CS 8803
Deep Reinforcement Learning

Lec 4: Policy Gradients
Fall 2024

Animesh Garg

Summary: MDP Equations

- Value iteration equation:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V_k(s') \right]$$
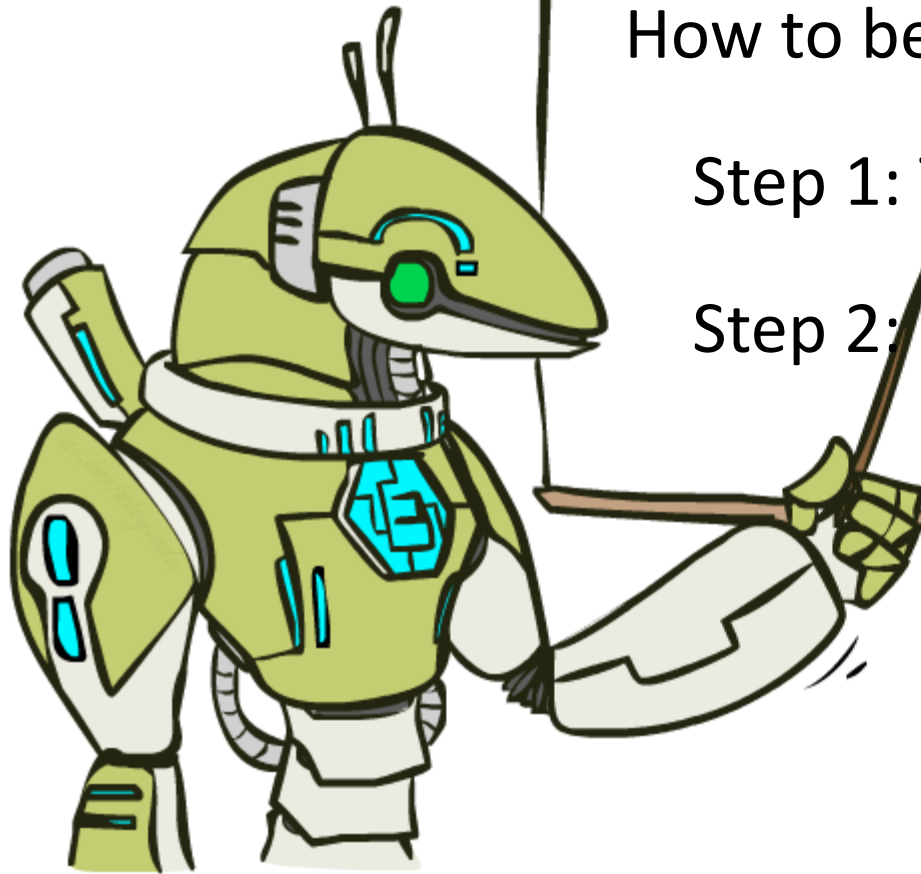
- Policy evaluation equation:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

- Policy iteration equation:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s') \right]$$

The Bellman Equations

How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

Convergence when Solving MDPs

- Redefine value update as general Bellman Utility update
  - Recursive update or utility (sum of discounted reward)

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

$$\downarrow$$

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s, a) U_i(s')$$

- How does this converge?
  - Assume fixed policy $\pi_i(s)$.
  - *R(s)* is the short term reward of being in *s*

Convergence when Solving MDPs

- How does this update rule converge?

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s,a)U_i(s')$$

- Re-write update: $\quad U_{i+1} \leftarrow BU_i \quad\quad v' = Av$

  Bellman Recursion

  - *B* is a linear operator (like a matrix)
  - *U* is a vector

- Interested in delta between Utilities: $\quad\quad \|U_{i+1} - U_i\|$

$$\|BU_{i+1} - BU_i\| \leq \gamma \|U_{i+1} - U_i\|$$

Detour: Convergence of Bellman backup

Let $\|V - V'\| = \max_s |V(s) - V'(s)|$ be the infinity norm

$$\|BV_k - BV_j\| = \left\| \max_a \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V_k(s') \right) - \max_{a'} \left(R(s, a') + \gamma \sum_{s' \in S} P(s'|s, a')V_j(s') \right) \right\|$$

$$\leq \max_a \left\| \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V_k(s') - R(s, a) - \gamma \sum_{s' \in S} P(s'|s, a)V_j(s') \right) \right\|$$

$$= \max_a \left\| \gamma \sum_{s' \in S} P(s'|s, a)(V_k(s') - V_j(s')) \right\|$$

$$\leq \max_a \left\| \gamma \sum_{s' \in S} P(s'|s, a)\|V_k - V_j\| \right\|$$

$$= \max_a \left\| \gamma \|V_k - V_j\| \sum_{s' \in S} P(s'|s, a)) \right\|$$

$$= \gamma \|V_k - V_j\|$$

Note: Even if all inequalities are equalities, this is still a contraction if $\gamma < 1$

Convergence when Solving MDPs

- How does this delta converge?

$$\|BU_{i+1} - BU_i\| \leq \gamma \|U_{i+1} - U_i\|$$

  - Utility error estimate reduced by γ each iteration:

- Total Utilities are bounded,

$$\sum_{i=0}^{\infty} R_{max}\gamma^i \qquad \pm \frac{R_{max}}{(1-\gamma)}$$
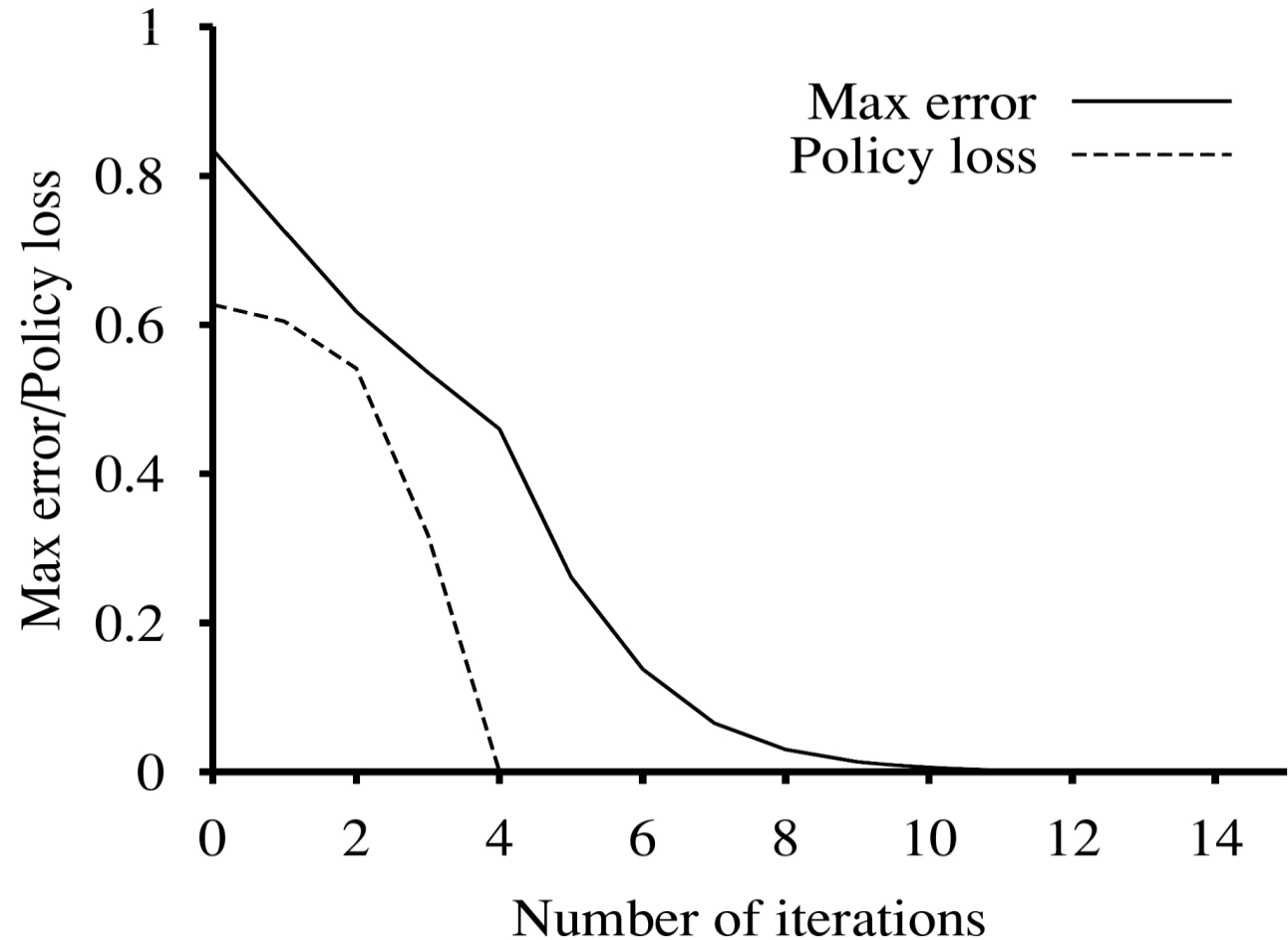
- Consider minimum initial error:
  - (Max norm)
- Max error: reduce by discount each step.

$$\|U_0 - U\| \leq \frac{2R_{max}}{(1-\gamma)}$$

Utility Error Bound

- Error at step 0:

$$\|U_0 - U\| \leq \frac{2R_{max}}{(1 - \gamma)}$$

- Error at step N:

$$\|U_N - U\| = \gamma^N \cdot \frac{2R_{max}}{(1 - \gamma)} < \epsilon$$

- Steps for error below ϵ:

$$N = \frac{log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right)}{log\left(\frac{1}{\gamma}\right)}$$

MDP Convergence Visualized

- Value iteration converges exponentially (with discount factor)

- Policy iteration will converge linearly to 0.

Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions
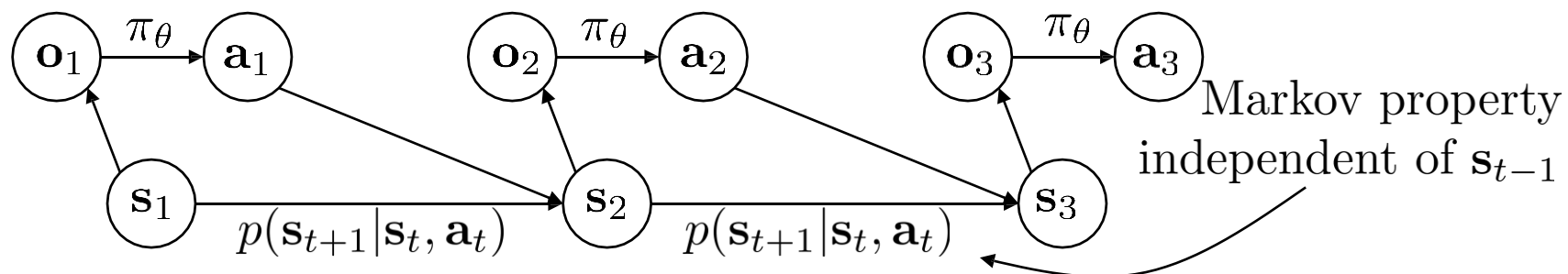
Definitions

Terminology & notation

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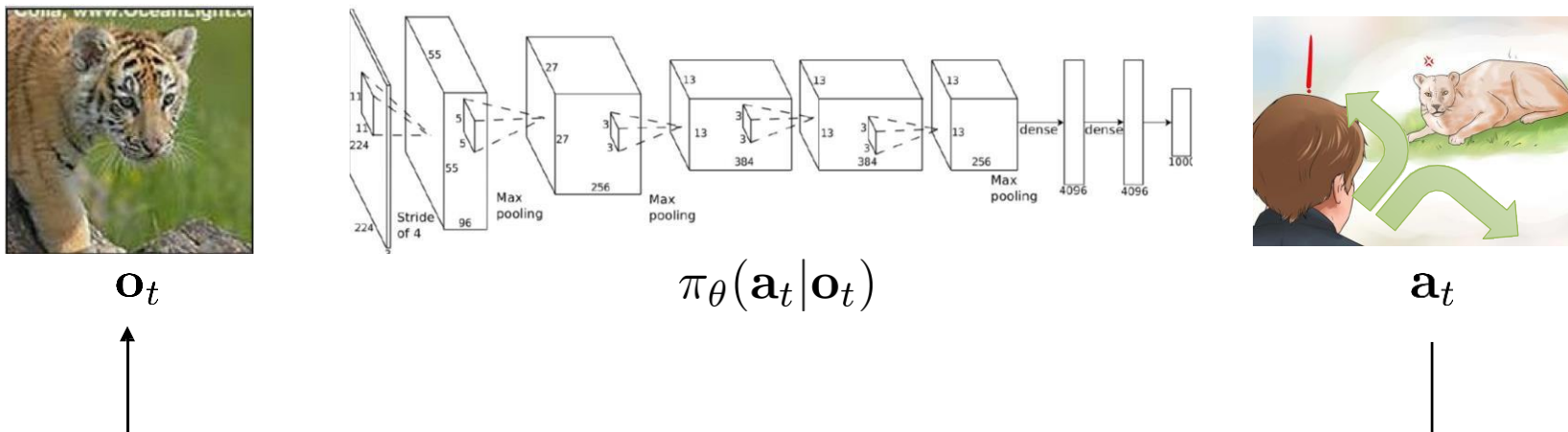
$\mathbf{s}_t$ − state

$\mathbf{o}_t$ − observation

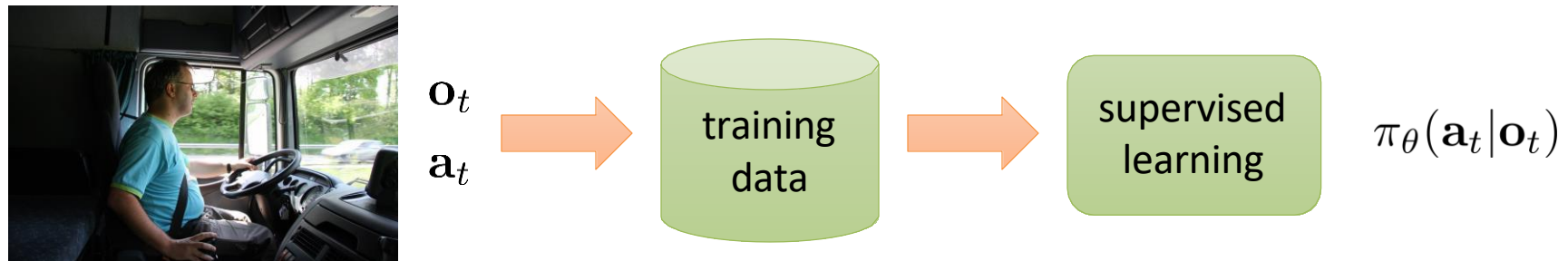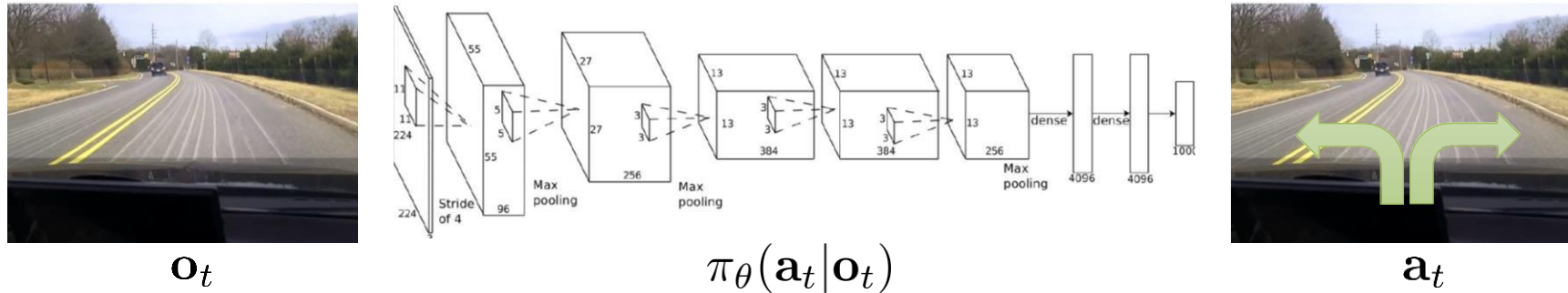$\mathbf{a}_t$ − action

