IASD M2 at Paris Dauphine

Deep Reinforcement Learning

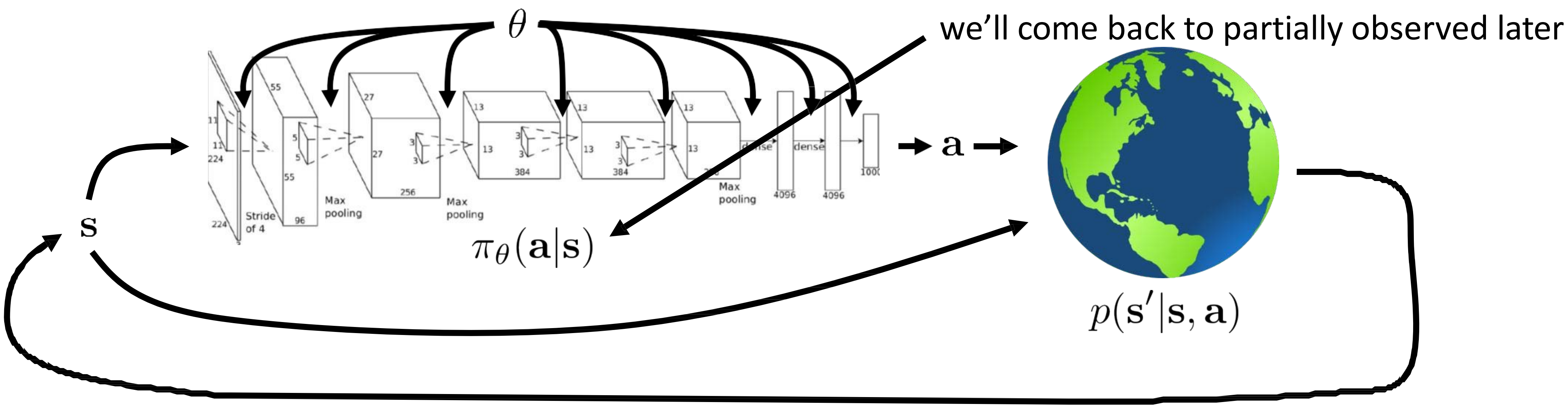
5: Policy Gradients

Eric Benhamou David Saltiel

Acknowledgement

Most of the materials of this course is based on the seminal course of Sergey Levine CS285

