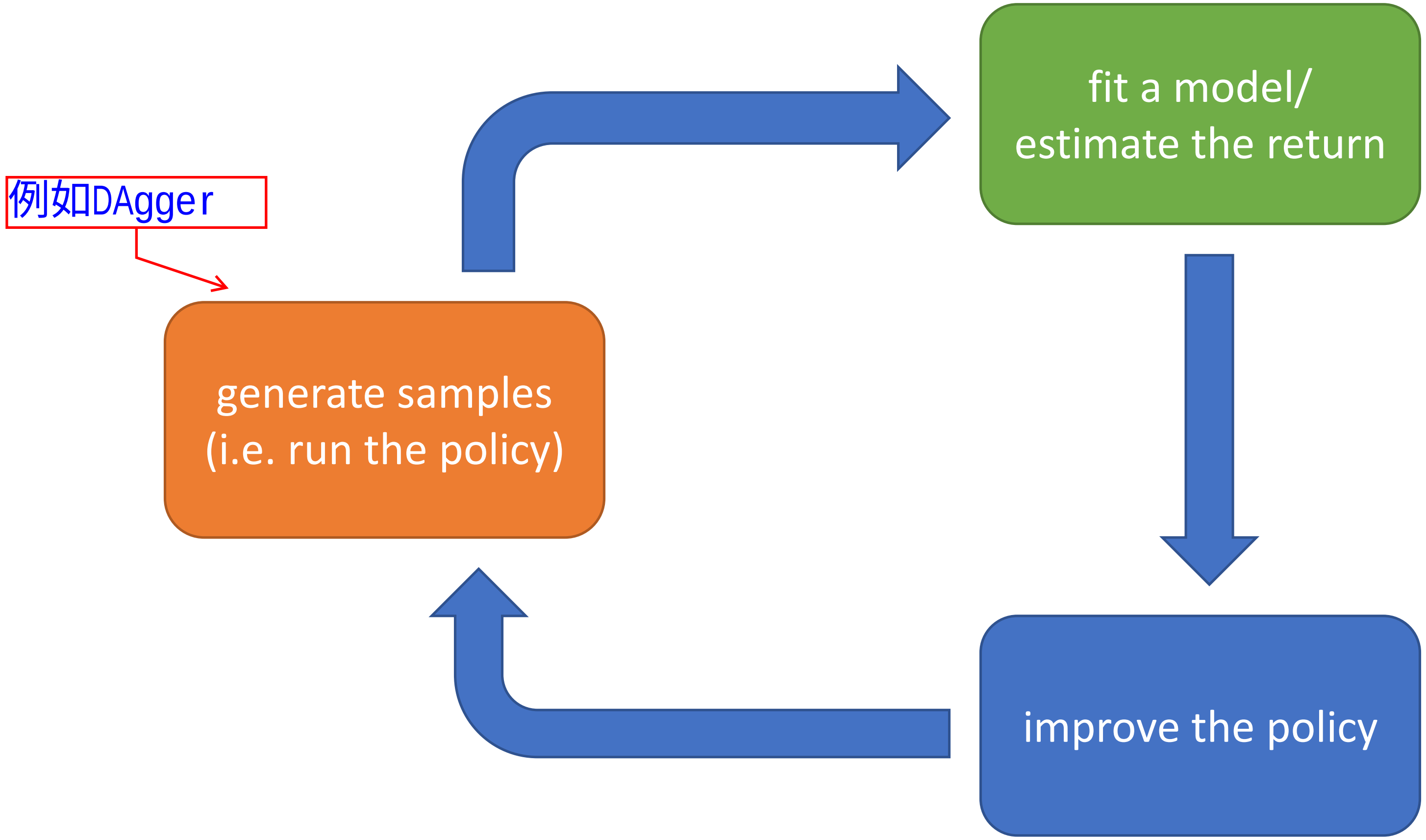
$$r(\text{fall}) - \textit{not}\ \text{smooth}$$

$$E_{p_\theta}[r(\text{fall})] - \textit{smooth}\ \text{in}\ \theta!$$

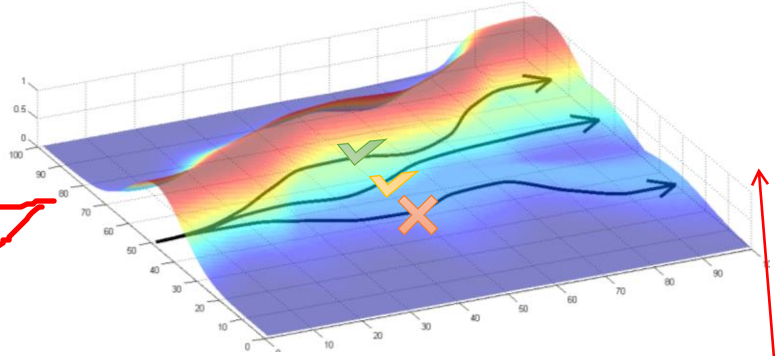
theta的目标函数是一个复杂的不连续函数。因为是一个复杂分布的期望，
所以我们很在意期望，期望可以让这个东西变得连续