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

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

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

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

$$\mathbf{o}_t \qquad\qquad \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) \qquad\qquad \mathbf{a}_t$$
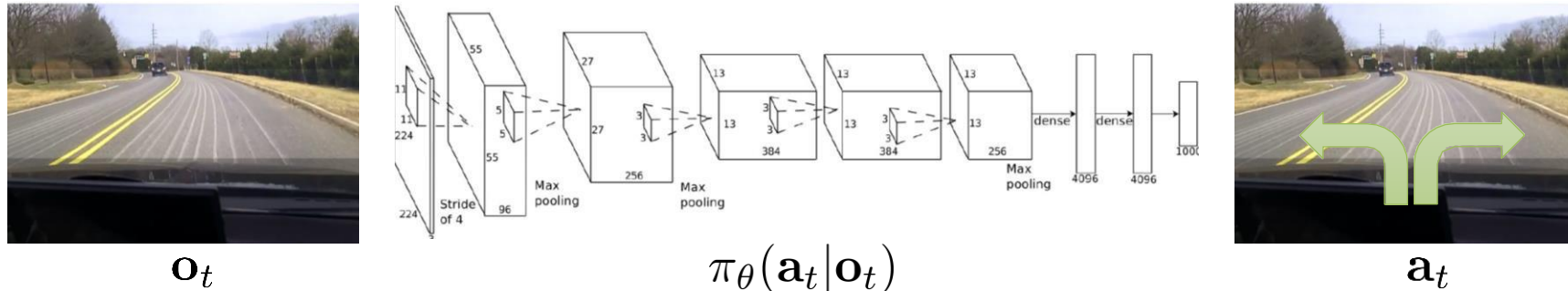
which action is better or worse?

$r(\mathbf{s}, \mathbf{a})$: 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
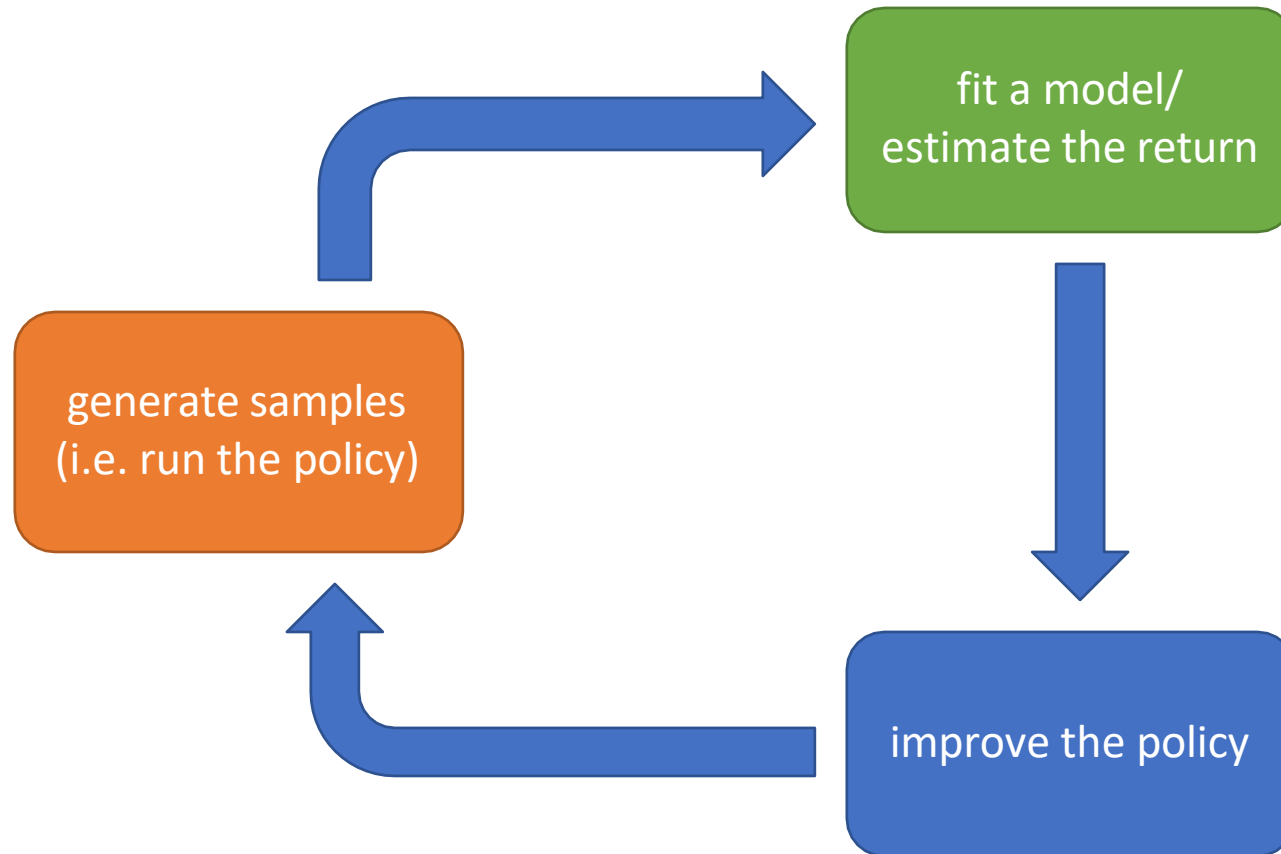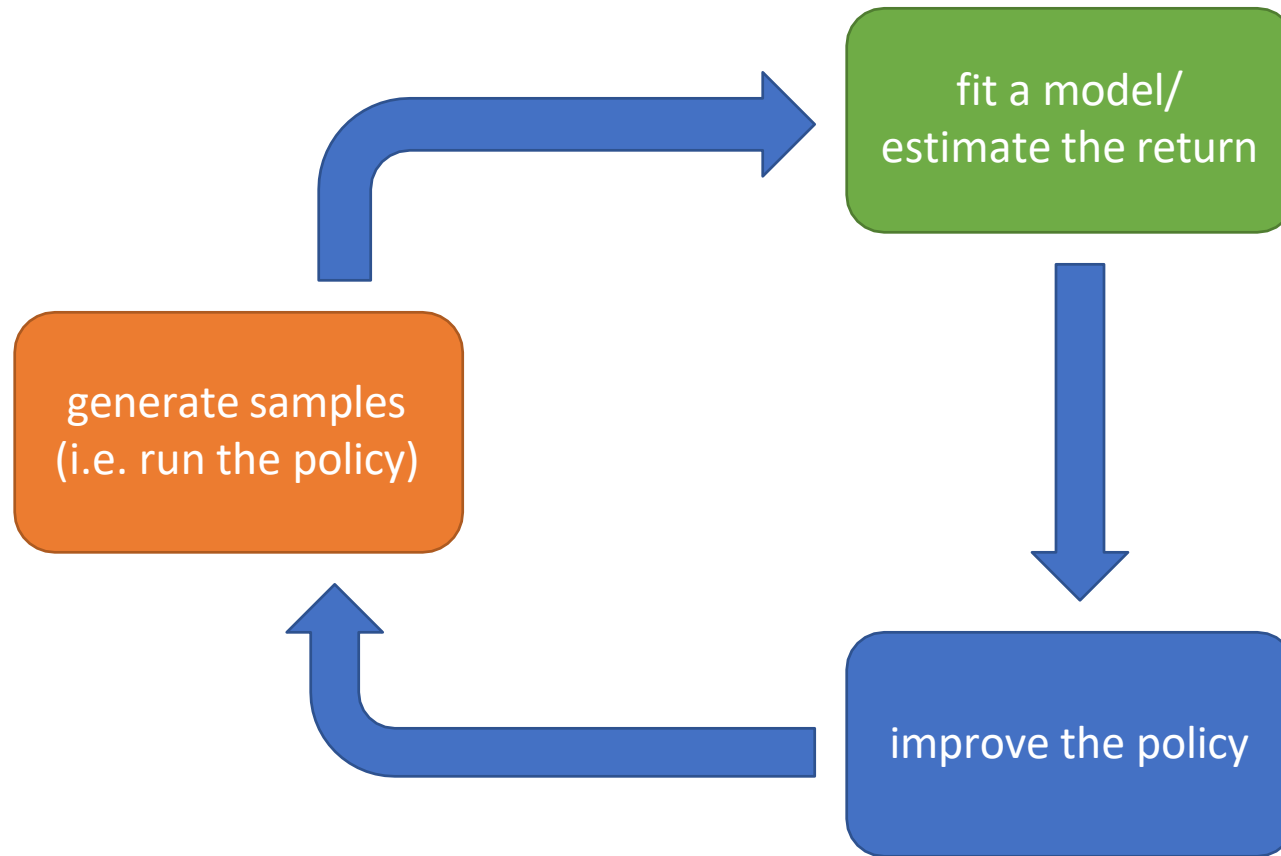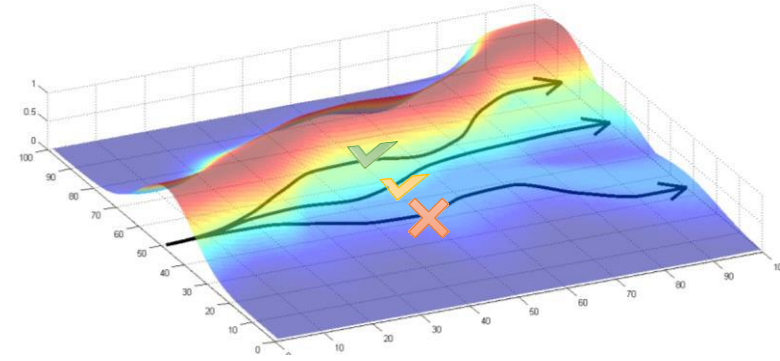Markov decision process

 high reward

 low reward

Algorithms

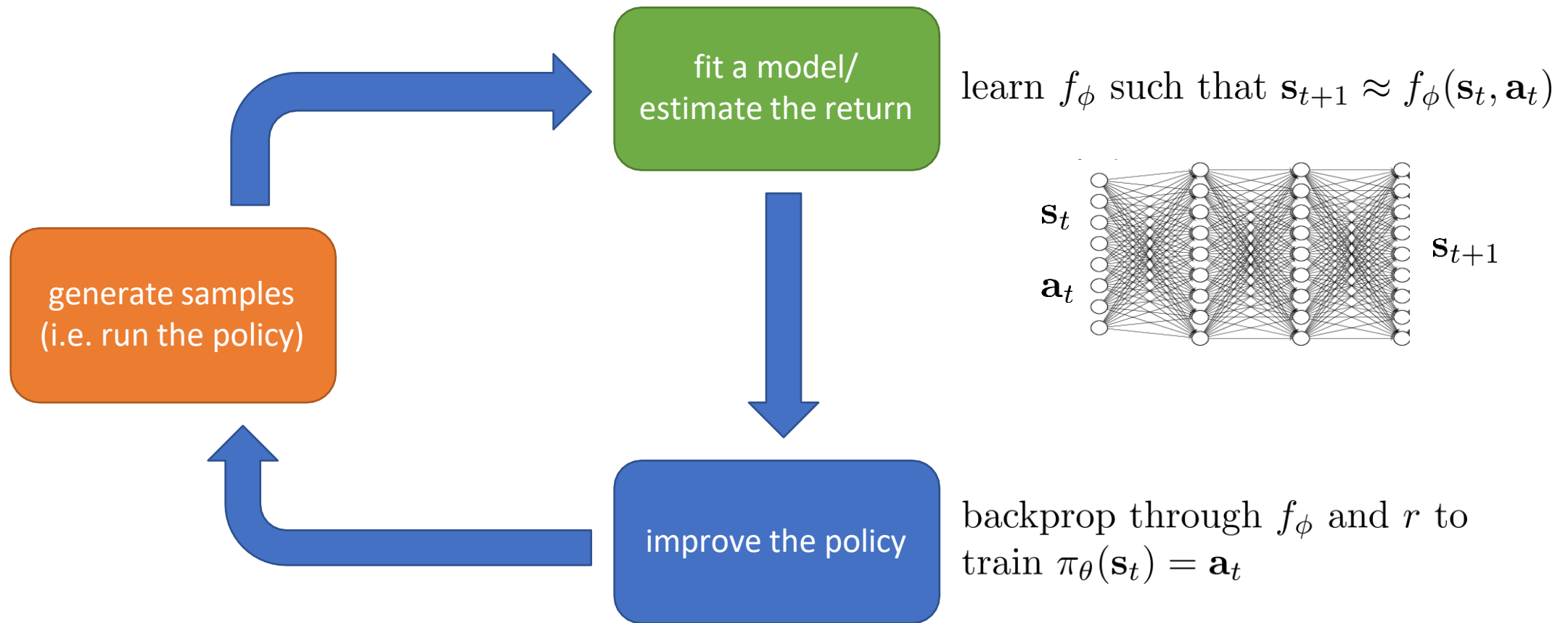# The anatomy of a reinforcement learning algorithm