The goal of reinforcement learning



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

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

The goal of reinforcement learning

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

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

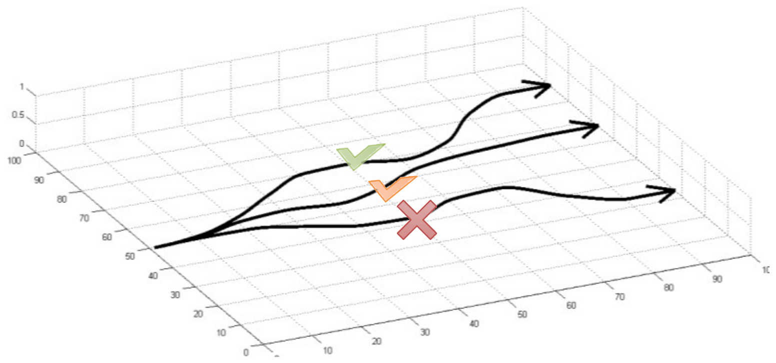
infinite horizon case

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

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

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

a convenient identity

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

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

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

Direct policy differentiation

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

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

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

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

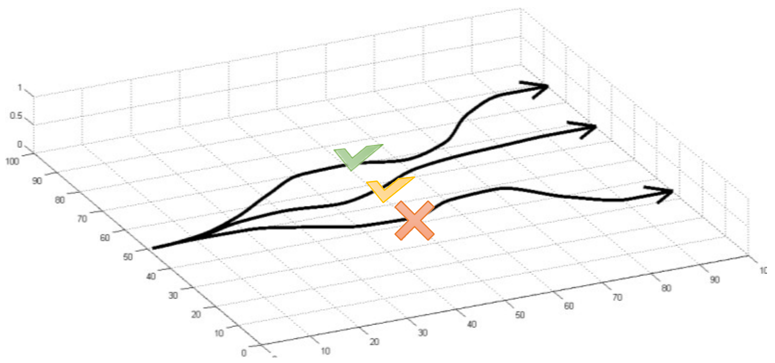
log of both sides

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

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

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

Evaluating the policy gradient

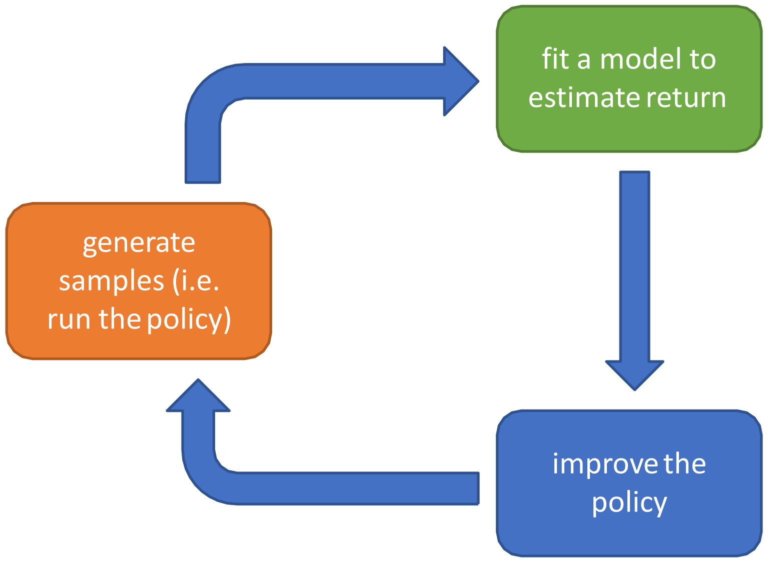


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

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

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

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

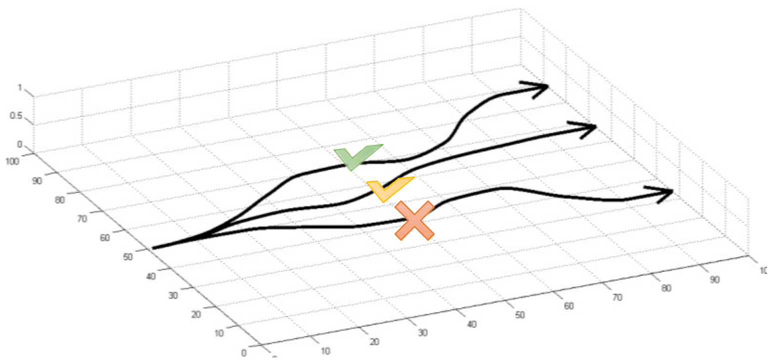


REINFORCE algorithm:

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

Understanding Policy Gradients

Evaluating the policy gradient



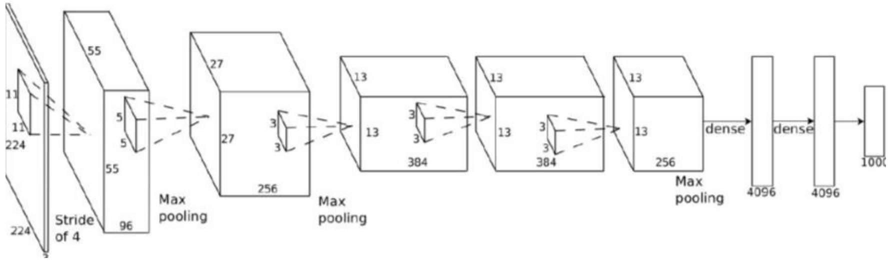
recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

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

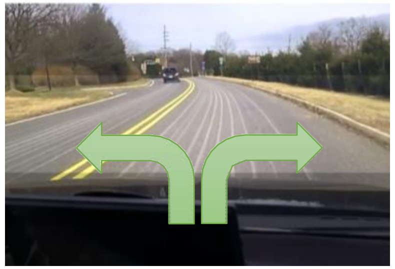
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

what is this?

$\mathbf{s}_t$



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



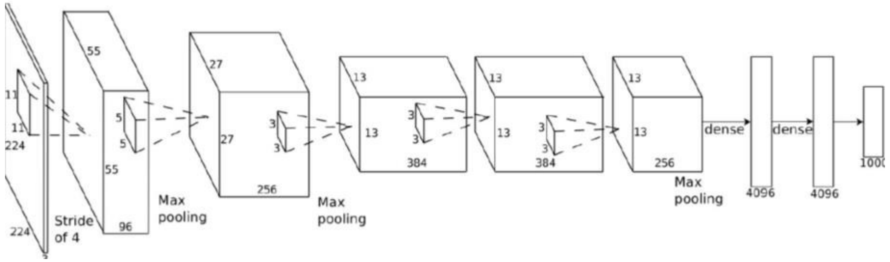
$\mathbf{a}_t$

Comparison to maximum likelihood

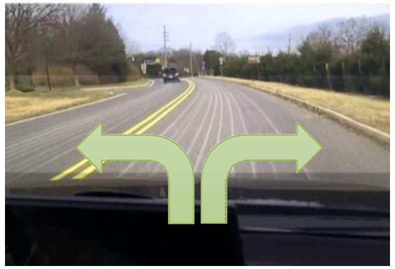
policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