Algorithms

The anatomy of a reinforcement learning algorithm



A simple example



generate samples
(i.e. run the policy)

fit a model/
estimate the return

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

对于好的，想增大概率。差的，减小概率

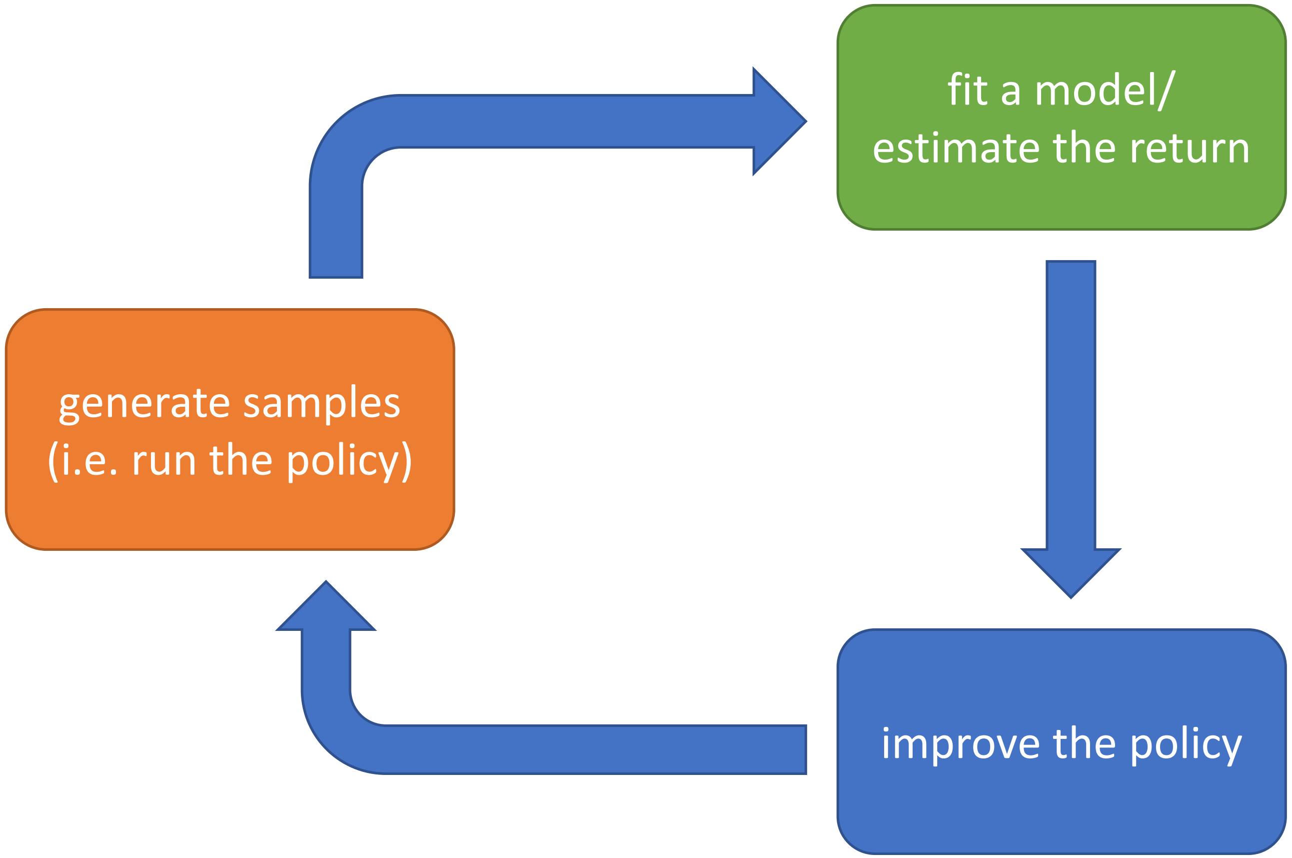
improve the policy

policy gradient

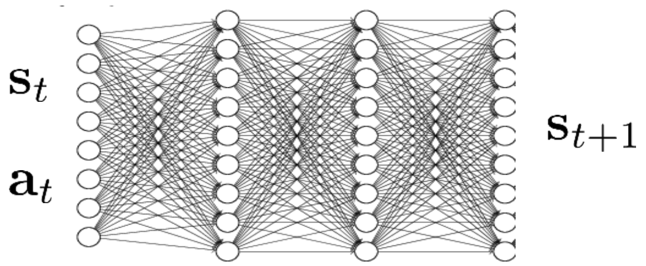
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Another example: RL by backprop

model based RL（可以预测下一个状态）



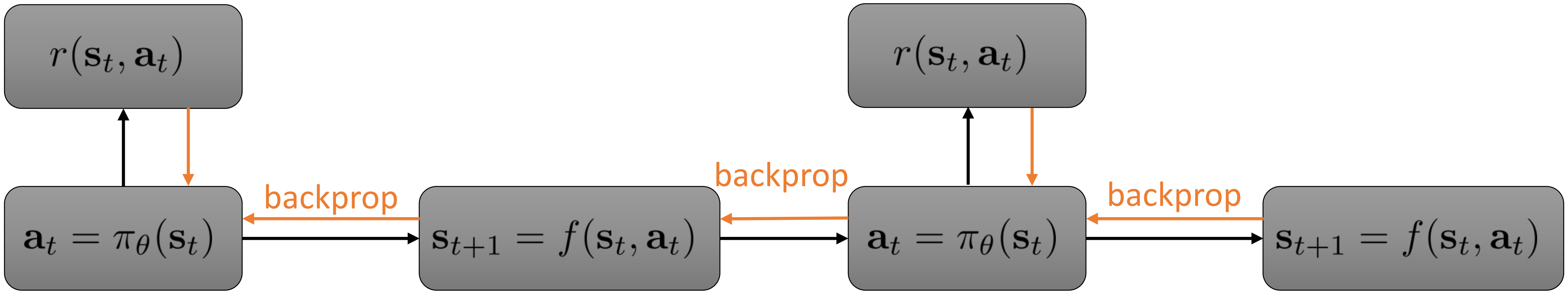
learn f_ϕ such that $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$