A simple example

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

fit a model/
estimate the return

generate samples
(i.e. run the policy)

improve the policy

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

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

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

generate samples (i.e. run the policy)

fit a model/ estimate the return

improve the policy

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

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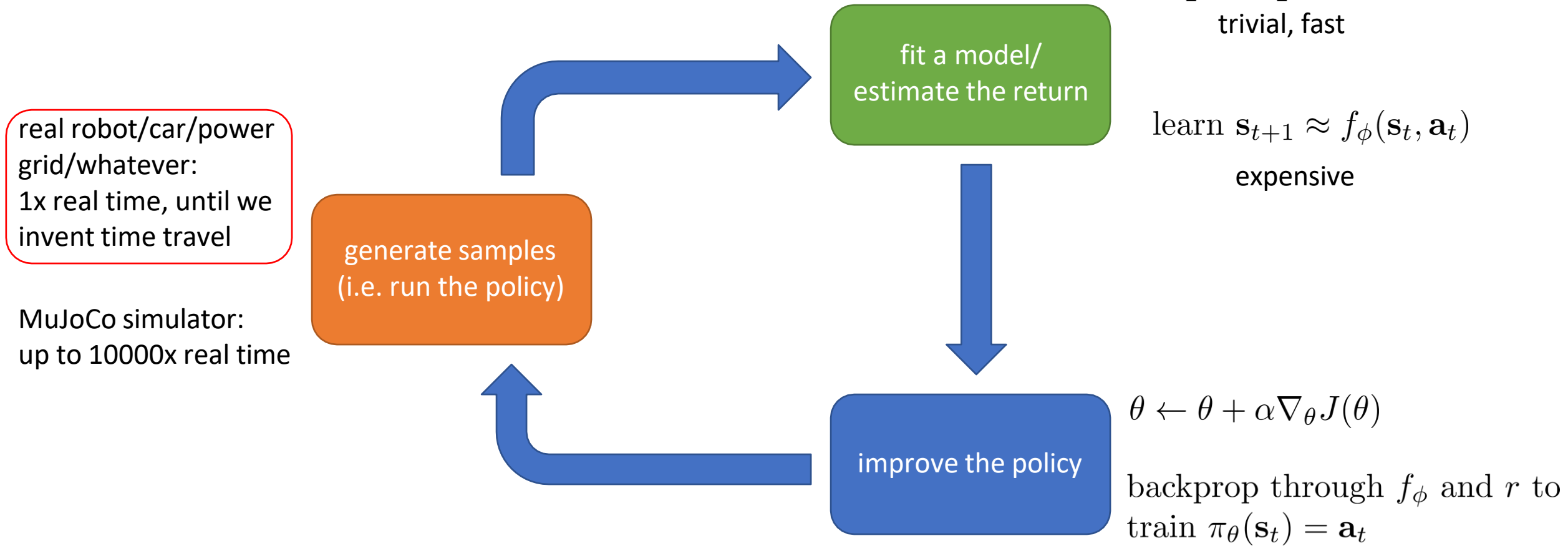
Value Functions

How do we deal with all these expectations?

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

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

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

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

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

$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}\left[V^\pi(\mathbf{s}_1)\right]$ 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)!
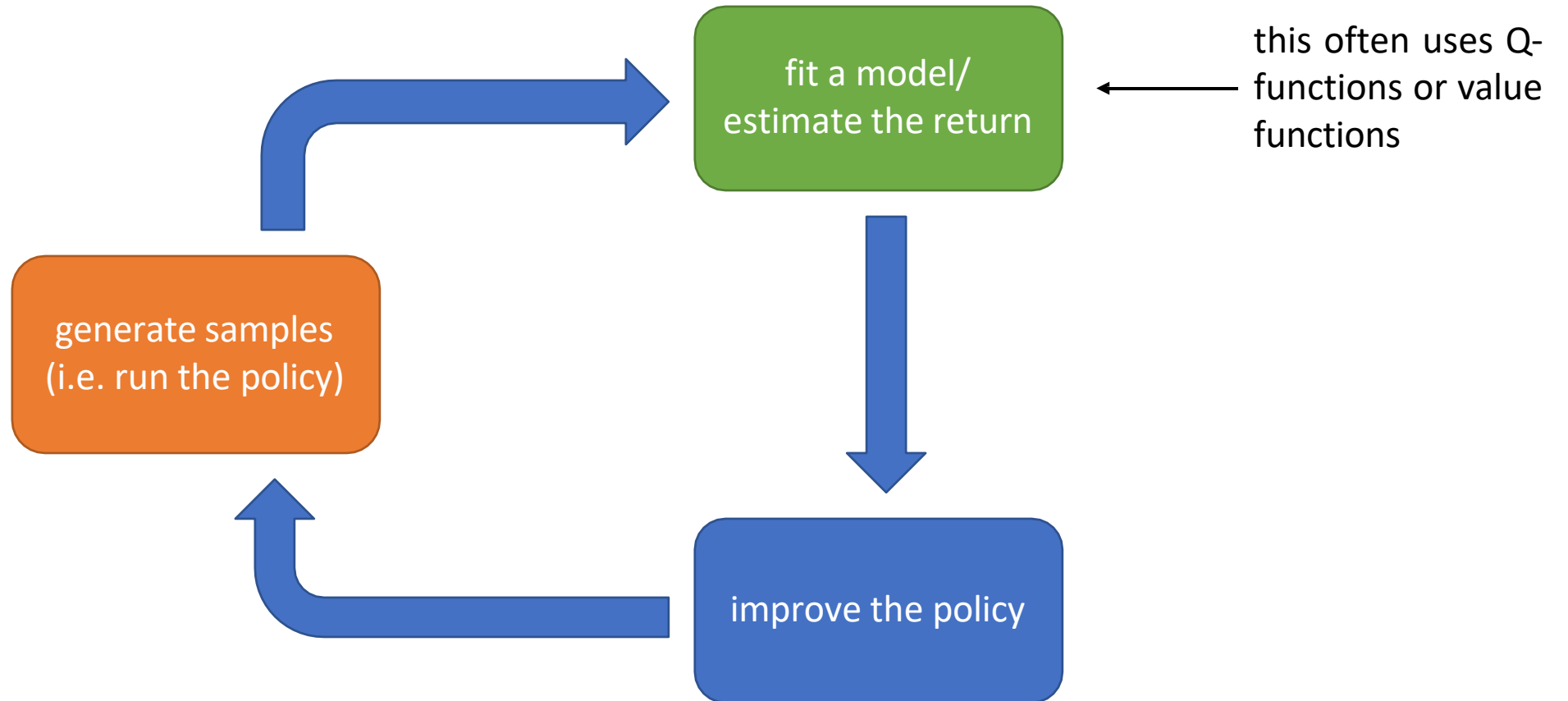
and it doesn't matter what π is

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
- 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
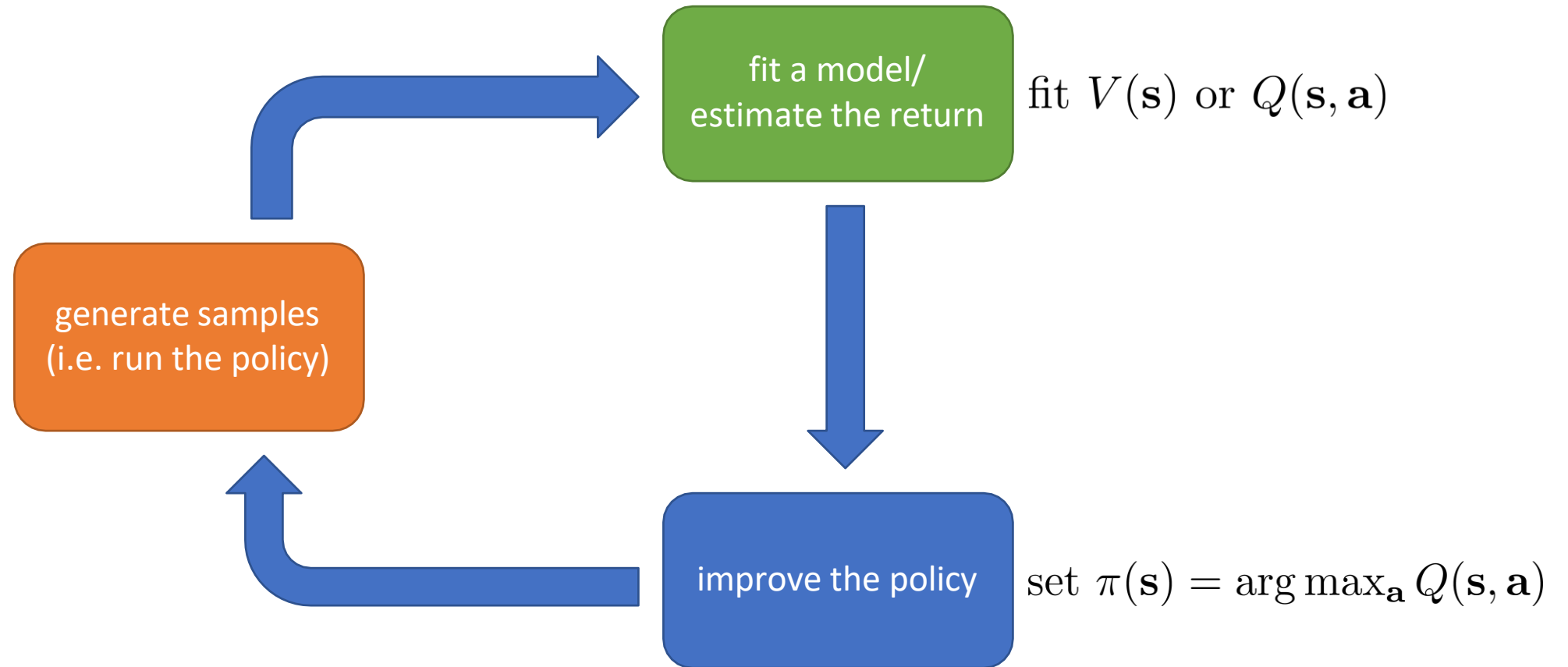
Model-based RL algorithms
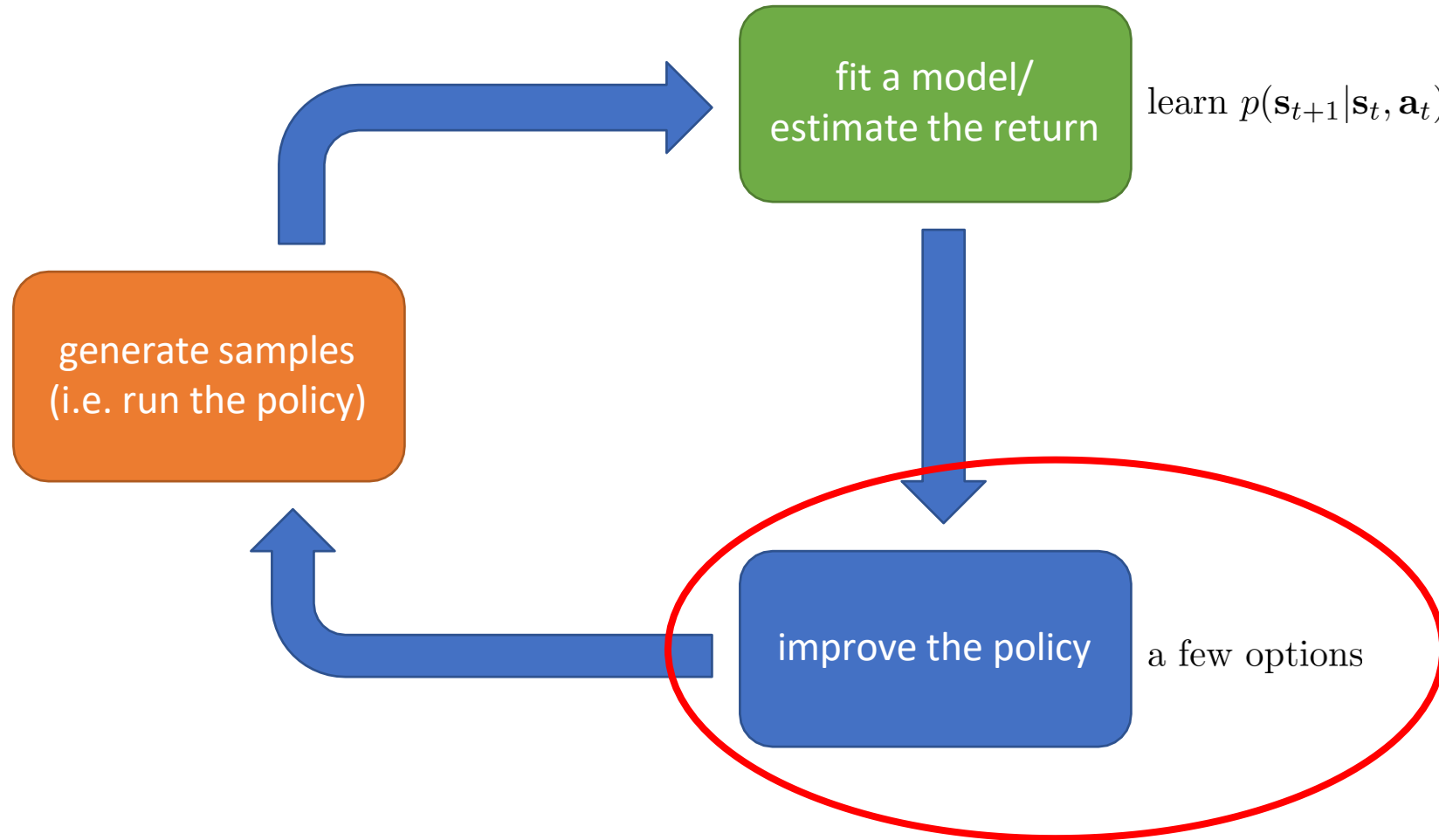
Model-based RL algorithms
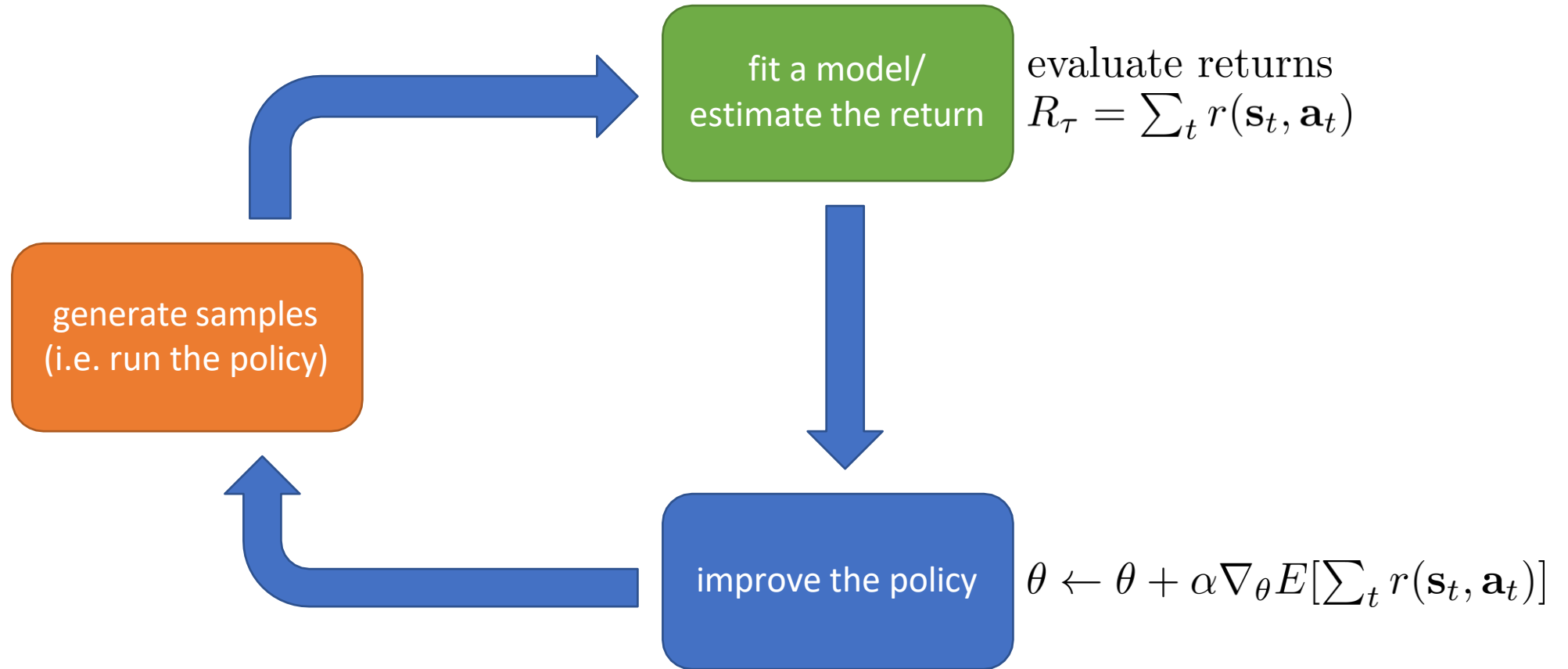
improve the policy   a few options