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

$\mathbf{s}_t$



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



$\mathbf{a}_t$

$\mathbf{s}_t$

$\mathbf{a}_t$

→ training data → supervised learning

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

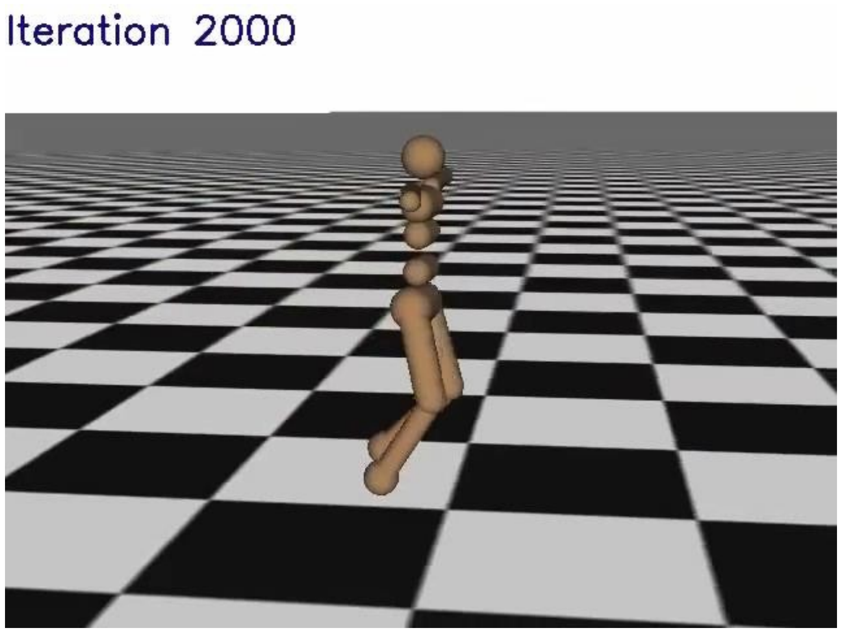
Example: Gaussian policies

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

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?

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

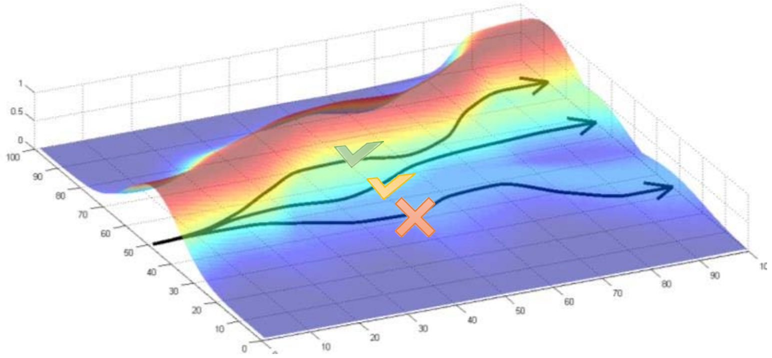
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T}\nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i)$$

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)$

good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of "trial and error"!

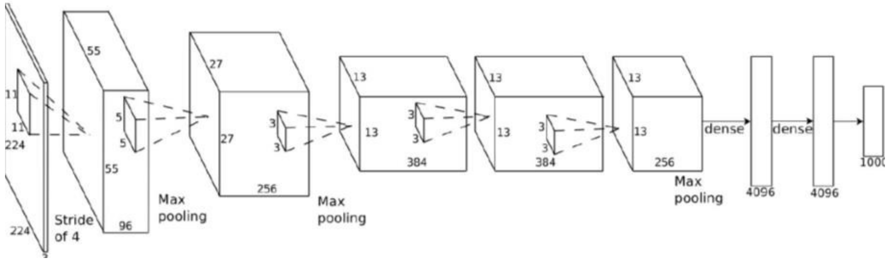


REINFORCE algorithm:

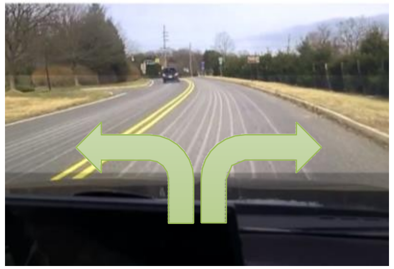
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha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

Partial observability

$\mathbf{o}_t$



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



$\mathbf{a}_t$

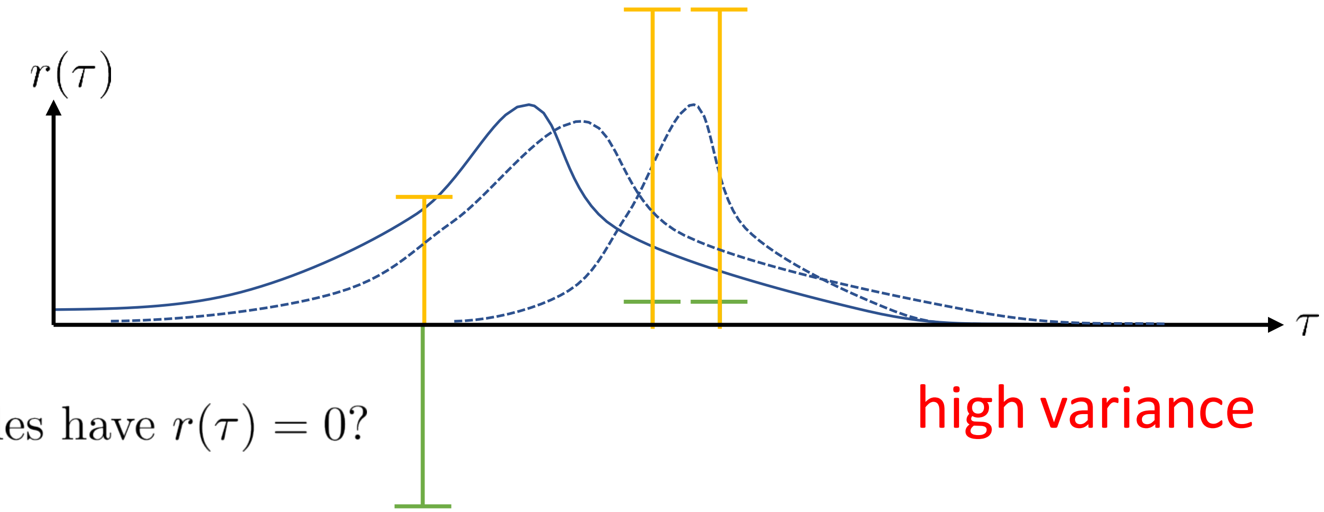
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{o}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

Markov property is not actually used!

Can use policy gradient in partially observed MDPs without modification

What is wrong with the policy gradient?