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

Simple example: RL by backprop

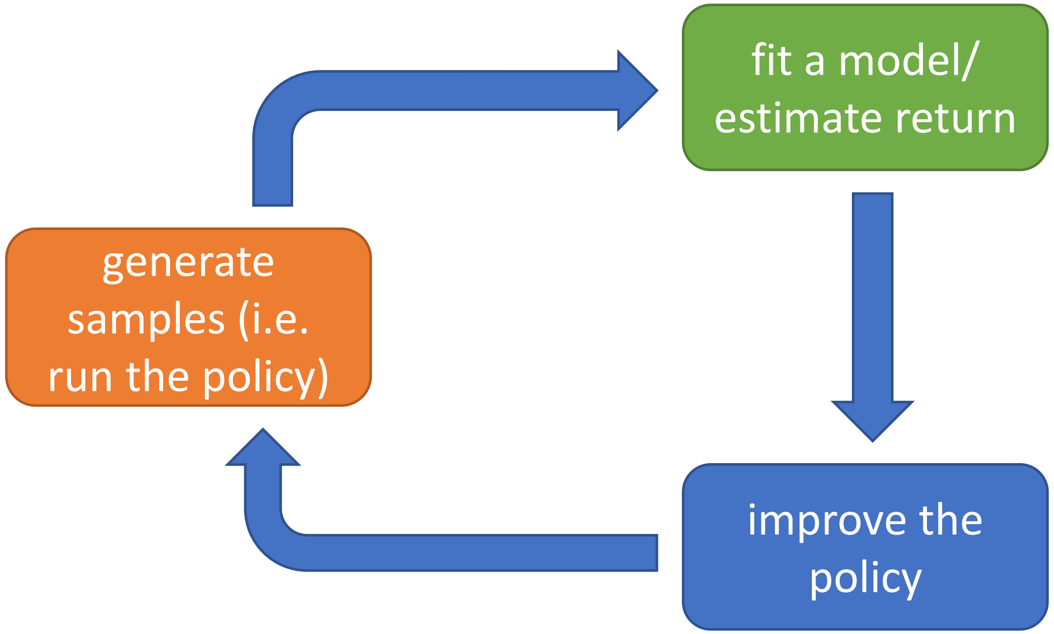
不是一个好的例子，
只是用来解释



collect data

update the model f

update the policy with backprop



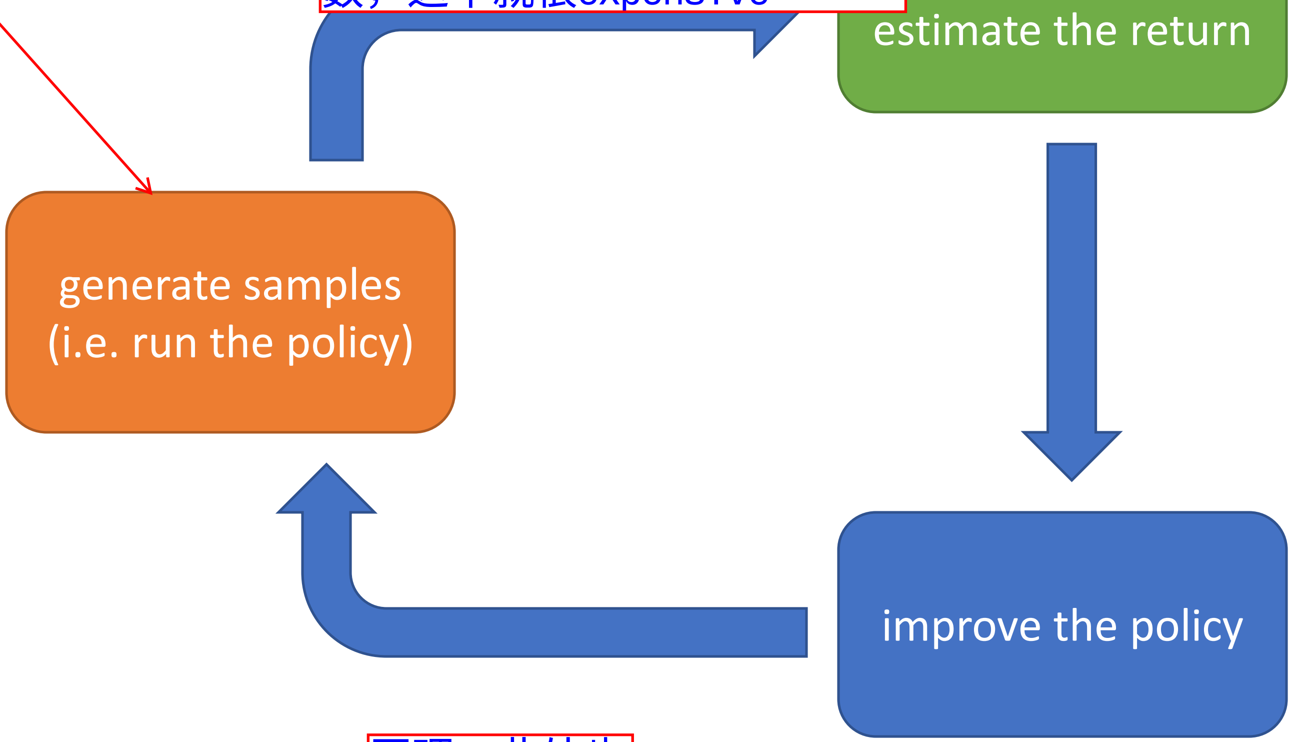
Which parts are expensive?