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
1. Backpropagate gradients into the policy
   - Requires some tricks to make it work
2. Use the model to learn a value function
   - Dynamic programming
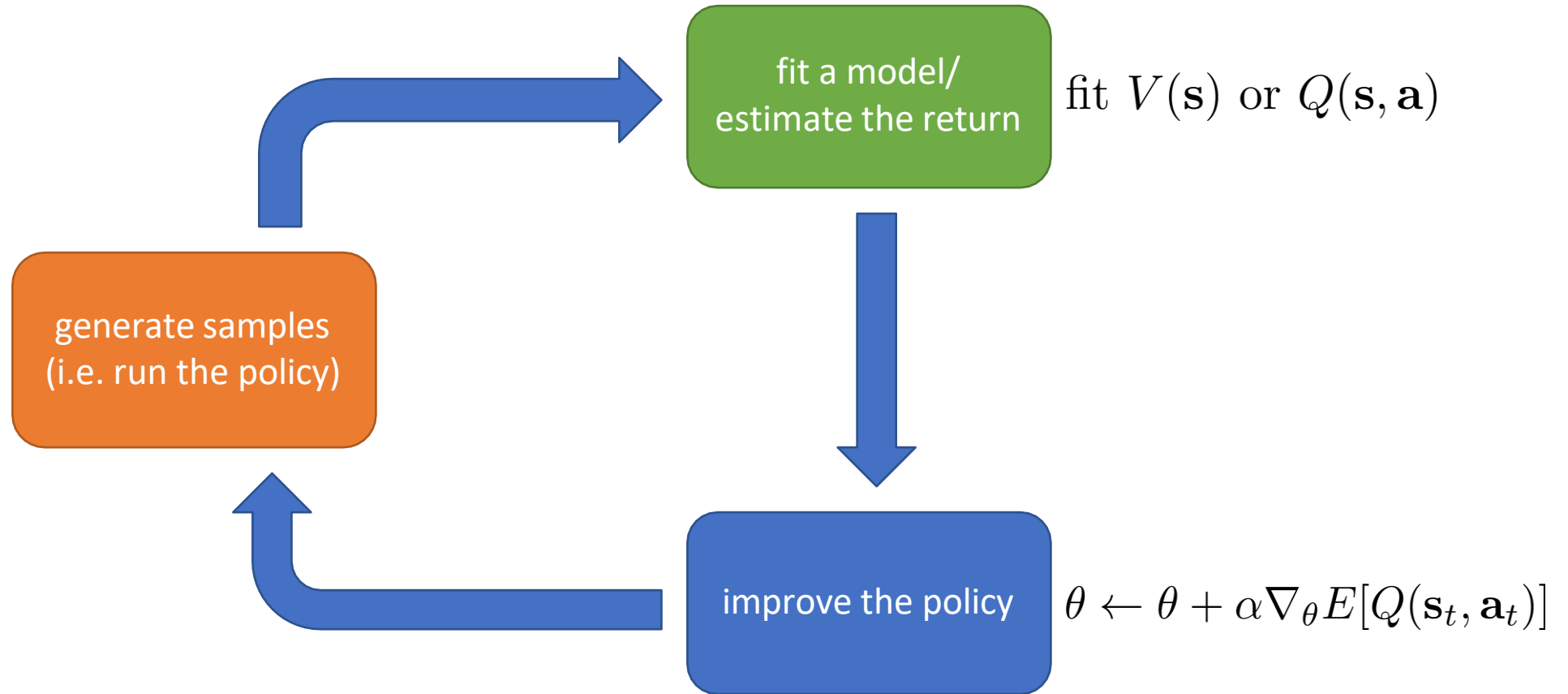   - Generate simulated experience for model-free learner

Value function based algorithms

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

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

fit a model/
estimate the return

generate samples
(i.e. run the policy)

improve the policy

Direct policy gradients

Actor-critic: value functions + policy gradients

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

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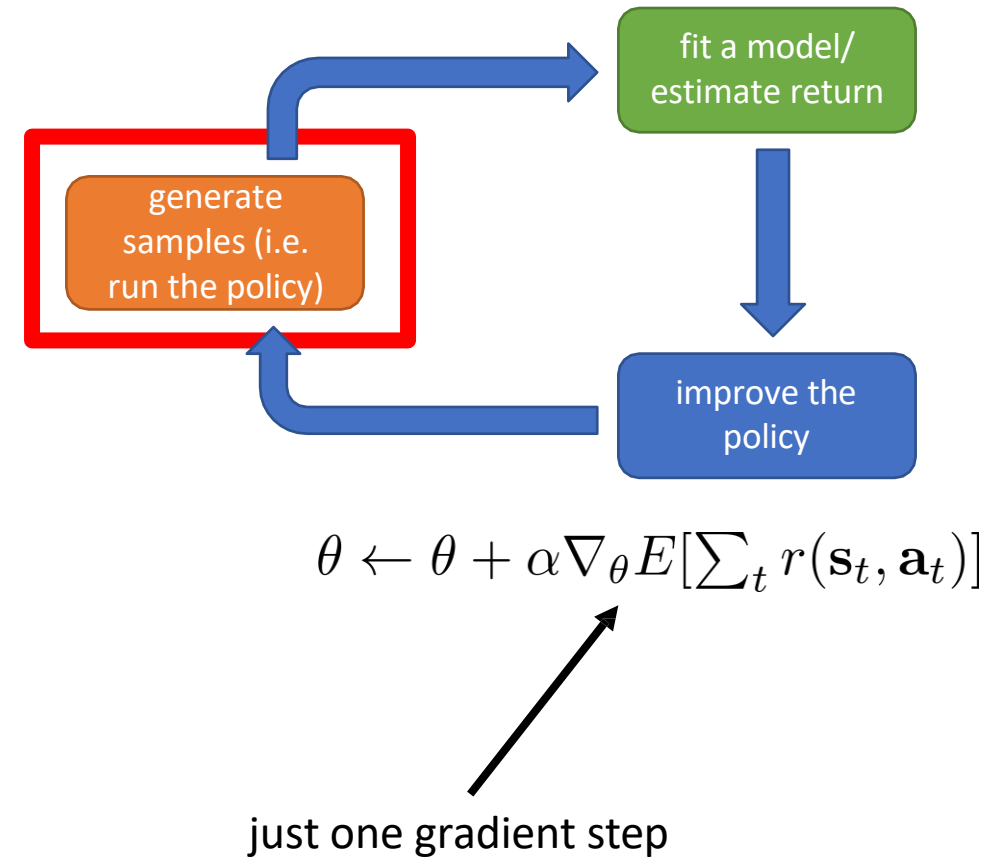
Tradeoffs Between Algorithms

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?
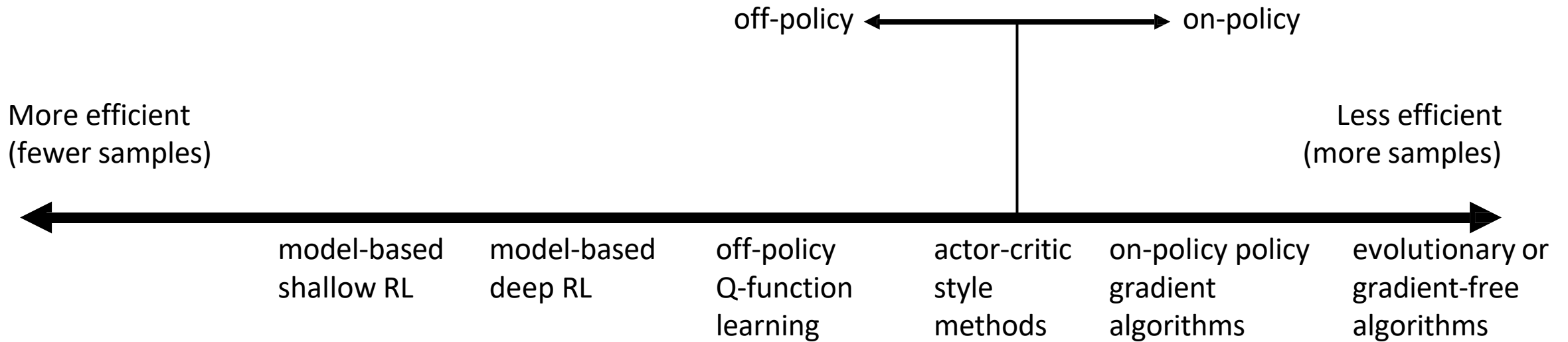 - Easier to represent the model?
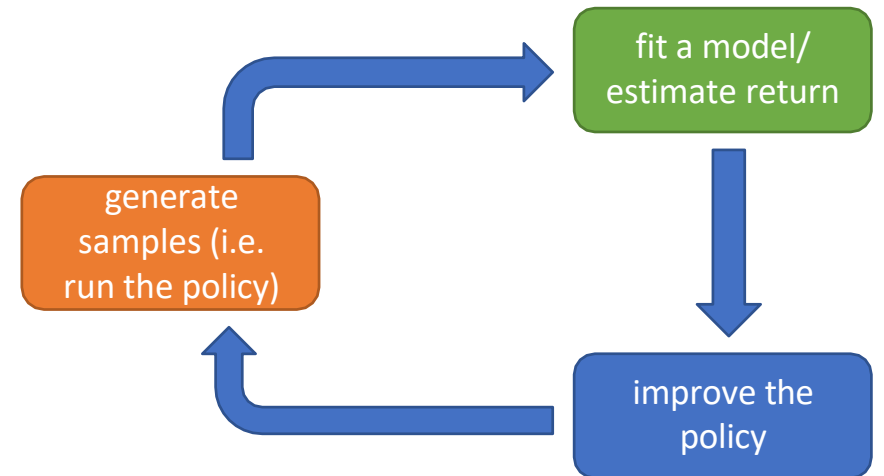
Comparison: sample efficiency

- 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

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!

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 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
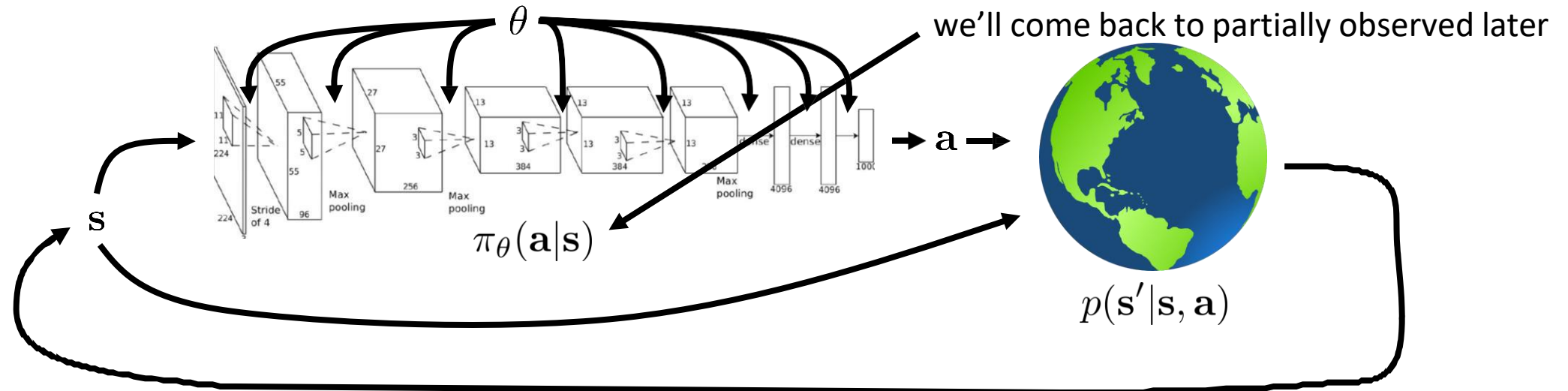 - Often assumed by some model-based RL methods
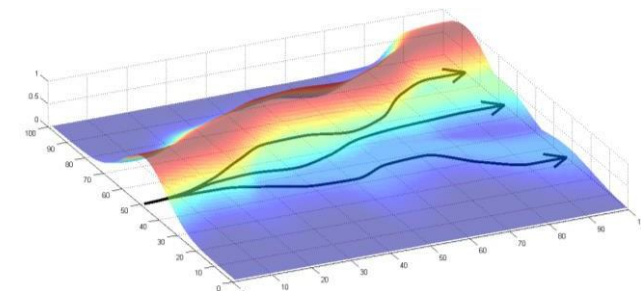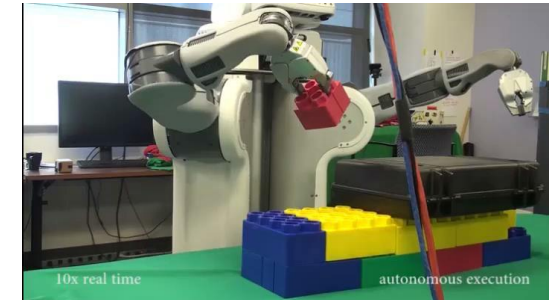
Examples of Algorithms

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!

