

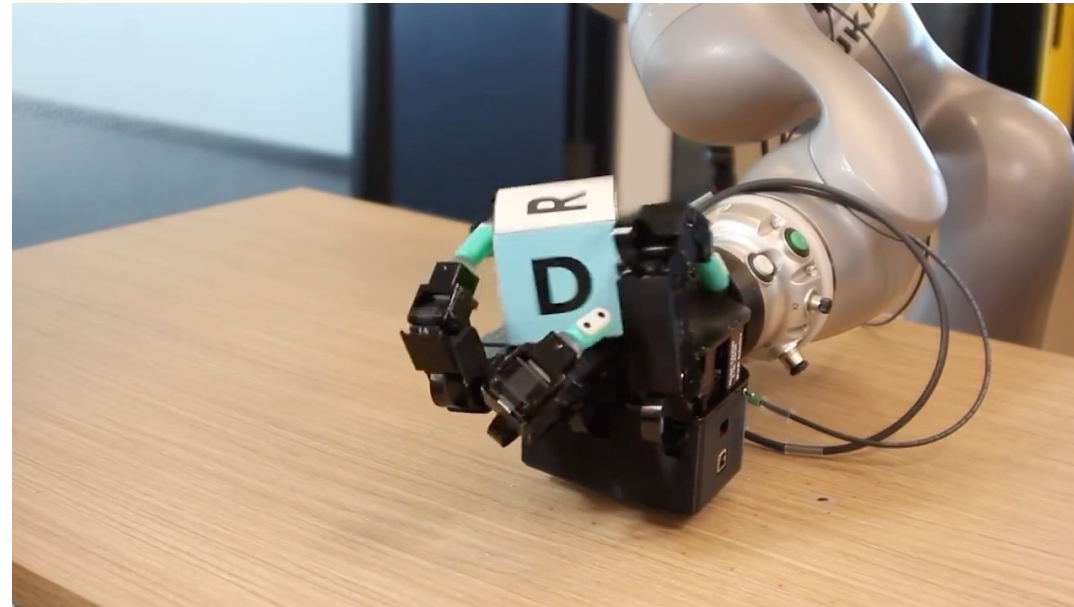


# Reinforcement Learning

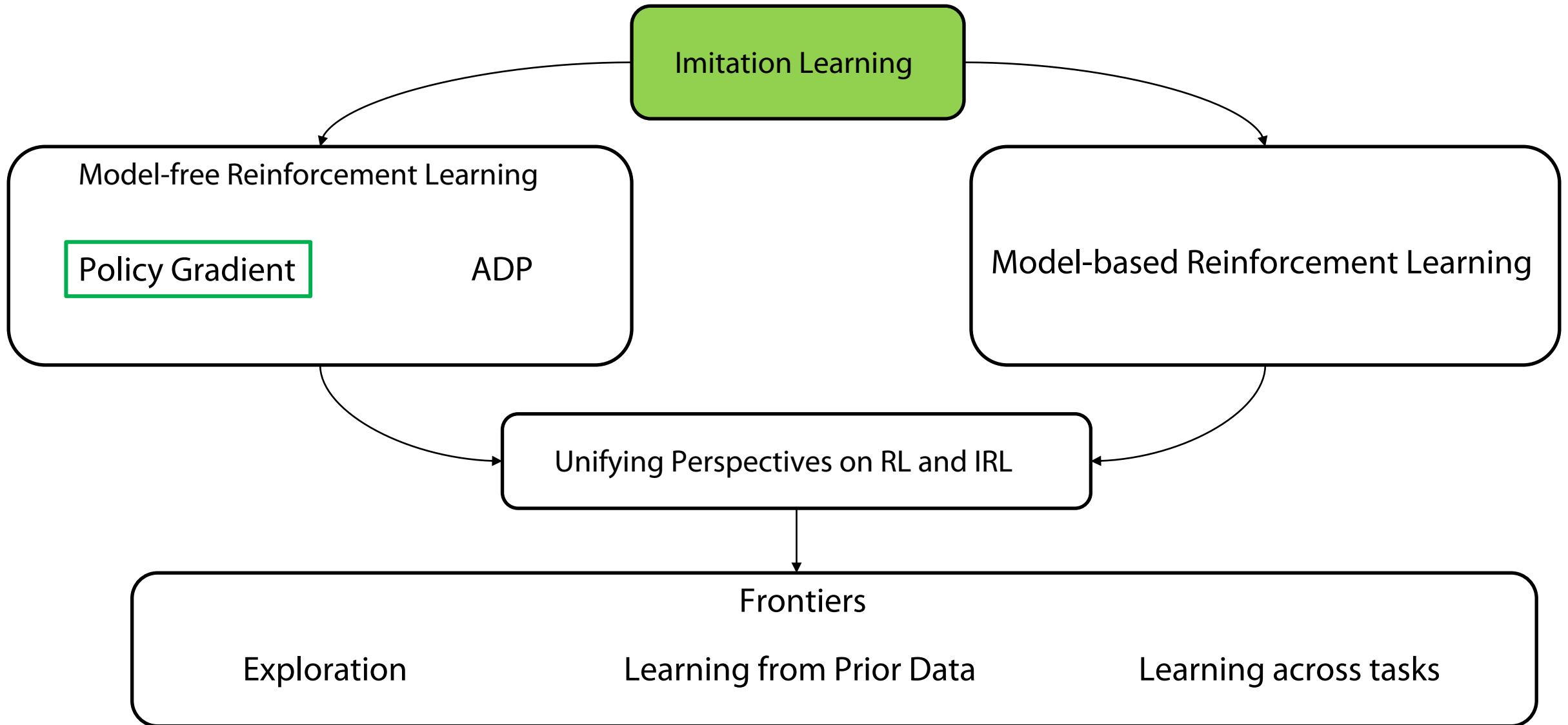
## Spring 2026

Abhishek Gupta

TA: Mateo Guaman Castro



# Class Structure



# Lecture outline

---

Deriving the Policy Gradient