受限于物理特性，通常这一部分是成本很高的

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

MuJoCo simulator:
up to 10000x real time

绿框有两种：
1.对于deterministic的error-based的目标函数，这个很容易实现。一行python
2.对于需要用NN来做的目标函数，这个就很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

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

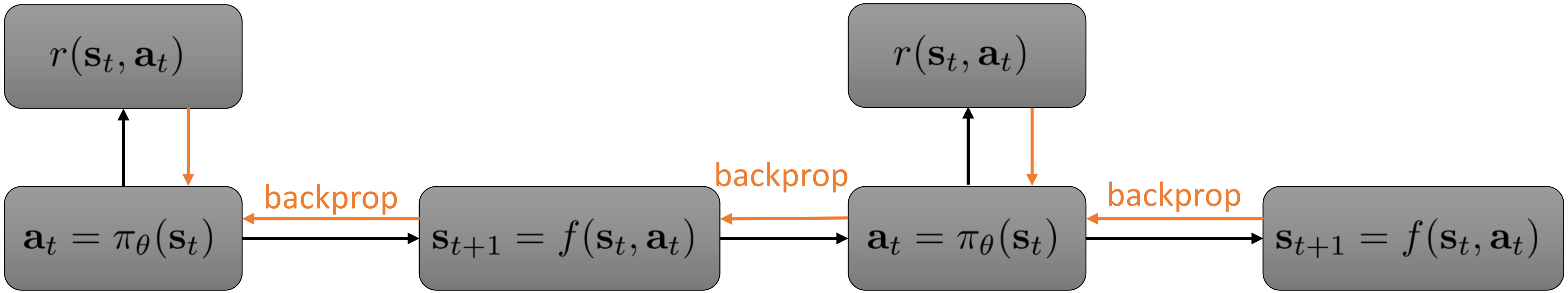
expensive

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

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

同理，此处也是由上面绿框中的函数决定容易或者简单

Why is this not enough?



- Only handles deterministic dynamics
- Only handles deterministic policies
- Only continuous states and actions
- Very difficult optimization problem
- We'll talk about this more later!

How can we work with *stochastic* systems?

Conditional expectations

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

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

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

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

change pi 来增大Q

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

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

Definition: Q-function

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

Definition: value function

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

state version of Q function

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

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

用 Q

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

V是Q在pi之下的平均
Q-V就是advantage，就是这个Q比平均水平好多少

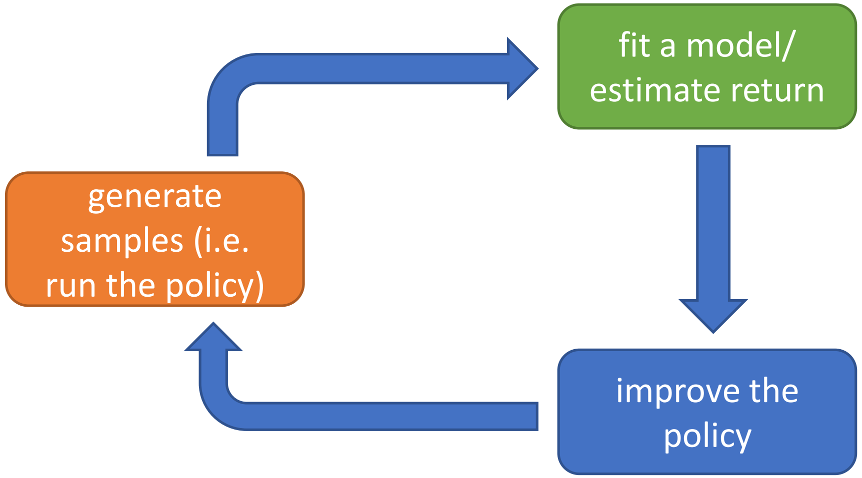
if $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$, then $\mathbf{a}$ is *better than average* (recall that $V^\pi(\mathbf{s}) = E[Q^\pi(\mathbf{s}, \mathbf{a})]$ under $\pi(\mathbf{a}|\mathbf{s})$)

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!

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
- Value functions and Q-functions