The goal of reinforcement learning

θ

we'll come back to partially observed later

$\mathbf{s}$

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

$\mathbf{a}$

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

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

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

The goal of reinforcement learning

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

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

infinite horizon case

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

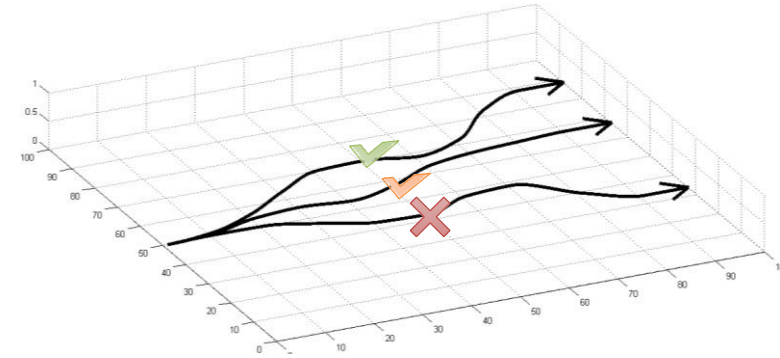finite horizon case

Evaluating the objective

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

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

sum over samples from π_θ

Direct policy differentiation

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

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

a convenient identity

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

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[\underbrace{r(\tau)}_{}] = \int p_\theta(\tau) r(\tau) d\tau$$

$$\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$$

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

Direct policy differentiation

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

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

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

log of both sides $p_\theta(\tau)$

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

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

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

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

Evaluating the policy gradient

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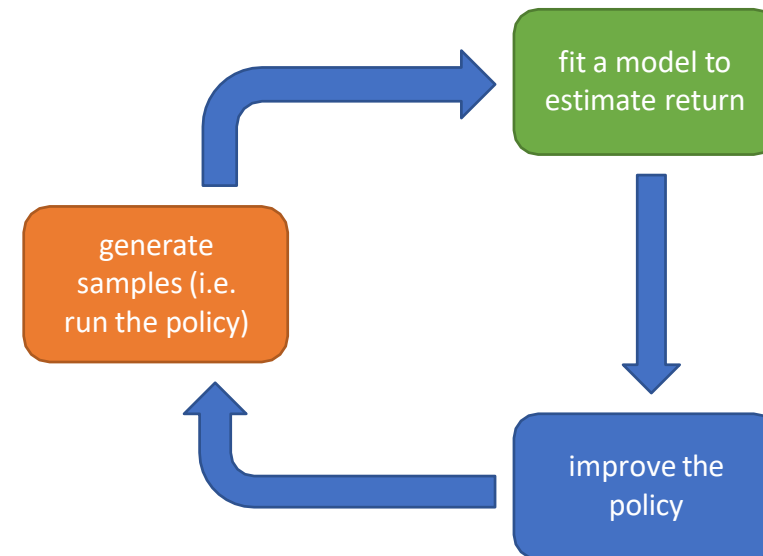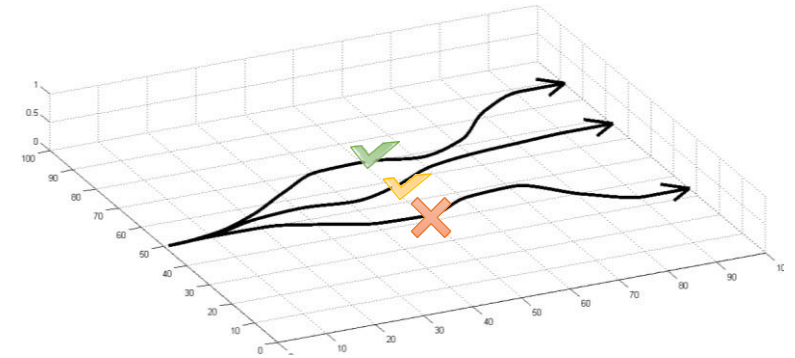
$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

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

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

REINFORCE algorithm:

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

fit a model to estimate return

generate samples (i.e. run the policy)
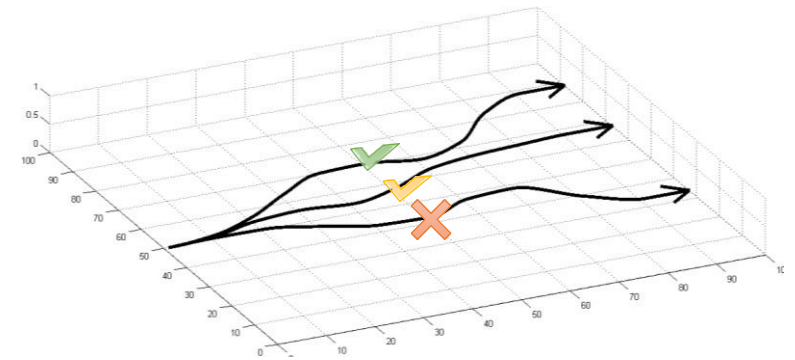
improve the policy

Understanding Policy Gradients

Evaluating the policy gradient

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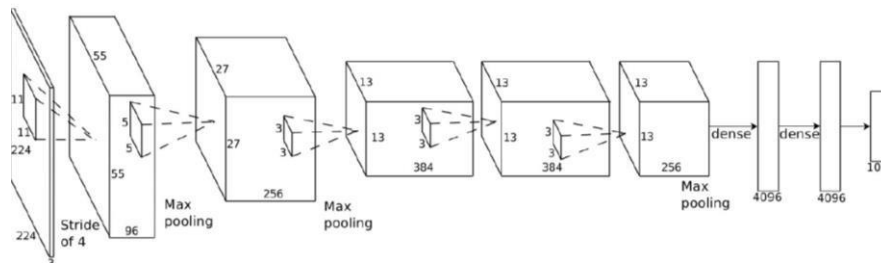
$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

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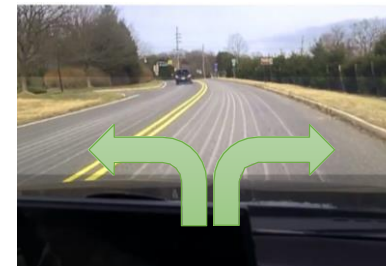
what is this?

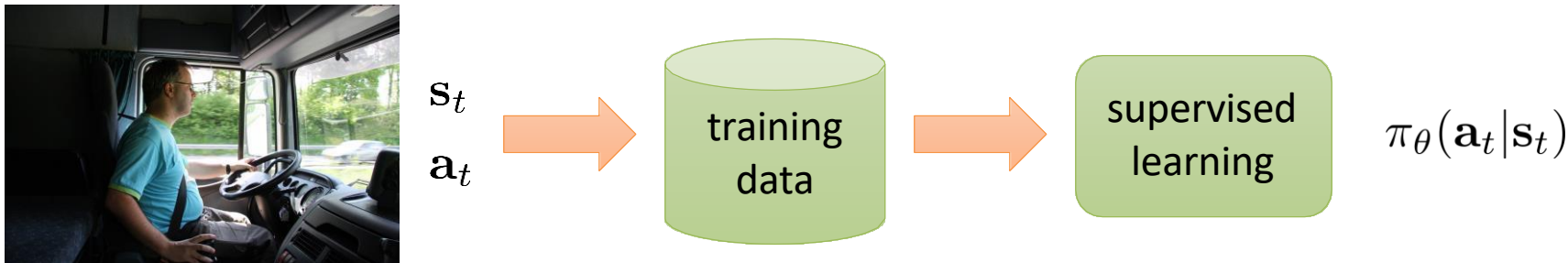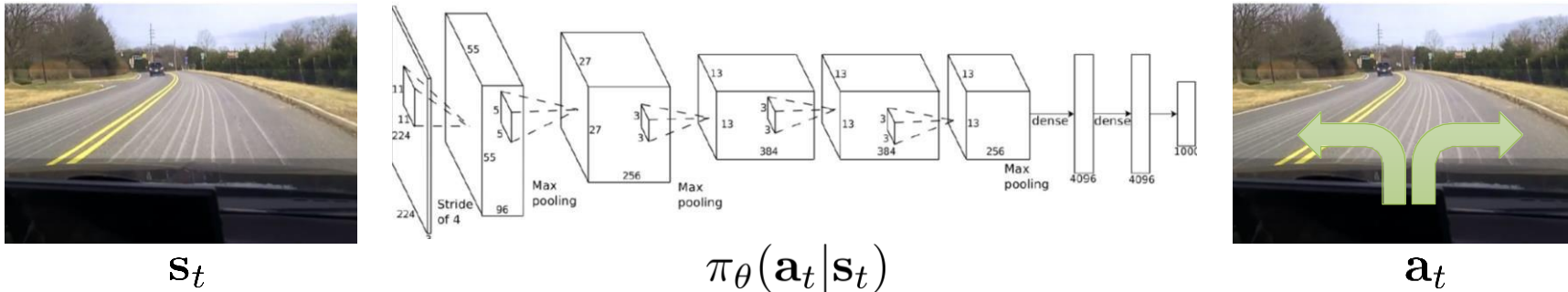$\mathbf{s}_t$

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

$\mathbf{a}_t$

Comparison to maximum likelihood

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

maximum likelihood: $\quad \nabla_\theta J_{\text{ML}}(\theta) \approx \dfrac{1}{N} \sum\limits_{i=1}^{N} \left(\sum\limits_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right)$

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

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

training data

supervised learning

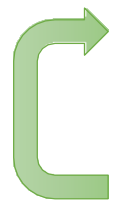$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

Example: Gaussian policies

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

example: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2} \| f(\mathbf{s}_t) - \mathbf{a}_t \|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2} \Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t) \frac{df}{d\theta}$$

REINFORCE algorithm:

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

What did we just do?

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

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T} \nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i) \qquad \text{maximum likelihood:} \quad \nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)$$
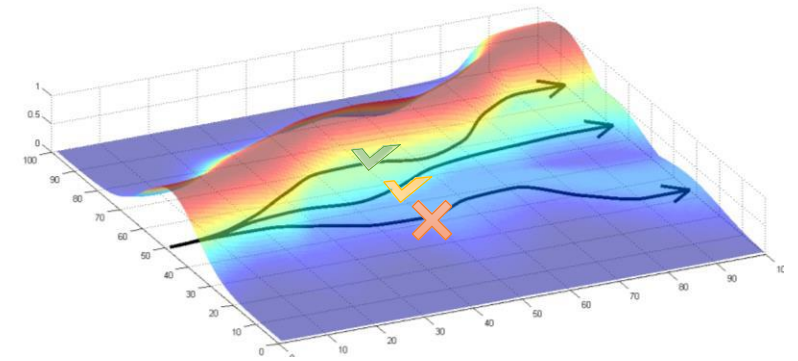
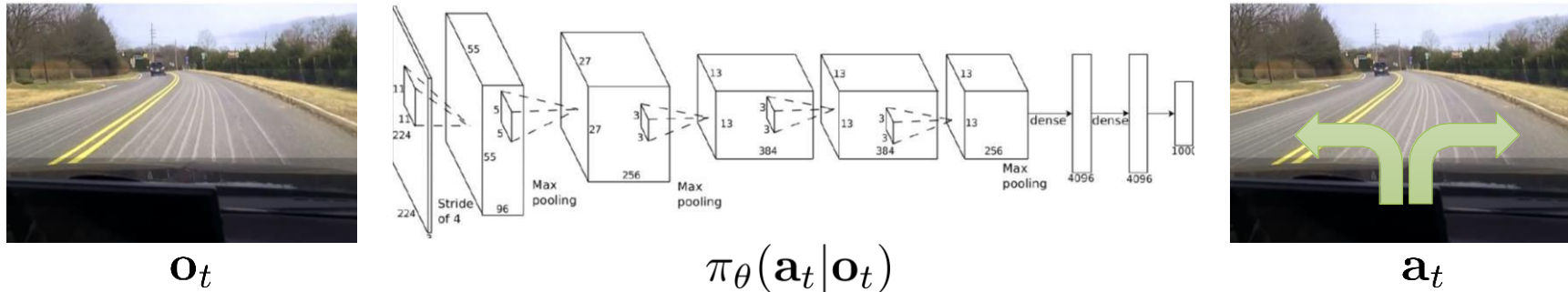good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of "trial and error"!

REINFORCE algorithm:

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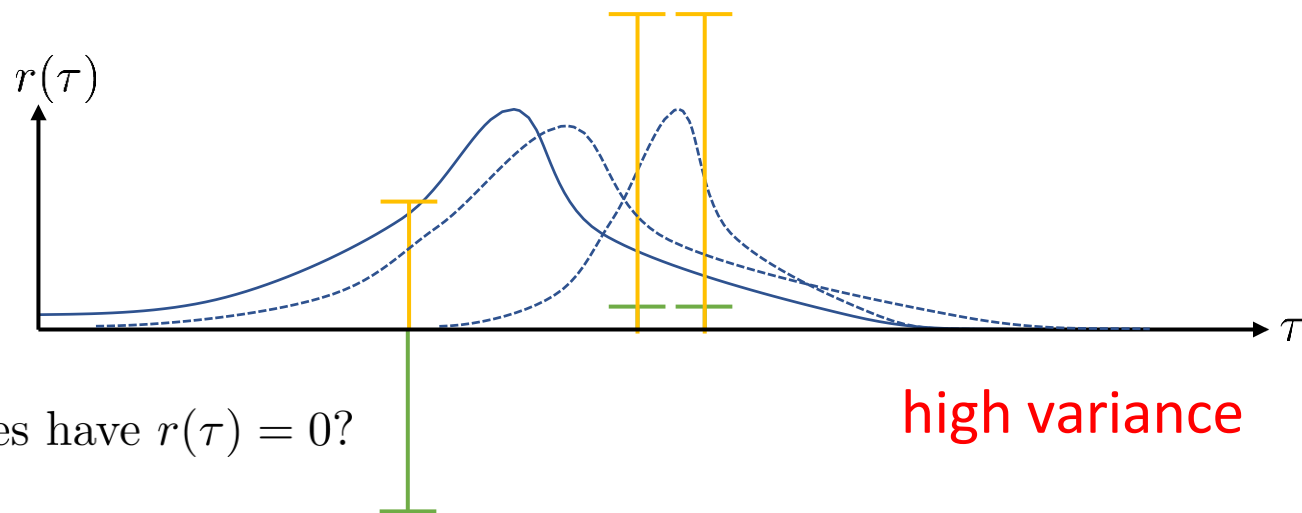3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Partial observability

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

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

Markov property is not actually used!

Can use policy gradient in partially observed MDPs without modification

What is wrong with the policy gradient?