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

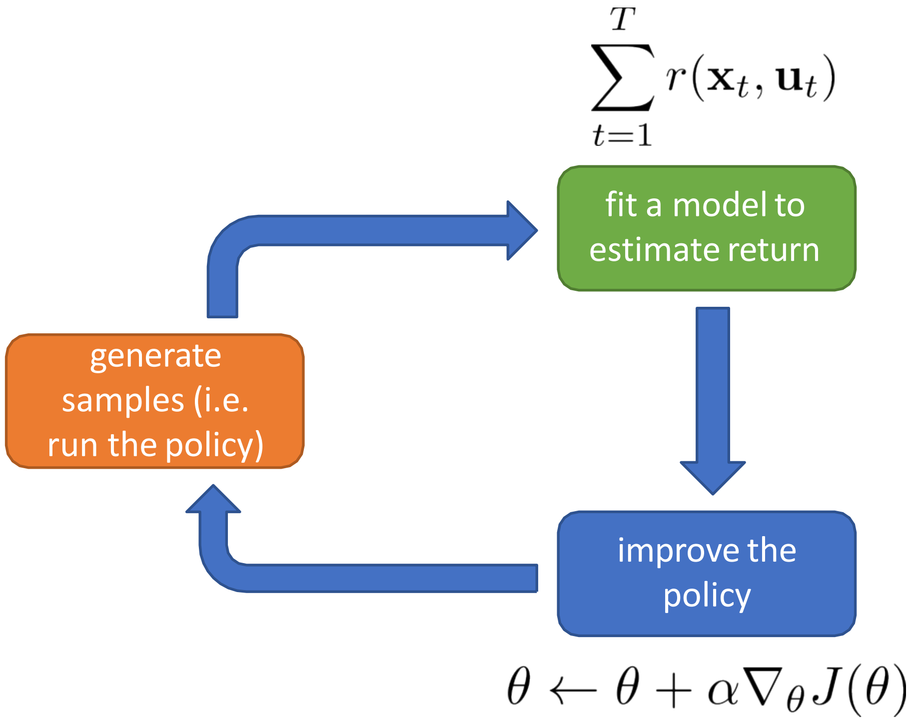


even worse: what if the two "good" samples have $r(\tau) = 0$?

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?



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

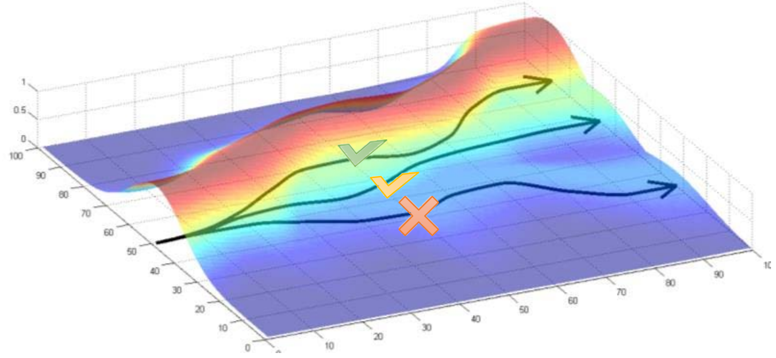
a convenient identity

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

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

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

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)

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

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

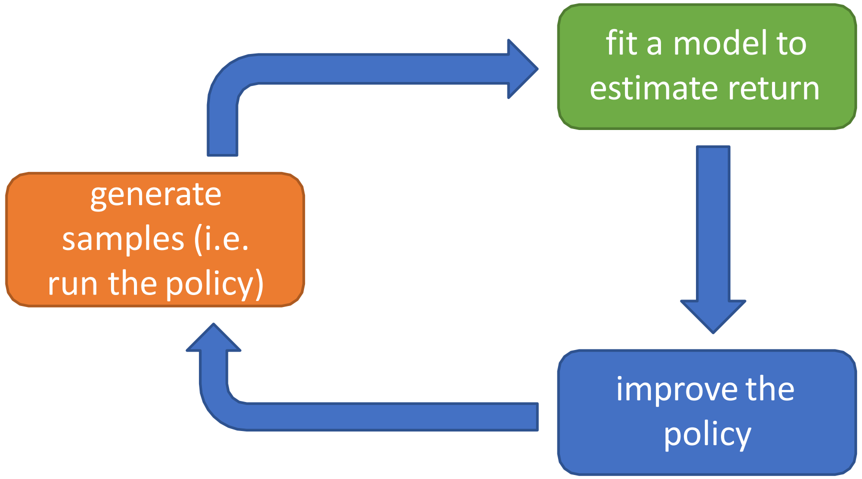
$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

← 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

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



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

Off-Policy Policy Gradients

Policy gradient is on-policy

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

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

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

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

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x) f(x) dx \\ &= \int \frac{q(x)}{q(x)} p(x) f(x) dx \\ &= \int q(x) \frac{p(x)}{q(x)} f(x) dx \\ &= E_{x \sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right] \end{aligned}$$

Deriving the policy gradient with IS

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

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

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

a convenient identity

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

The off-policy policy gradient

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

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

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

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

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

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)