Break

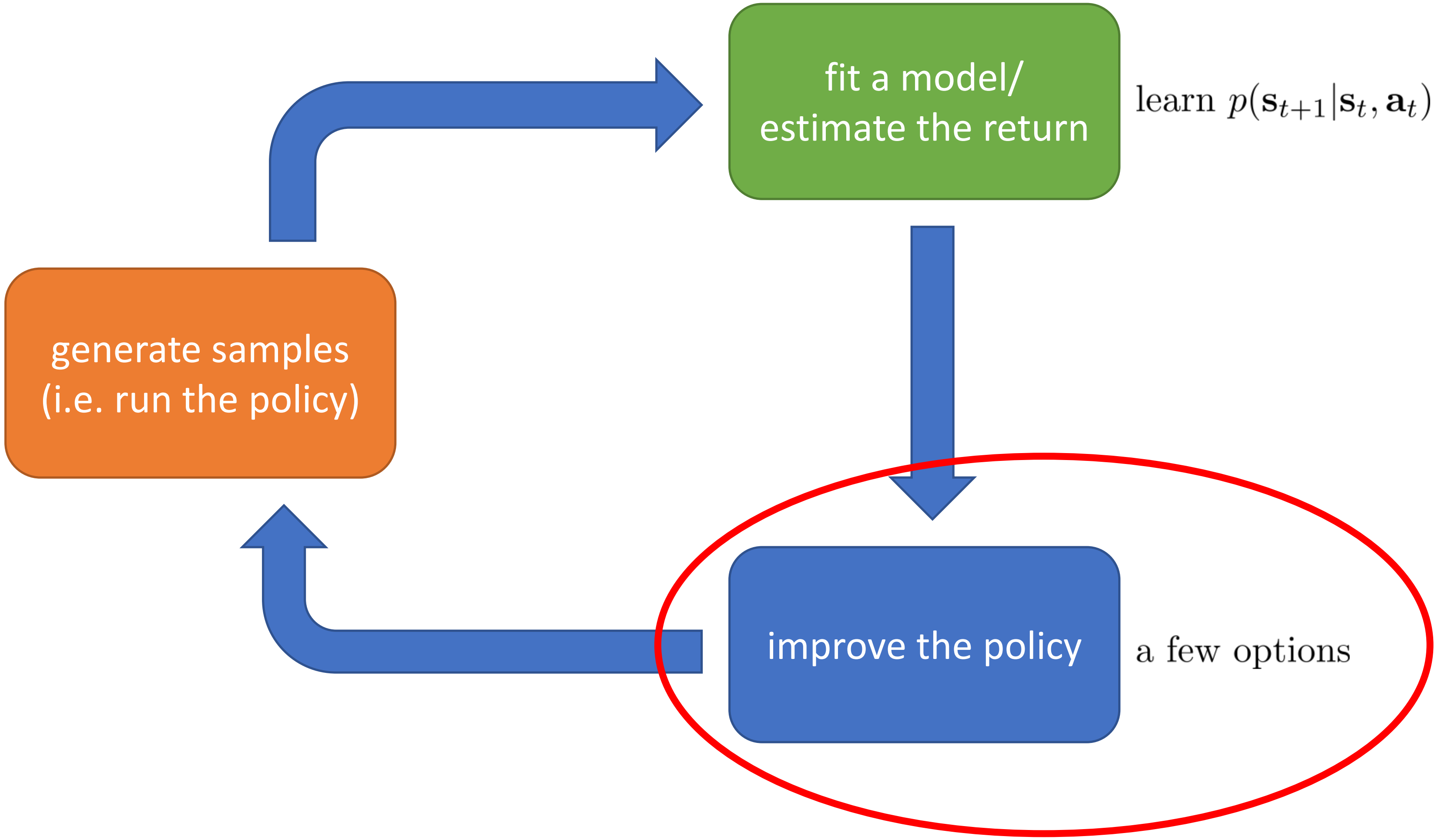
Types of RL algorithms

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

make a sample based approximation to

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy) 用NN表示，然后recover optimal policy with argmax trick
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy 是上面两个的总结，先NN估计Q，然后gradient
- 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