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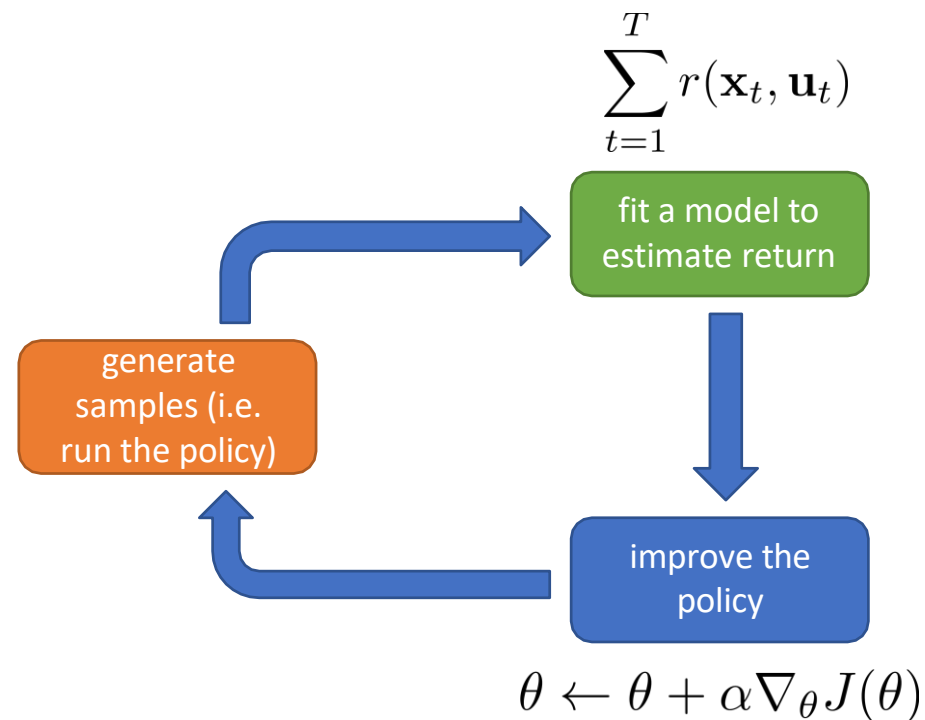
$r(\tau)$

even worse: what if the two "good" samples have $r(\tau) = 0$?

high variance

Review

- Evaluating the RL objective
  - Generate samples
- Evaluating the policy gradient
  - Log-gradient trick
  - Generate samples
- Understanding the policy gradient
  - Formalization of trial-and-error
- Partial observability
  - Works just fine
- What is wrong with policy gradient?

$$\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)$$

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

Reducing Variance

Reducing variance

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

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

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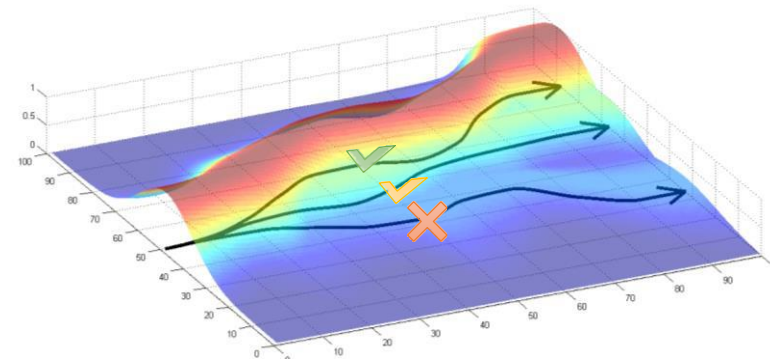
"reward to go"

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

Baselines

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

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

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

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

subtracting a baseline is *unbiased* in expectation!

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

Analyzing variance

can we write down the variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

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

$$\text{Var} = E_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]^2}$$

this bit is just $E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)r(\tau)]$
(baselines are unbiased in expectation)

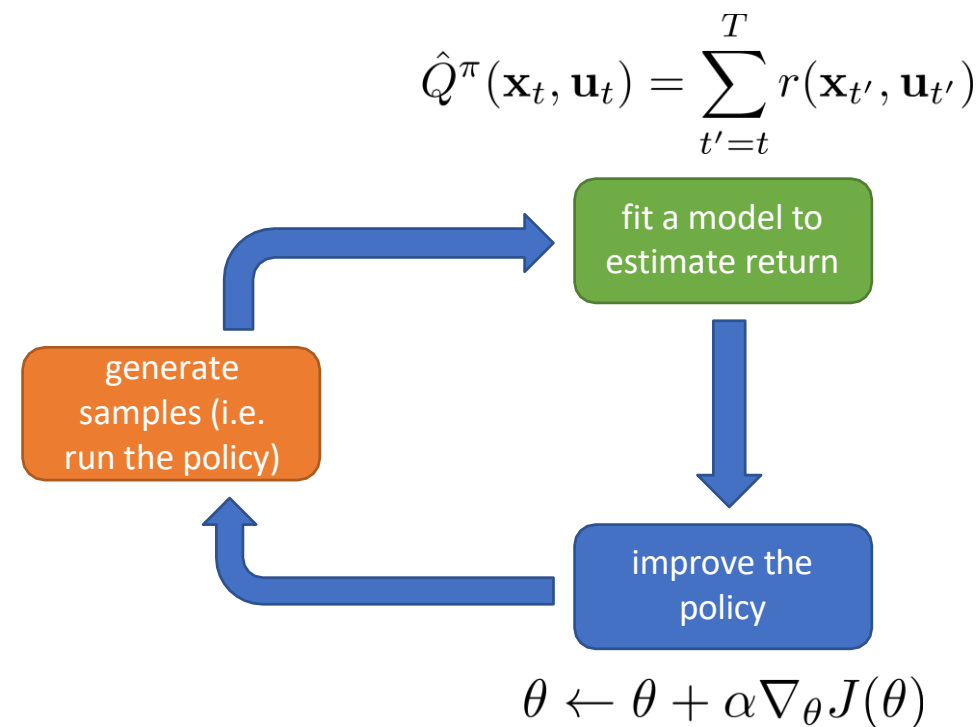$$\frac{d\text{Var}}{db} = \frac{d}{db}E[g(\tau)^2(r(\tau) - b)^2] = \frac{d}{db}\left(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau)b] + b^2 E[g(\tau)^2]\right)$$

$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]} \longleftarrow$$

This is just expected reward, but weighted by gradient magnitudes!

Review

- The high variance of policy gradient

- Exploiting causality
  - Future doesn't affect the past

- Baselines
  - Unbiased!

- Analyzing variance
  - Can derive optimal baselines

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

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$
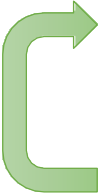
Off-Policy Policy Gradients

Policy gradient is on-policy

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

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

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

this is trouble…

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

can't just skip this!

REINFORCE algorithm:

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

Off-policy learning & importance sampling

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

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

what if we don't have samples from $p_\theta(\tau)$?

(we have samples from some $\bar{p}(\tau)$ instead)

$$J(\theta) = E_{\tau \sim \bar{p}(\tau)}\left[\frac{p_\theta(\tau)}{\bar{p}(\tau)}r(\tau)\right]$$

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

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

importance sampling

$$E_{x \sim p(x)}[f(x)] = \int p(x)f(x)dx$$

$$= \int \frac{q(x)}{q(x)}p(x)f(x)dx$$

$$= \int q(x)\frac{p(x)}{q(x)}f(x)dx$$

$$= E_{x \sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$$

Deriving the policy gradient with IS

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

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

a convenient identity

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

can we estimate the value of some *new* parameters θ'?

$$J(\theta') = E_{\tau \sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)}r(\tau)\right]$$

the only bit that depends on θ'

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)}\left[\frac{\nabla_{\theta'} p_{\theta'}(\tau)}{p_\theta(\tau)}r(\tau)\right] = E_{\tau \sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)}\nabla_{\theta'} \log p_{\theta'}(\tau)r(\tau)\right]$$

now estimate locally, at $\theta = \theta'$: $\quad \nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)r(\tau)]$

The off-policy policy gradient

$$\theta^\star = \arg\max_\theta J(\theta) \qquad\qquad J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau) \right] \qquad \text{when } \theta \neq \theta'$$

$$\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} = \frac{\prod_{t=1}^{T} \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

$$= E_{\tau \sim p_\theta(\tau)} \left[\left(\prod_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \right) \left(\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right] \quad \text{what about causality?}$$

$$= E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \left(\prod_{t''=t}^{t'} \frac{\pi_{\theta'}(\mathbf{a}_{t''}|\mathbf{s}_{t''})}{\pi_\theta(\mathbf{a}_{t''}|\mathbf{s}_{t''})} \right) \right) \right]$$

future actions don't affect current weight

if we ignore this, we get
a policy iteration algorithm
(more on this in a later lecture)

A first-order approximation for IS (preview)

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \underbrace{\left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right)} \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

exponential in T...

let's write the objective a bit differently...

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

$(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \sim \pi_\theta(\mathbf{s}_t, \mathbf{a}_t)$

off-policy policy gradient: $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$

We'll see why this is reasonable later in the course!

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t})} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

ignore this part

Implementing Policy Gradients

Policy gradient with automatic differentiation

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

pretty inefficient to compute these explicitly!

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

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

Just implement "pseudo-loss" as a weighted maximum likelihood:

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

cross entropy (discrete) or squared error (Gaussian)

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Maximum likelihood:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
loss = tf.reduce_mean(negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

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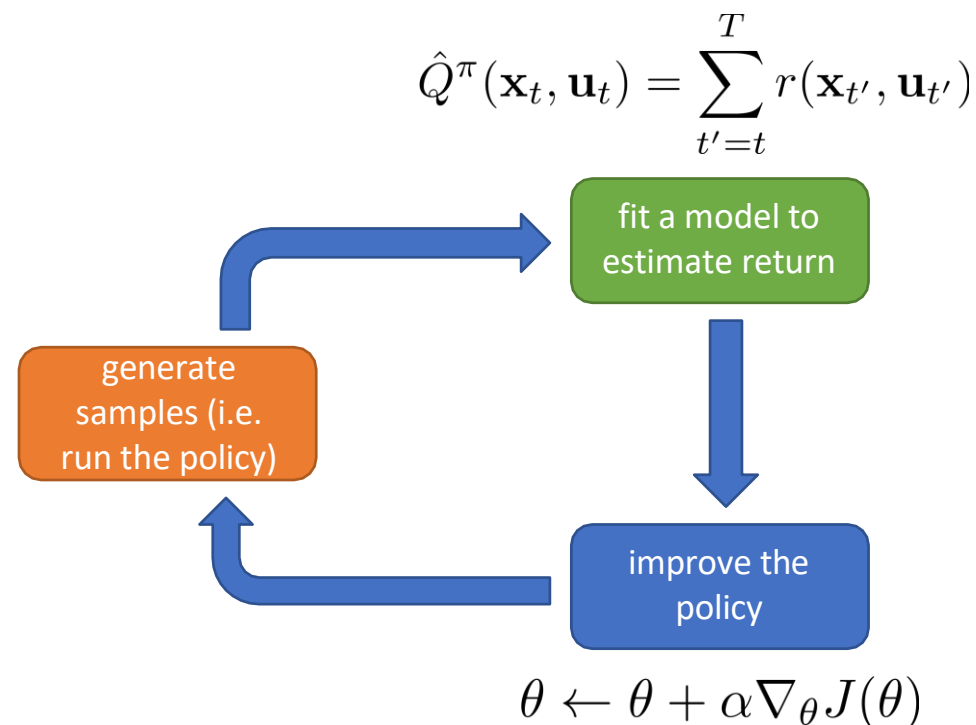
q_values

Policy gradient in practice

- Remember that the gradient has high variance
  - This isn't the same as supervised learning!
  - Gradients will be really noisy!
- Consider using much larger batches
- Tweaking learning rates is very hard
  - Adaptive step size rules like ADAM can be OK-ish
  - We'll learn about policy gradient-specific learning rate adjustment methods later!

Review

- Policy gradient is on-policy
- Can derive off-policy variant
  - Use importance sampling
  - Exponential scaling in T
  - Can ignore state portion (approximation)
- Can implement with automatic differentiation – need to know what to backpropagate
- Practical considerations: batch size, learning rates, optimizers

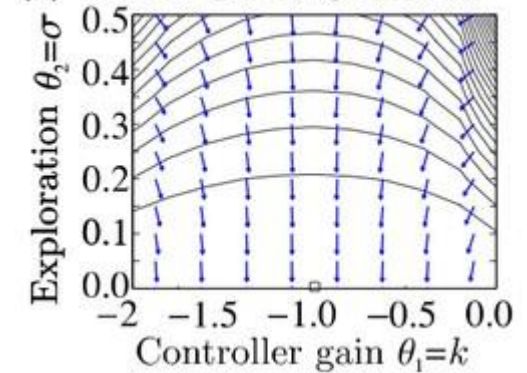$$\hat{Q}^{\pi}(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^{T} r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

Advanced Policy Gradients

What *else* is wrong with the policy gradient?
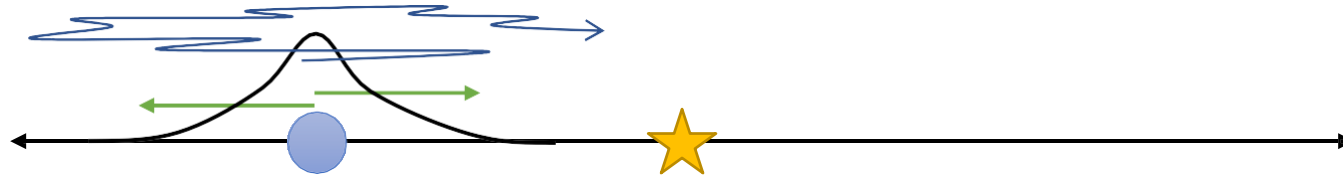
$$r(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^2 - \mathbf{a}_t^2$$

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

(a)'Vanilla' policy gradients

Exploration $\theta_2 = \sigma$

Controller gain $\theta_1 = k$

(image from Peters & Schaal 2008)

Essentially the same
problem as this:
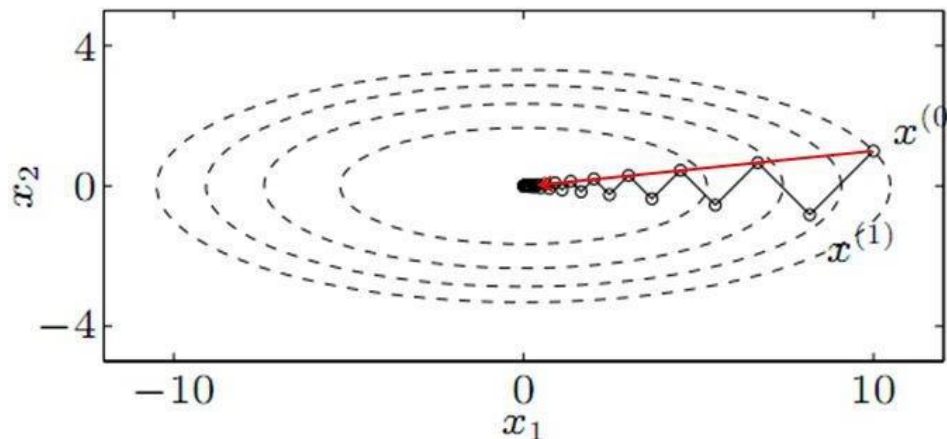
Covariant/natural policy gradient

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \qquad\qquad \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$$

some parameters change probabilities a lot more than others!