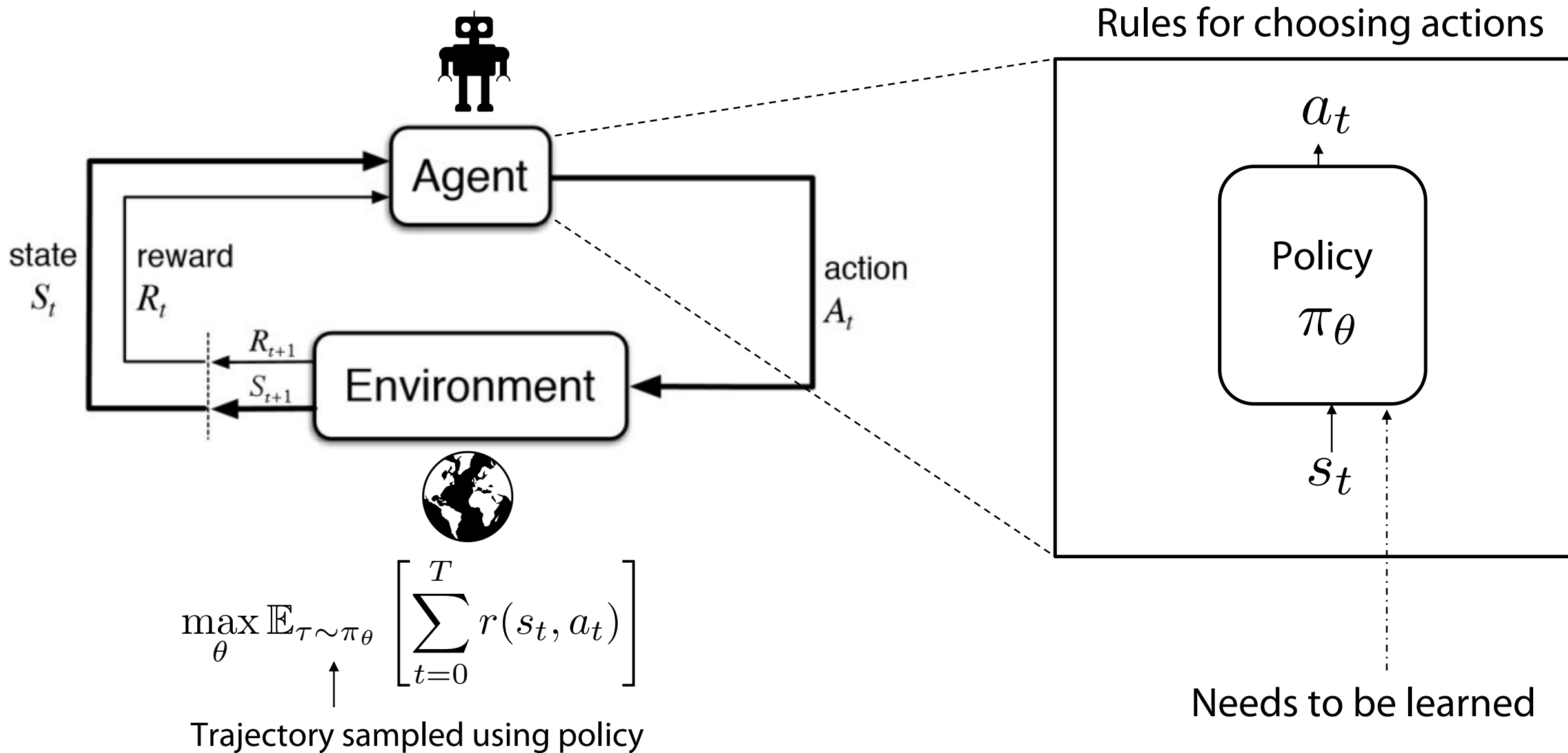


What makes the Policy Gradient Challenging? - Variance



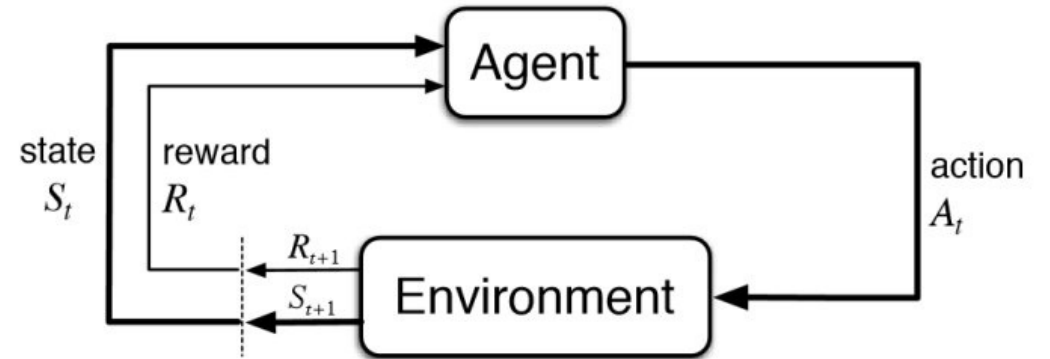
Natural Policy Gradients and Covariant Parameterization

# Objective of Reinforcement Learning



# Finite horizon vs infinite horizon objective

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$



## Finite horizon

$$\mathbb{E}_{\pi_{\theta}^t} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$

Time-dependent policy  
(not stationary)