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

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

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

ignore this part

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

Implementing Policy Gradients

Policy gradient with automatic differentiation

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

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

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

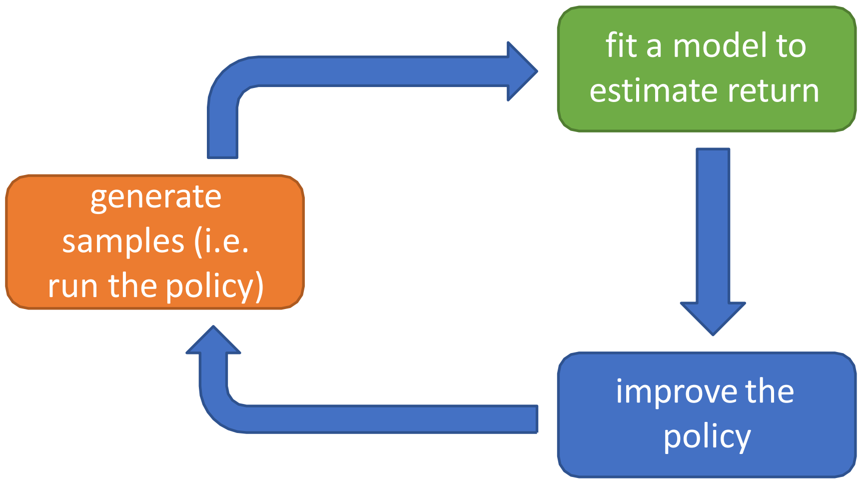
Policy gradient in practice

- 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

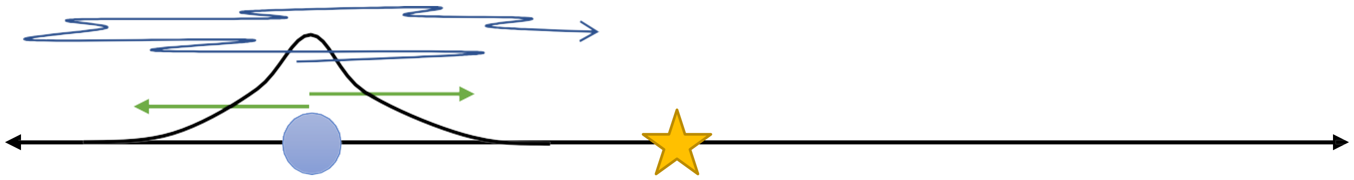
$$\hat{Q}^{\pi}(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^{T} r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$



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

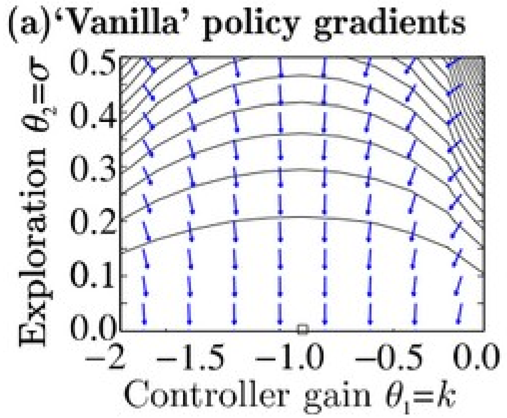
Advanced Policy Gradients

What *else* is wrong with the policy gradient?



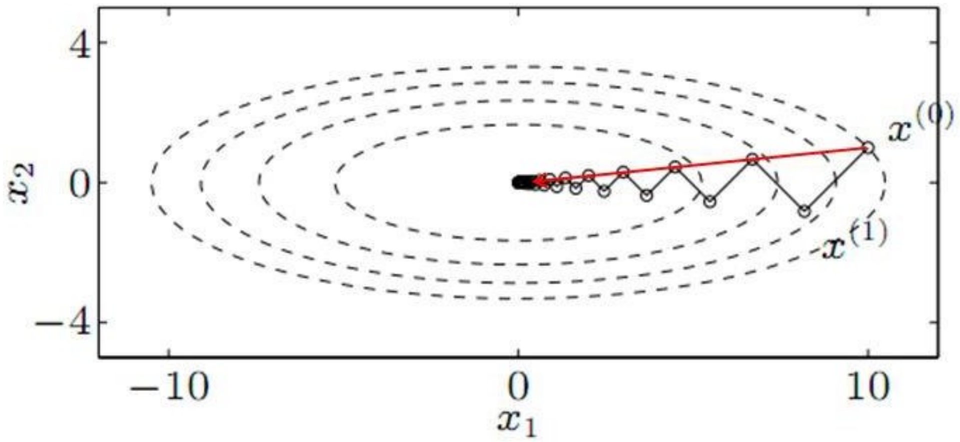
$$r(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^2 - \mathbf{a}_t^2$$

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



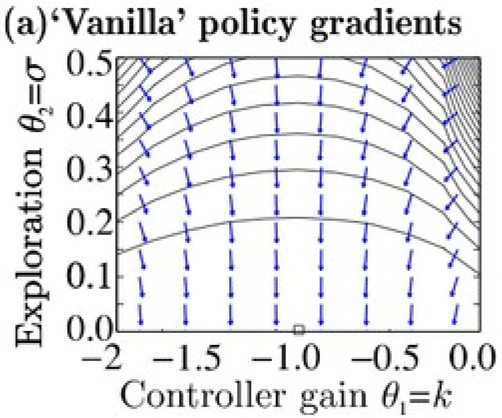
(image from Peters & Schaal 2008)

Essentially the same problem as this:



Covariant/natural policy gradient

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



some parameters change probabilities a lot more than others!