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \text{ s.t. } \underline{\|\theta' - \theta\|^2 \leq \epsilon}$$

controls how far we go

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

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

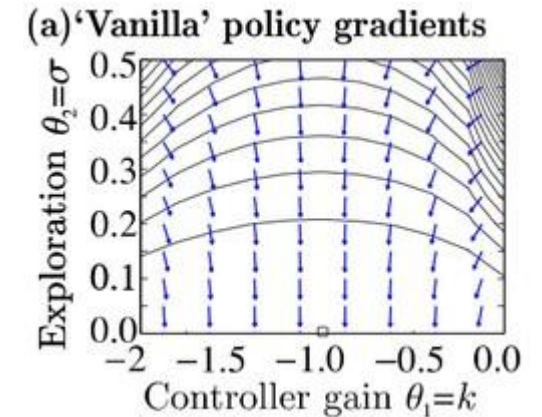parameterization-independent divergence measure

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

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

Fisher-information matrix   can estimate with samples

(a) 'Vanilla' policy gradients

Covariant/natural policy gradient

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

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

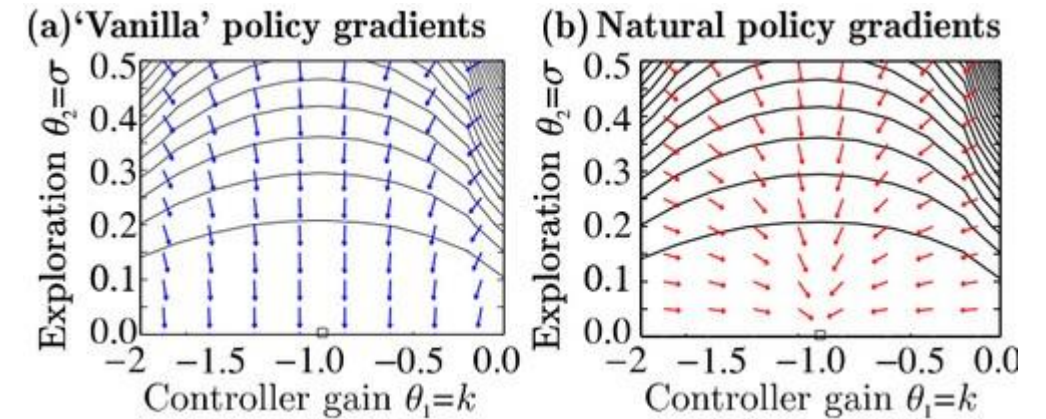$$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$$

natural gradient: pick α

trust region policy optimization: pick ϵ

can solve for optimal α while solving $\mathbf{F}^{-1} \nabla_\theta J(\theta)$

conjugate gradient works well for this

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

(figure from Peters & Schaal 2008)
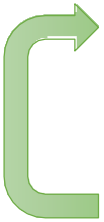
Advanced policy gradient topics

- What more is there?
- Next time: introduce value functions and Q-functions
- Later in the class: more on natural gradient and automatic step size adjustment

Policy gradients suggested readings

- Classic papers
  - Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm
  - Baxter & Bartlett (2001). Infinite-horizon policy-gradient estimation: temporally decomposed policy gradient (not the first paper on this! see actor-critic section later)
  - Peters & Schaal (2008). Reinforcement learning of motor skills with policy gradients: very accessible overview of optimal baselines and natural gradient

- Deep reinforcement learning policy gradient papers
  - Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient (unrelated to later discussion of guided policy search)
  - Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
  - Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient
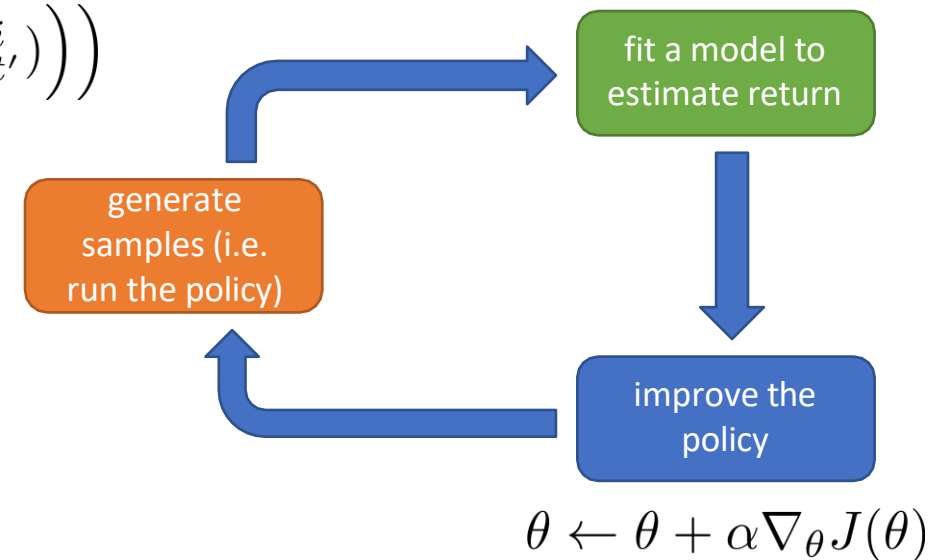
Recap: policy gradients

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i) \right) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

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

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

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

"reward to go"

Improving the policy gradient

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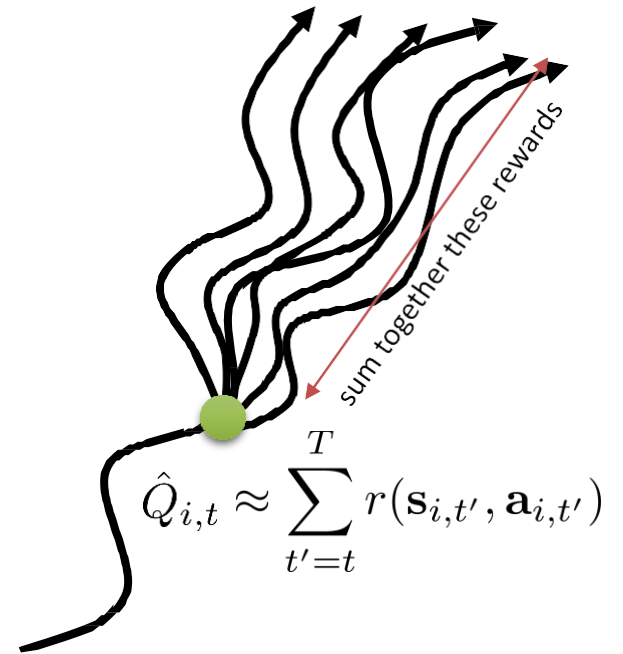
"reward to go"

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

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

can we get a better estimate?

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

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

sum together these rewards

$$\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$
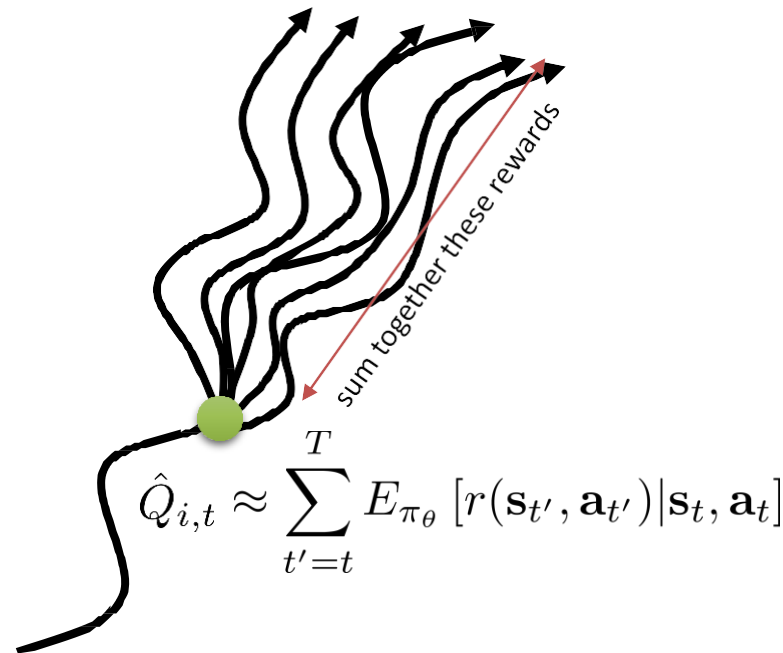
What about the baseline?

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

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

$b_t = \dfrac{1}{N}\sum_{i} Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$ average what?

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

sum together these rewards

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

State & state-action value functions

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

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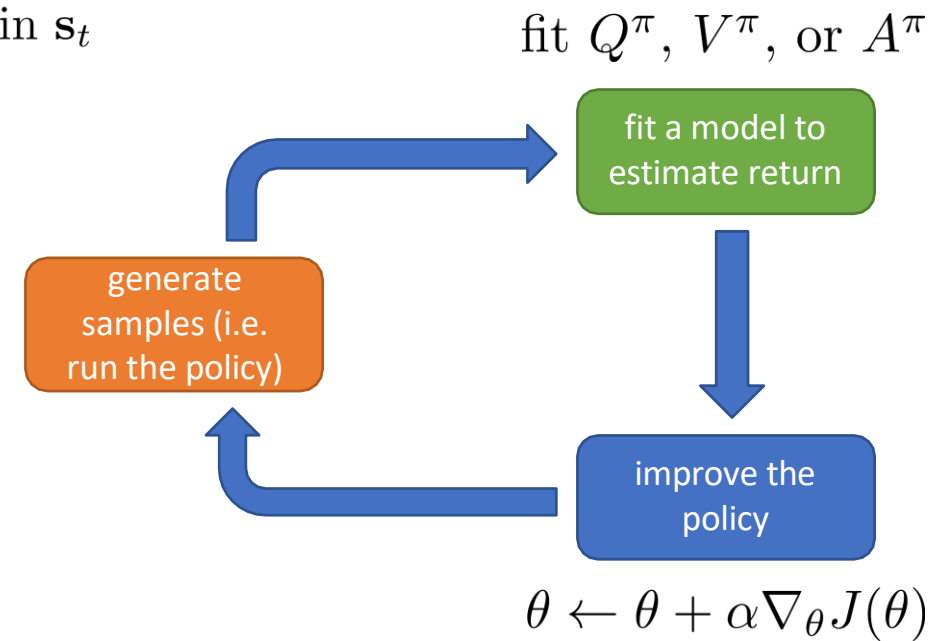
$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how much better $\mathbf{a}_t$ is

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

the better this estimate, the lower the variance

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

unbiased, but high variance single-sample estimate

fit Q^π, V^π, or A^π

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

Value function fitting

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

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

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

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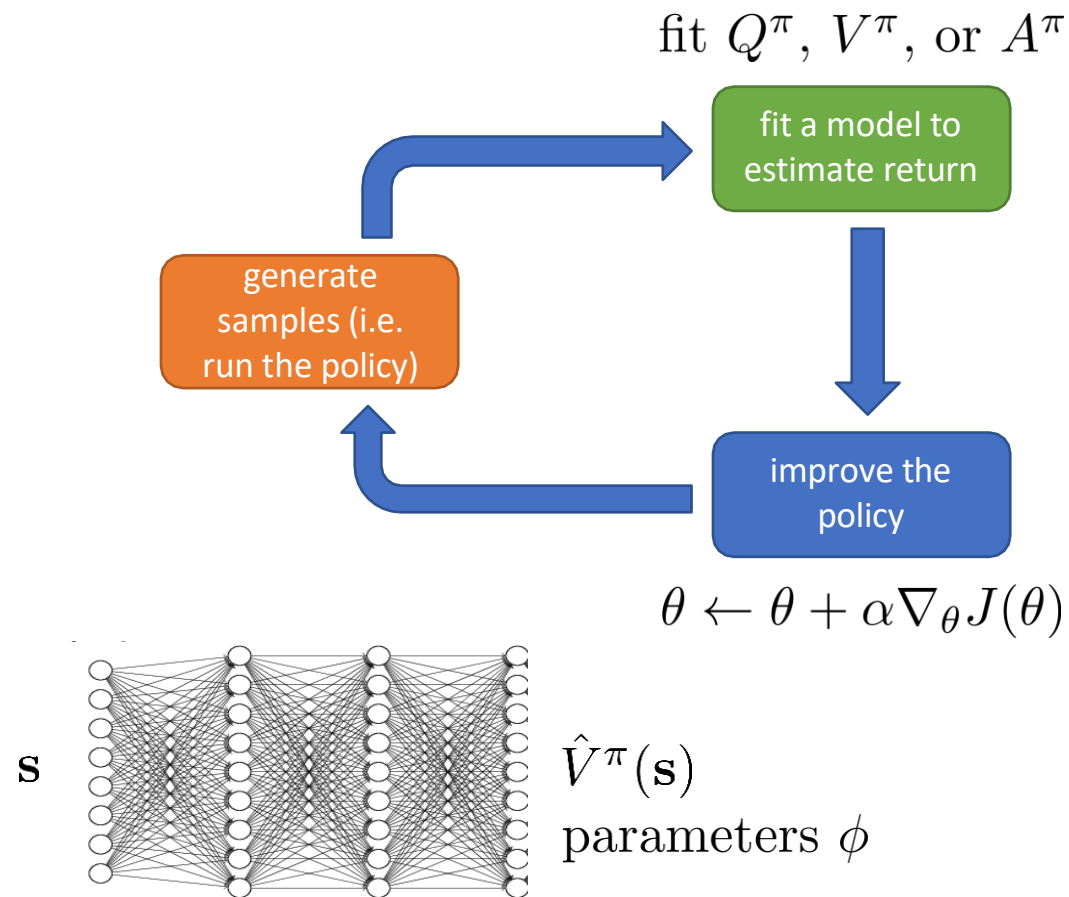
fit *what* to *what*?

Q^π, V^π, A^π?