$$\mathbb{E}_{\pi_{\theta}^t} \left[ \sum_{t=0}^T \gamma^t r(s_t, a_t) \right]$$

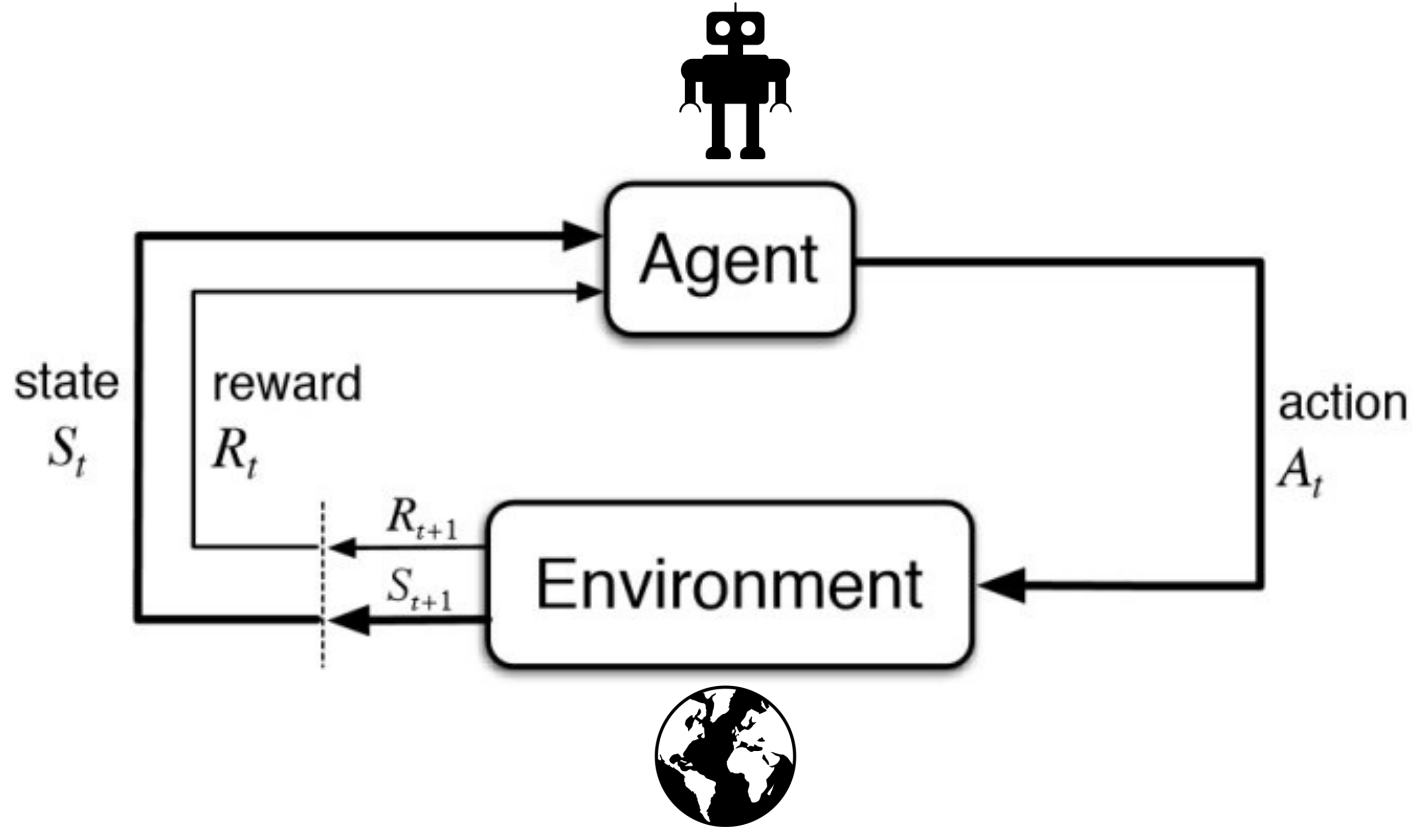
## Infinite horizon discounted

$$\mathbb{E}_{\pi_{\theta}} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

Time-independent (stationary) policy  
→ Need discount to prevent blow up

**Lemma:** there always exists a stationary optimal policy

# Objective of Reinforcement Learning



Assumptions:

1. Rewards are additive
2. Dynamics can be sampled from, but functional form is unknown
3. Rewards are provided as every state is visited, functional form is unknown