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

controls how far we go

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

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

parameterization-independent divergence measure

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

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

Fisher-information matrix can estimate with samples

Covariant/natural policy gradient

$D_{\mathrm{KL}}(\pi_{\theta'} \| \theta_\pi) \approx (\theta' - \theta)^T \mathbf{F}(\theta' - \theta)$

$\mathbf{F} = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})^T]$

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

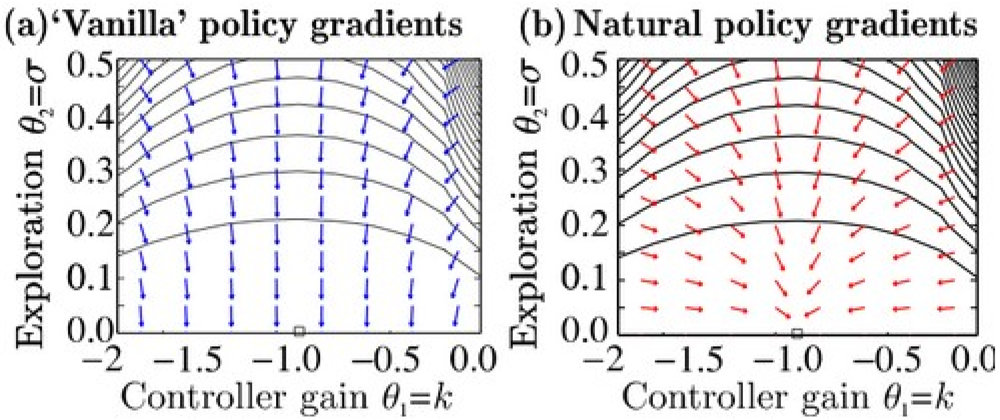
$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$

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



(figure from Peters & Schaal 2008)