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

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

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

fit $Q^\pi, V^\pi,$ or A^π

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

$\mathbf{s}$

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

Policy evaluation

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

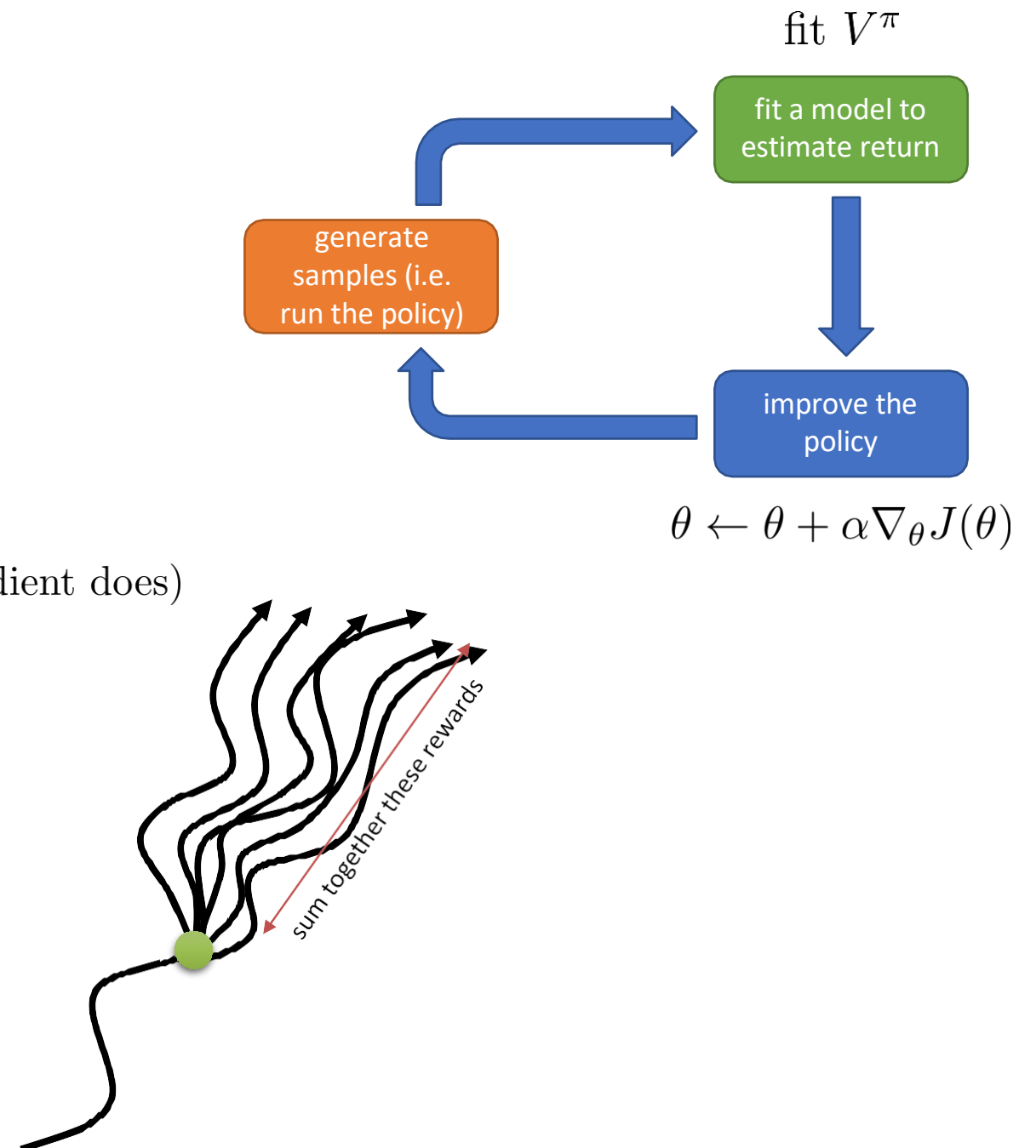$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}[V^\pi(\mathbf{s}_1)]$$

how can we perform policy evaluation?

Monte Carlo policy evaluation (this is what policy gradient does)
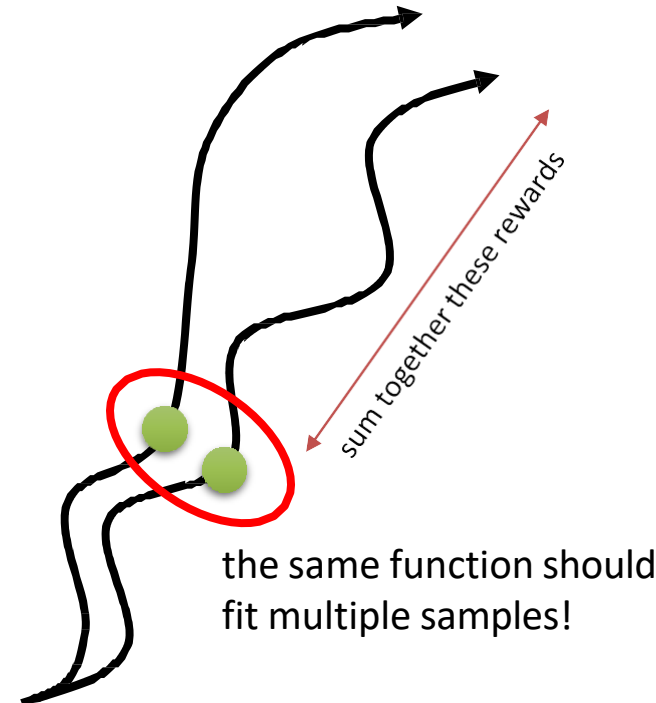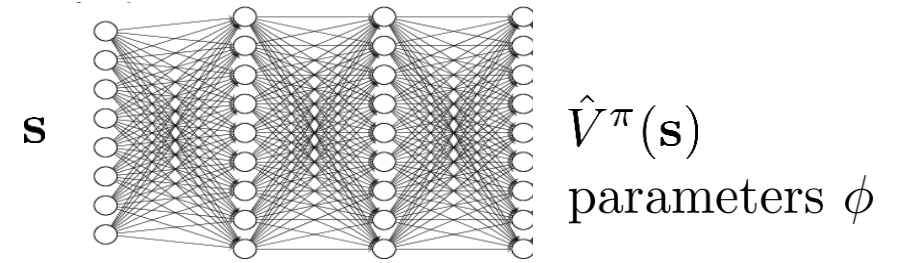
$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

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

(requires us to reset the simulator)

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

sum together these rewards

Monte Carlo evaluation with function approximation

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

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

parameters ϕ

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

but still pretty good!

training data: $\left\{ \left(\mathbf{s}_{i,t}, \underbrace{\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})}_{y_{i,t}} \right) \right\}$

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

sum together these rewards

the same function should
fit multiple samples!

Can we do better?

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

Monte Carlo target: $y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$ $\qquad\qquad\qquad$ directly use previous fitted value function!

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

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

sometimes referred to as a "bootstrapped" estimate

Policy evaluation examples

TD-Gammon, Gerald Tesauro 1992

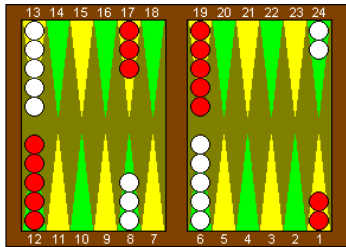AlphaGo, Silver et al. 2016

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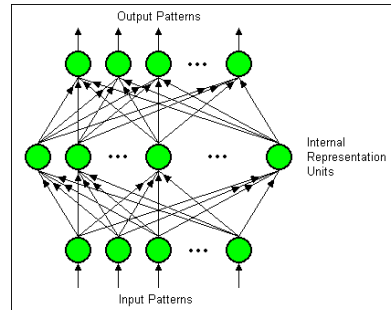

Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:
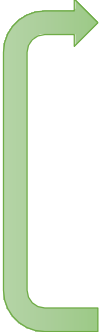expected outcome given board state

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:
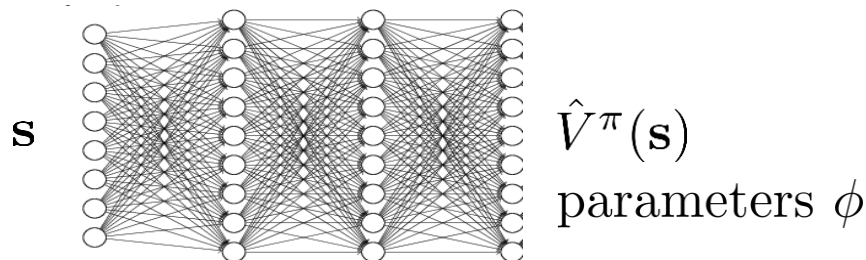expected outcome given board state

From Evaluation to Actor Critic
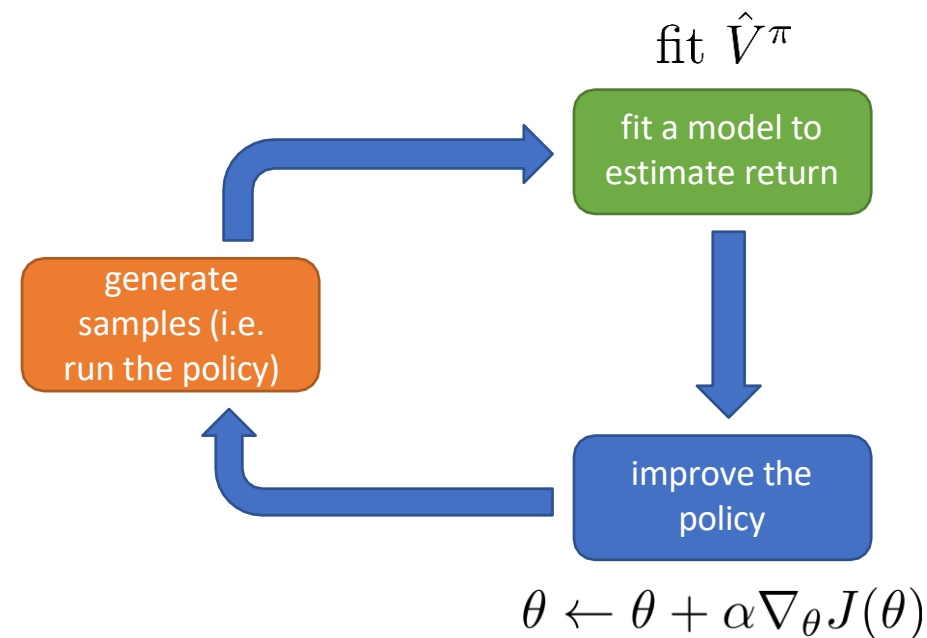
An actor-critic algorithm

batch actor-critic algorithm:

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

fit $\hat{V}^\pi$

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

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

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

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

$\mathbf{s}$

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

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

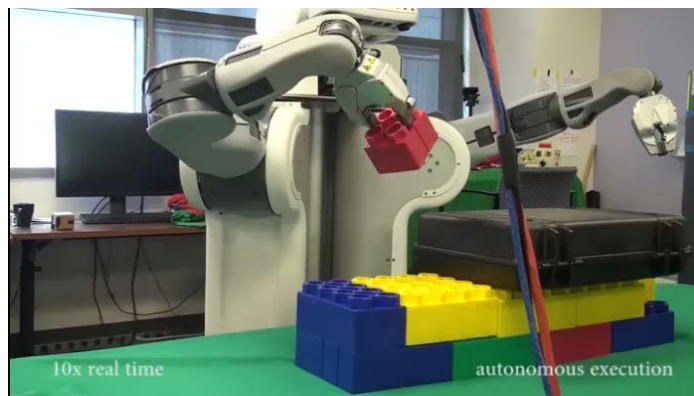# Aside: discount factors

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

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

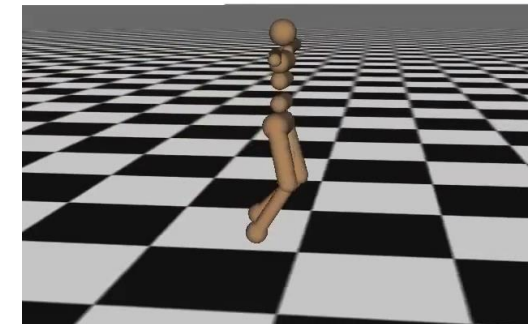what if T (episode length) is ∞?

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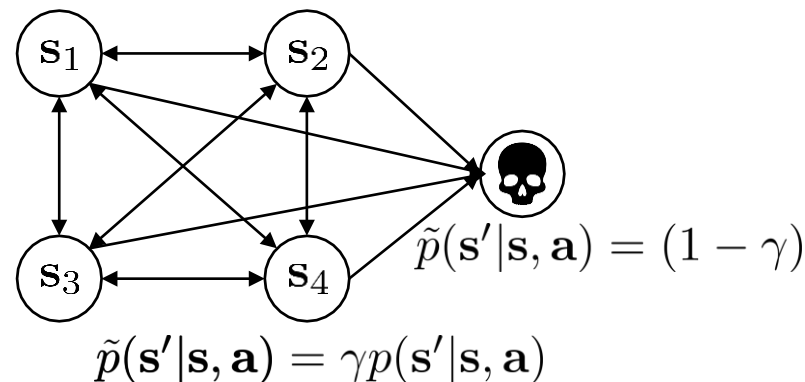

episodic tasks

Iteration 2000

continuous/cyclical tasks

simple trick: better to get rewards sooner than later

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

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

γ changes the MDP:

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

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

Aside: discount factors for policy gradients

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

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

with critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \overbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{i,t}) \right)}^{\hat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

what about (Monte Carlo) policy gradients?

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

not the same!

option 2:
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} \gamma^{t-1} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

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

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

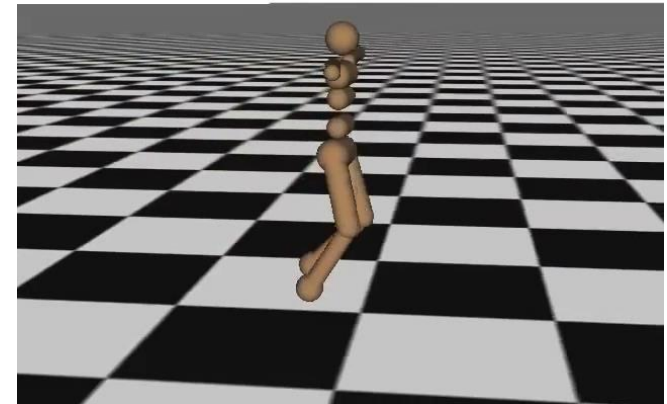(later steps matter less)

Which version is the right one?

option 1: $\quad \nabla_\theta J(\theta) \approx \dfrac{1}{N} \sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum\limits_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$
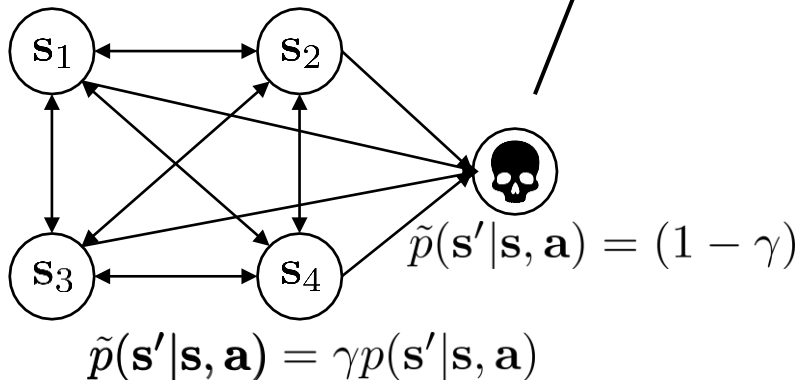
this is what we actually use...

why?

Iteration 2000

option 2: $\quad \nabla_\theta J(\theta) \approx \dfrac{1}{N} \sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T} \gamma^{t-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum\limits_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$
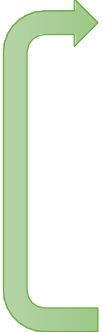
later steps don't matter if you're dead!

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

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

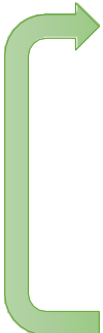**Further reading: Philip Thomas, Bias in natural actor-critic algorithms. ICML 2014**

Actor-critic algorithms (with discount)

batch actor-critic algorithm:

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

online actor-critic algorithm:

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

Acknowledgements