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$

↑  
Trajectory sampled using policy

# Connection to Optimal Control

Closely related: typically problem of finding control given a plant

$$\min_{x,u} \int_0^x L(t, x(t), u(t)).dx$$

w.r.t

$$x'(t) = f(x(t), u(t))$$

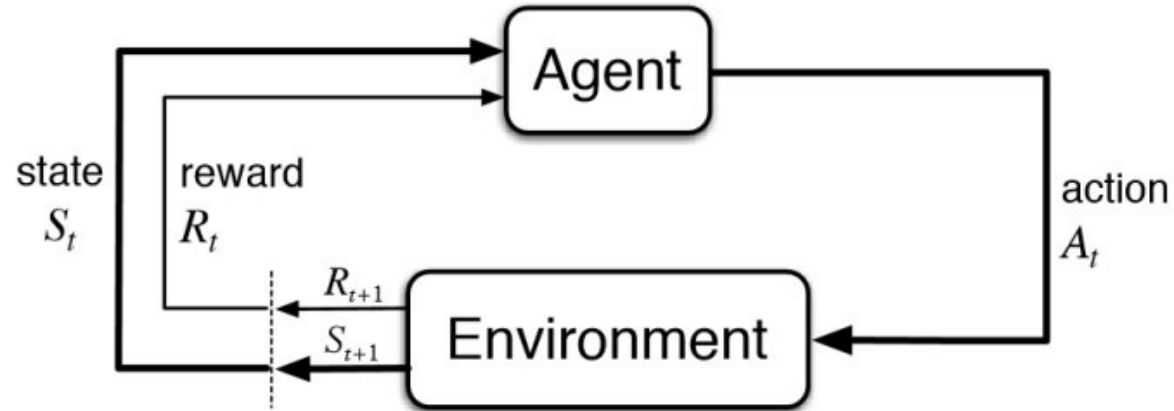


$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$

Main difference: model known vs unknown

Minor differences: Cost vs reward, discrete vs continuous time

# How should we optimize this objective?



$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$

Gradient Ascent

Dynamic Programming

Model-Based Optimization

Each method has its own +/-

# Lecture outline

---

Deriving the Policy Gradient

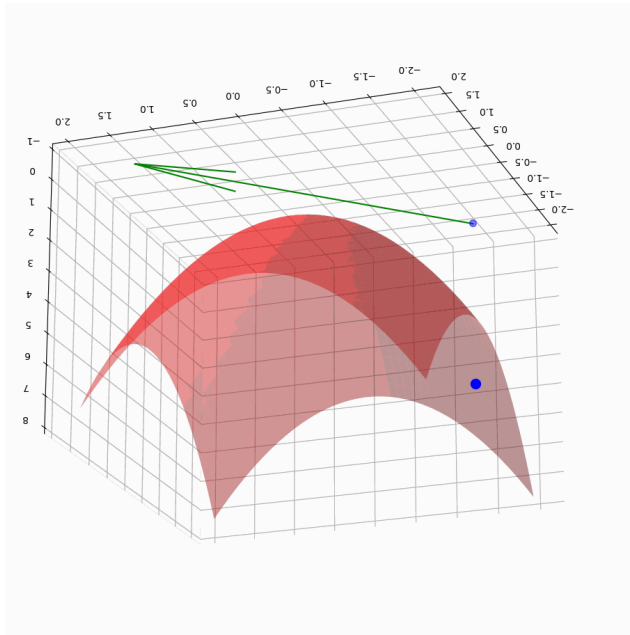
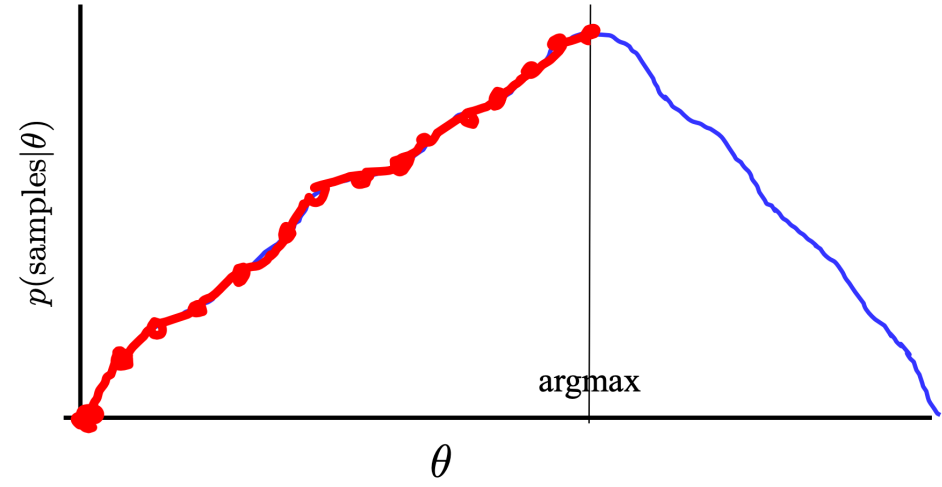
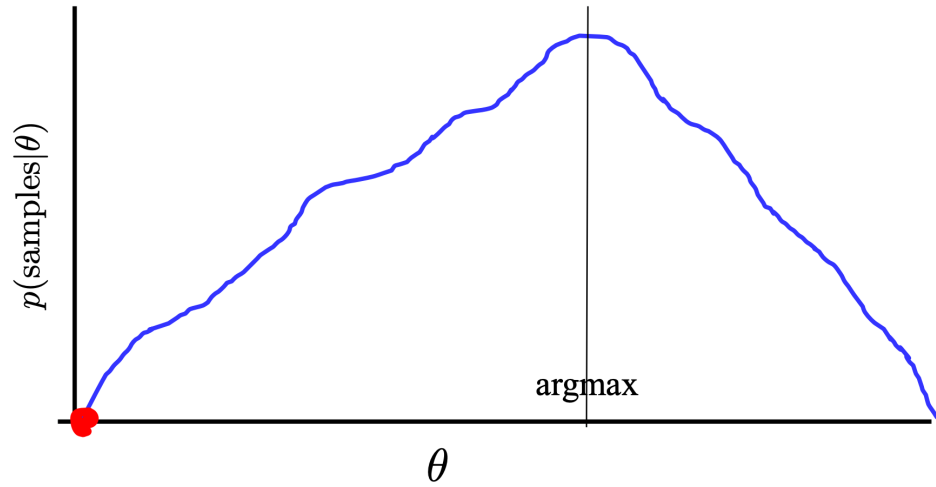