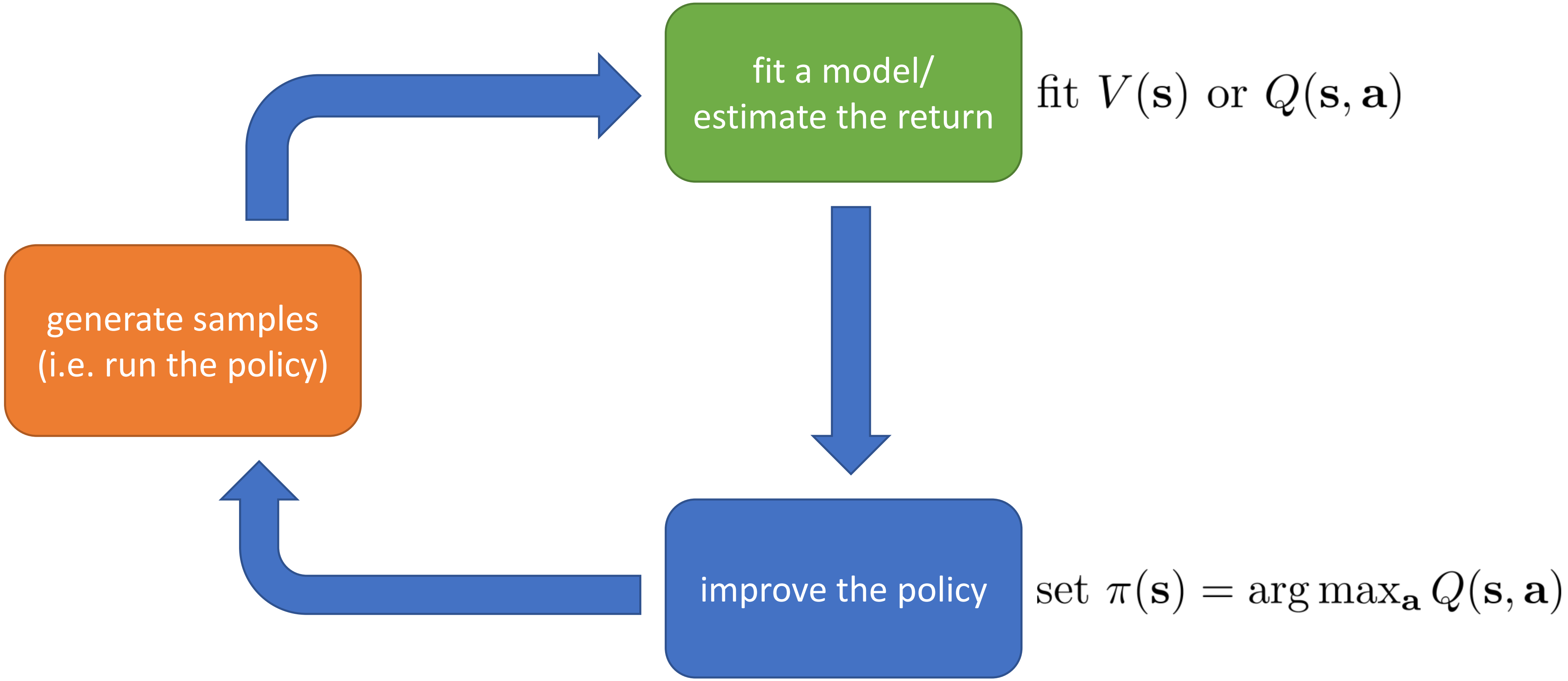
a few options

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

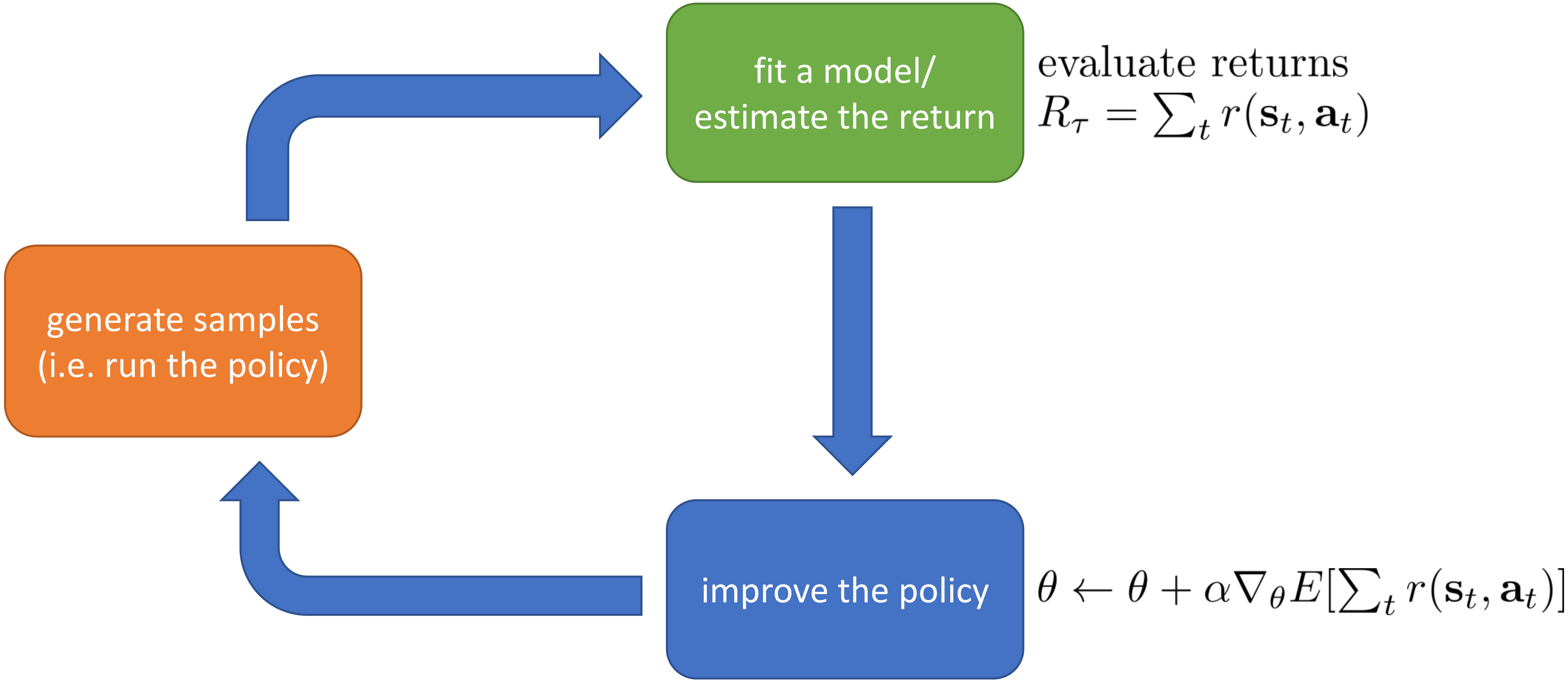
   一般不好搞，还是需要tricks
3. Use the model to learn a value function
   - Dynamic programming
   - Generate simulated experience for model-free learner (Dyna)