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

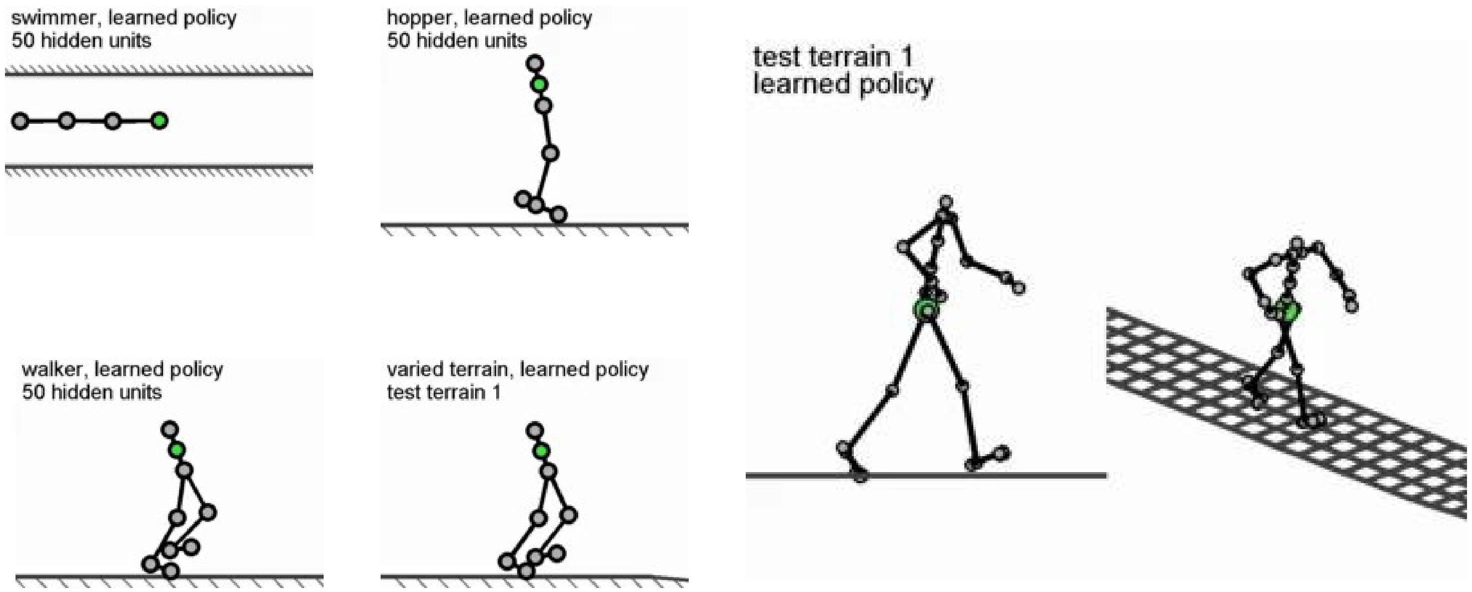
Advanced policy gradient topics

- 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

Example: policy gradient with importance sampling

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

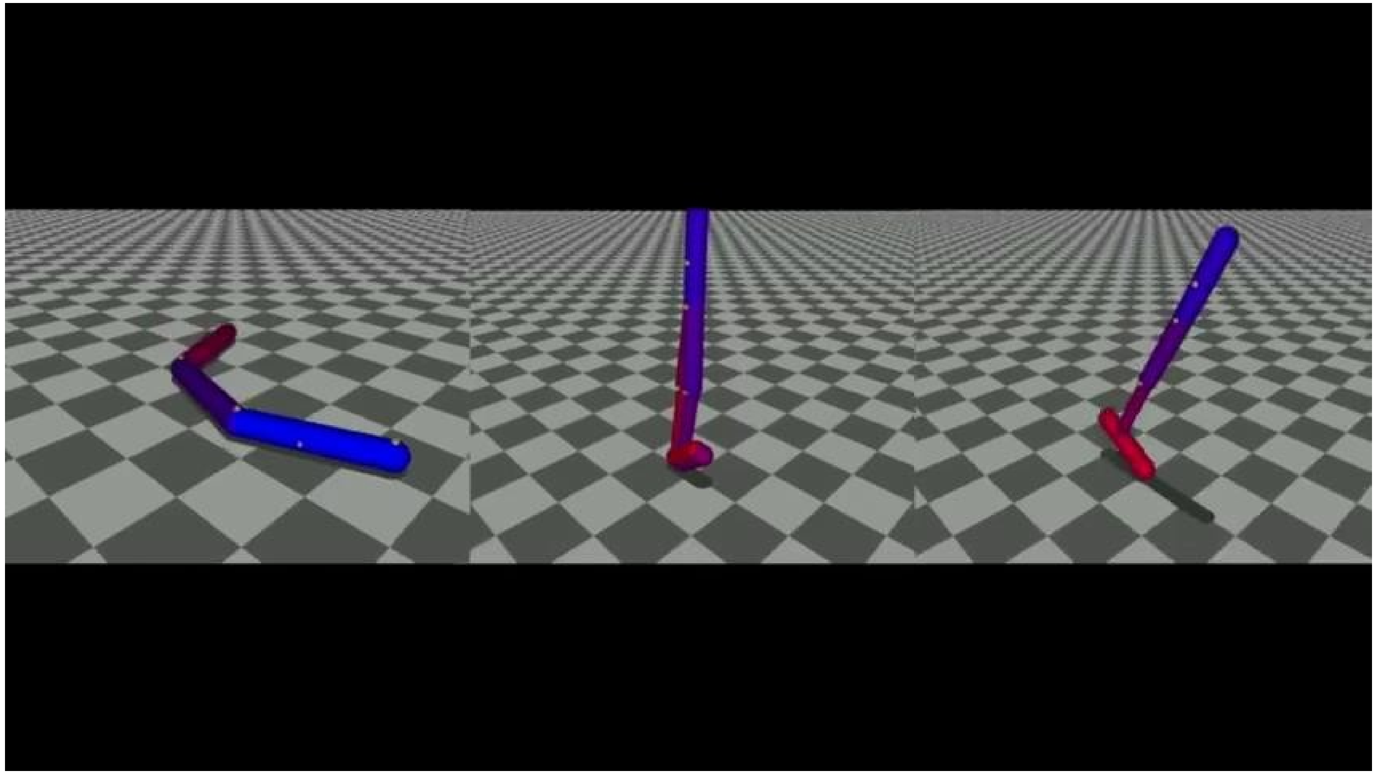
- Incorporate example demonstrations using importance sampling
- Neural network policies



Levine, Koltun ‘13

Example: trust region policy optimization

- Natural gradient with automatic step adjustment
- Discrete and continuous actions
- Code available (see Duan et al. '16)



Schulman, Levine, Moritz, Jordan, Abbeel. '15

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