What makes the Policy Gradient Challenging? - Variance



What makes the Policy Gradient Challenging? – Covariant Parameterization

# Gradient Ascent



Simple view – move the parameters in the direction of the gradient of the objective

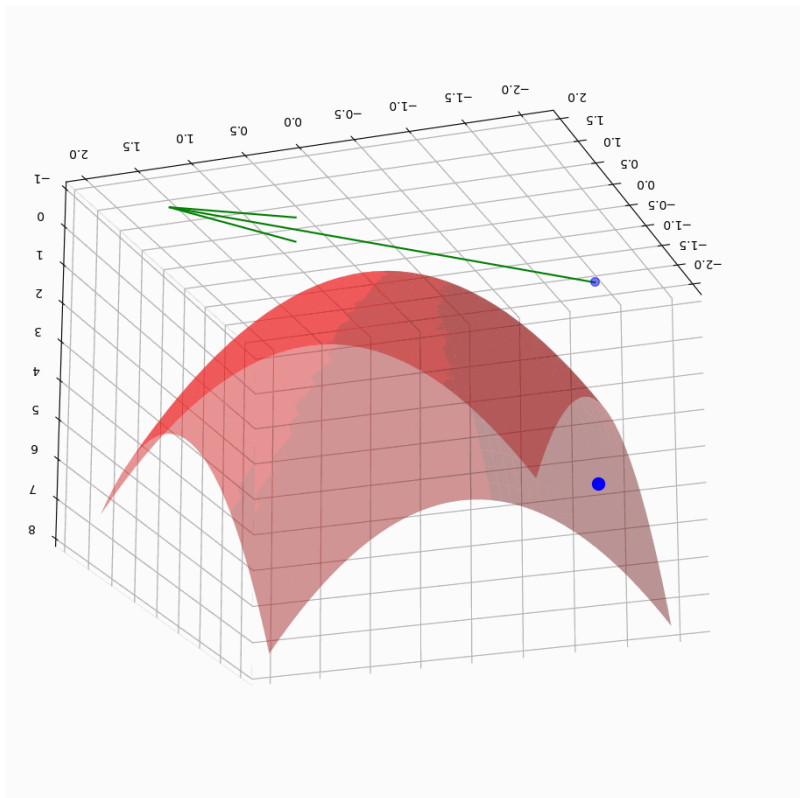
$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J(\theta) |_{\theta=\theta_i}$$

More later: can be derived as steepest ascent in Euclidean norm

# Gradient Ascent for Supervised Learning

Recall our imitation learning objective  $\arg \max_{\theta} \mathbb{E}_{(s^*, a^*) \sim \mathcal{D}} [\log \pi_{\theta}(a^* | s^*)]$

Let's apply gradient ascent



$$\nabla_{\theta} \mathbb{E}_{(s^*, a^*) \sim \mathcal{D}} [\log \pi_{\theta}(a^* | s^*)]$$

$$\nabla_{\theta} \int p(s^*, a^*) \log \pi_{\theta}(a^* | s^*) ds^* da^*$$

$$\int p(s^*, a^*) \nabla_{\theta} \log \pi_{\theta}(a^* | s^*) ds^* da^*$$

$$\mathbb{E}_{(s^*, a^*) \sim \mathcal{D}} [\nabla_{\theta} \log \pi_{\theta}(a^* | s^*)]$$

Compute gradient and average

# Ok let's do gradient ascent for the RL objective

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$
$$= \int p_{\theta}(\tau) R(\tau) d\tau$$



REINFORCE gradient descent (RL)

$$\nabla_{\theta} \mathbb{E}_{x \sim p_{\theta}(x)} [f(x)]$$

(Cannot simply compute average of expectation)

Standard gradient descent (supervised learning)

Gradient wrt expectation variable, not of integrand!