   还可以加入其他那三种来产生fake experience

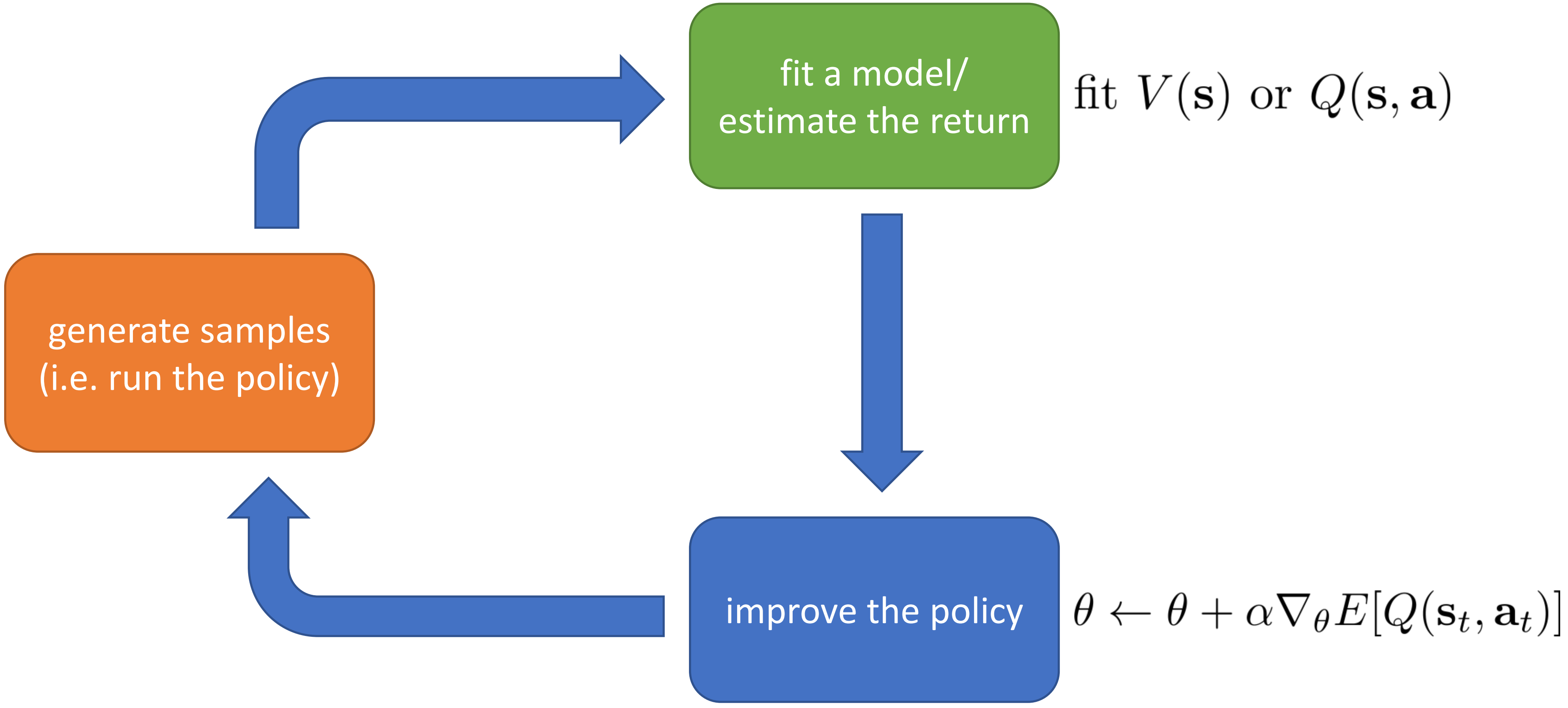
Value function based algorithms



Direct policy gradients



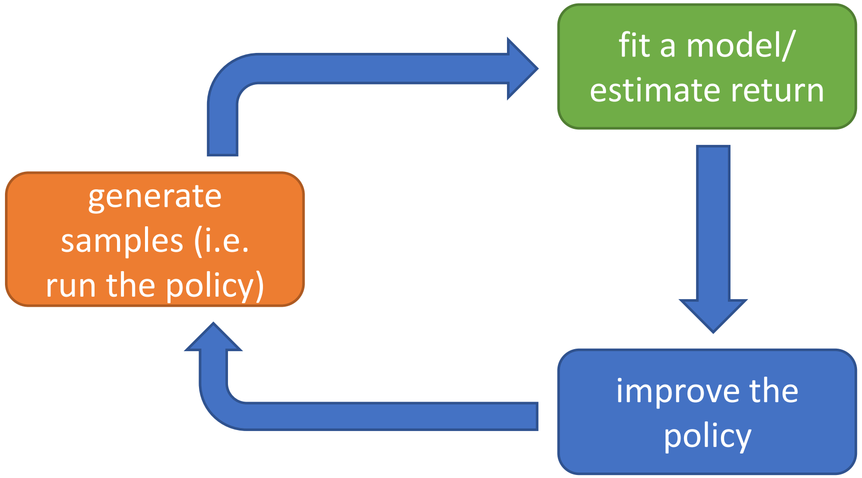
Actor-critic: value functions + policy gradients



Tradeoffs

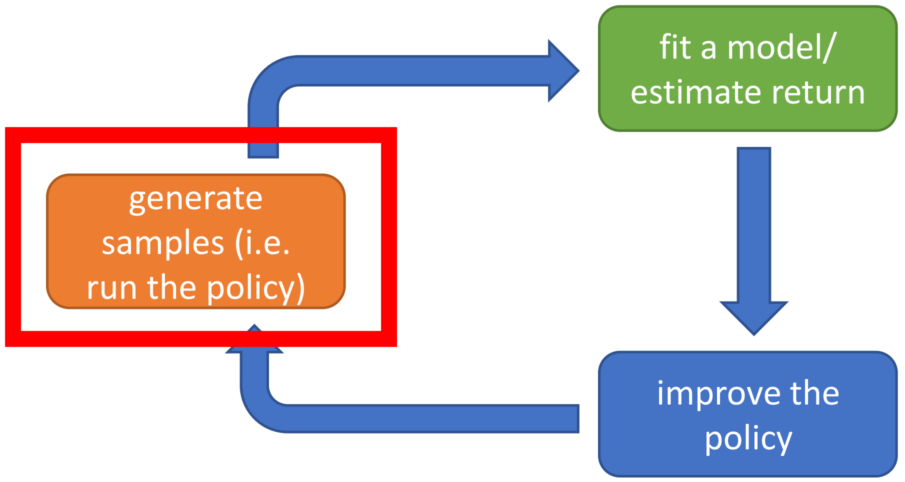
Why so many RL algorithms?

- 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?



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

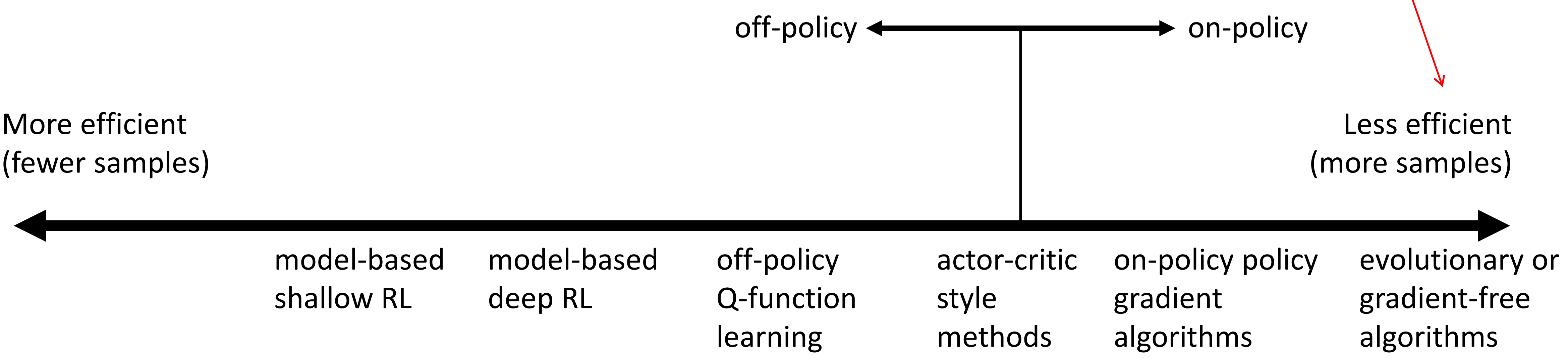


$$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$$

just one gradient step

Comparison: sample efficiency

如果有高速并行模拟器，这类方法其实也无所谓



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!

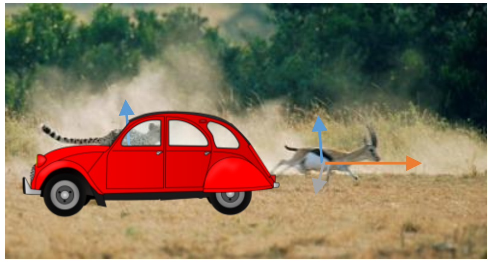
Comparison: stability and ease of use

- 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

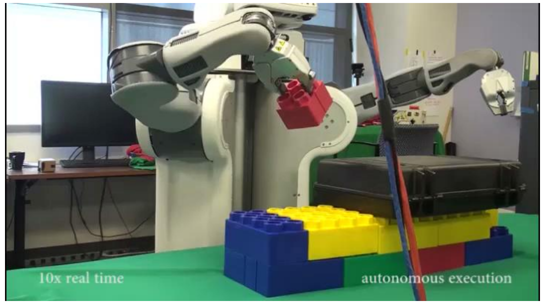
满足马尔可夫

- Common assumption #1: full observability
  - Generally assumed by value function fitting methods
  - Can be mitigated by adding recurrence

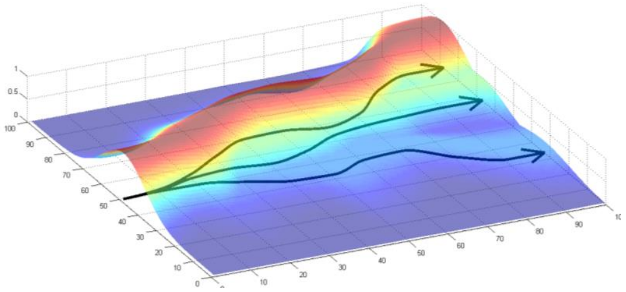


可以不断重来的

- 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

参看李弘毅教授的讲义

- 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

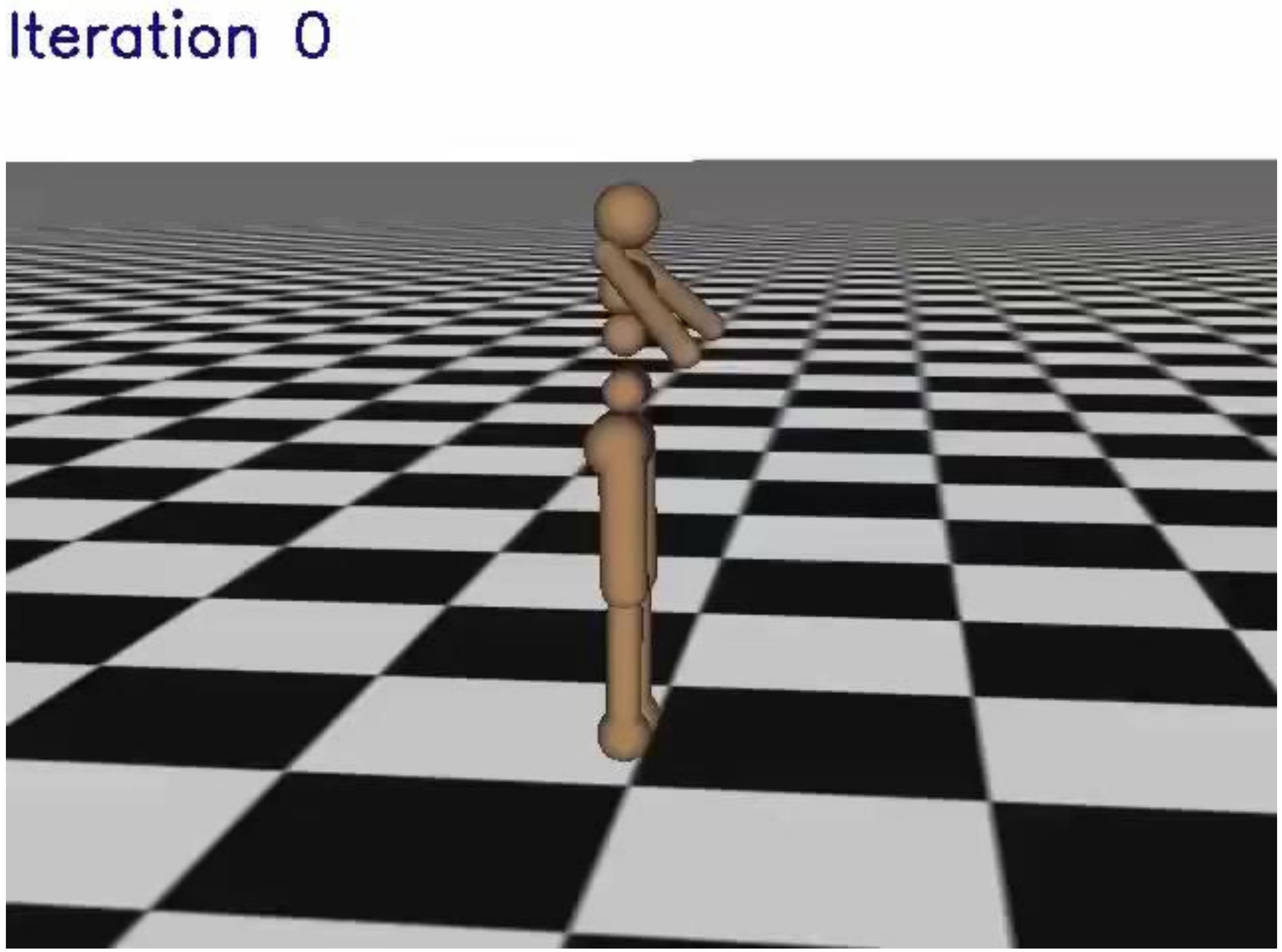
- Playing Atari with deep reinforcement learning, Mnih et al. '13
- Q-learning with convolutional neural networks

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

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