(Whiteboard)

$$\nabla_{\theta} \mathbb{E}_{x \sim g(x)} [f_{\theta}(x)]$$

(Gradient passes inside the expectation – compute gradient and average)

# Taking the gradient of sum of rewards

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right]$$

Let's take the gradient of this objective

$$J(\theta) = \int p_{\theta}(\tau) R(\tau) d(\tau)$$

Let's think about this from the trajectory view

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \int p_{\theta}(\tau) R(\tau) d(\tau)$$

We need to express this in a way that we can evaluate with expectations

$$= \int \nabla_{\theta} p_{\theta}(\tau) R(\tau) d(\tau) = \int \frac{p_{\theta}(\tau)}{p_{\theta}(\tau)} \nabla_{\theta} p_{\theta}(\tau) R(\tau) d(\tau)$$

$$= \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) R(\tau) d(\tau) = \mathbb{E}_{p_{\theta}(\tau)} [\nabla_{\theta} \log p_{\theta}(\tau) R(\tau)]$$

$$\frac{d \log(x)}{d\theta} = \frac{d \log(x)}{dx} \frac{dx}{d\theta} = \frac{1}{x} \frac{dx}{d\theta}$$

Use chain rule

REINFORCE trick

# Taking the gradient of return

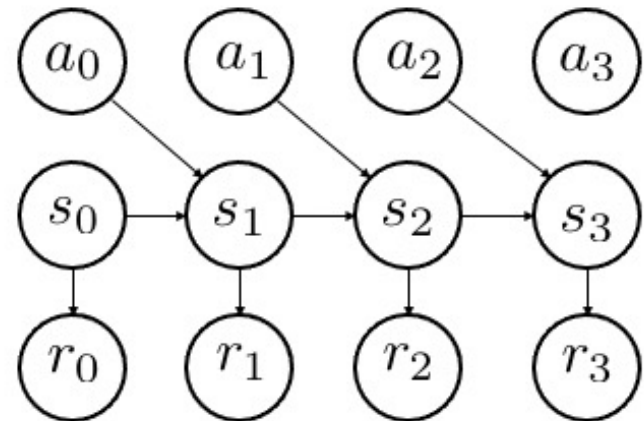
Initial State

Dynamics

Policy

$$p_{\theta}(\tau) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1} | s_t, a_t) \pi(a_t | s_t)$$

(Ancestral sampling)



$$\log p_{\theta}(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log p(s_{t+1} | s_t, a_t) + \log \pi(a_t | s_t)$$

$$\nabla_{\theta} \log p_{\theta}(\tau) = \cancel{\nabla_{\theta} \log p(s_0)} + \sum_{t=0}^{T-1} \cancel{\nabla_{\theta} \log p(s_{t+1} | s_t, a_t)} + \nabla_{\theta} \log \pi(a_t | s_t)$$

$$\nabla_{\theta} \log p_{\theta}(\tau) = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi(a_t | s_t)$$

Model Free!!

# Taking the gradient of return

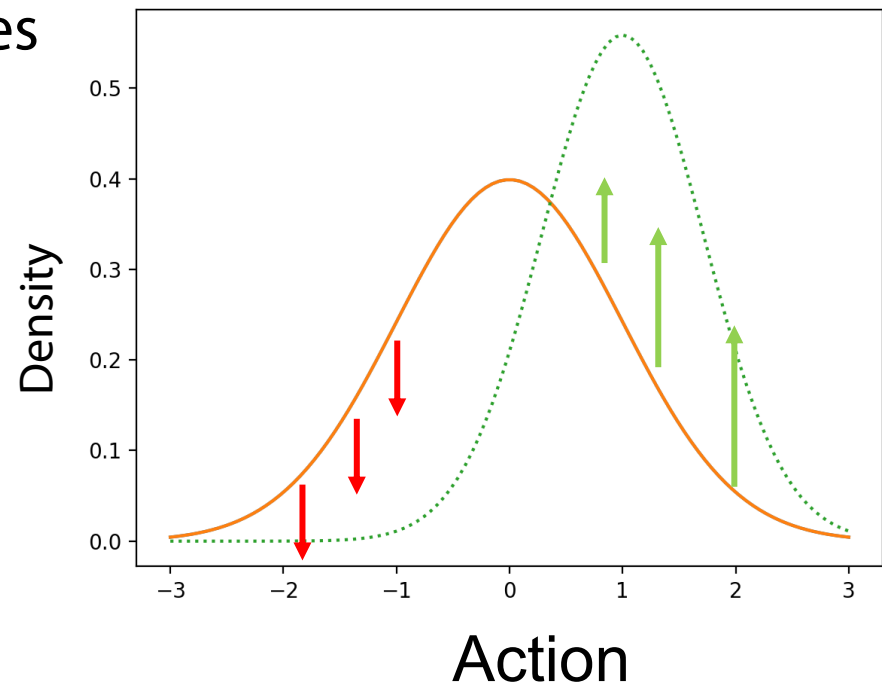
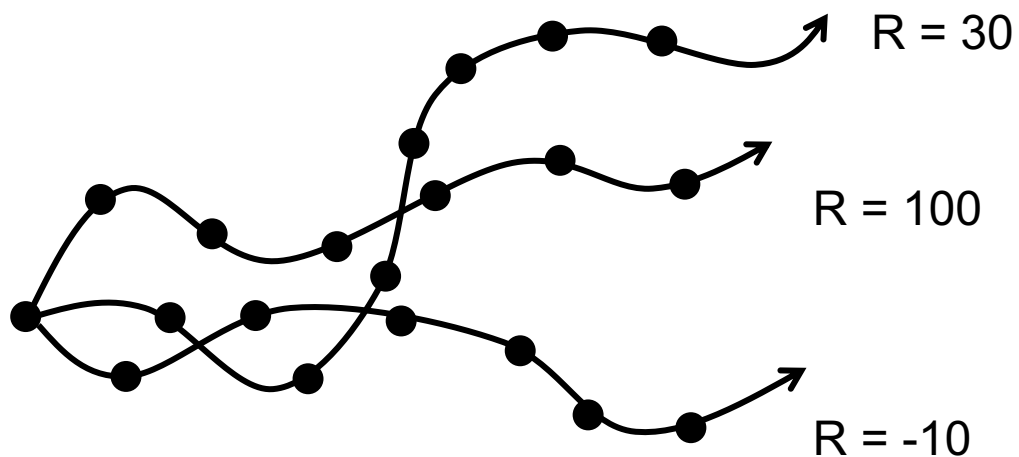
$$\begin{aligned} \nabla_{\theta} J(\theta) &= \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[ \nabla_{\theta} \log p_{\theta}(\tau) \sum_{t=0}^T r(s_t, a_t) \right] \\ \nabla_{\theta} J(\theta) &= \mathbb{E}_{\substack{s_0 \sim p(s_0) \\ s_{t+1} \sim p(s_{t+1} | s_t, a_t) \\ a_t \sim \pi(a_t | s_t)}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \sum_{t'=0}^T r(s_{t'}, a_{t'}) \right] \\ &\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i) \quad (\text{approximating using samples}) \end{aligned}$$

(Monte-Carlo approximation)

# What does this mean?

$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau \approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)$$

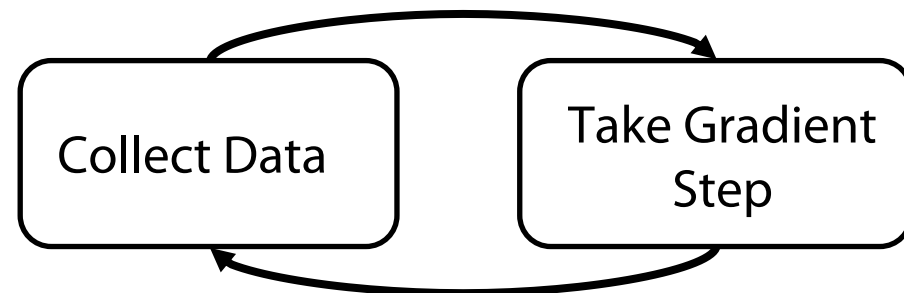
Increase the likelihood of actions in high return trajectories



# Resulting Algorithm (REINFORCE)

$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau$$

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J(\theta) |_{\theta=\theta_i}$$



REINFORCE algorithm:

On-policy



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

# How is this related to supervised learning?

## Reinforcement Learning

$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau$$

$$\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)$$

## Supervised Learning

$$\max_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} [\log p_{\theta}(y|x)]$$

$$\approx \frac{1}{N} \sum_i \nabla_{\theta} \log p_{\theta}(y^i | x^i)$$

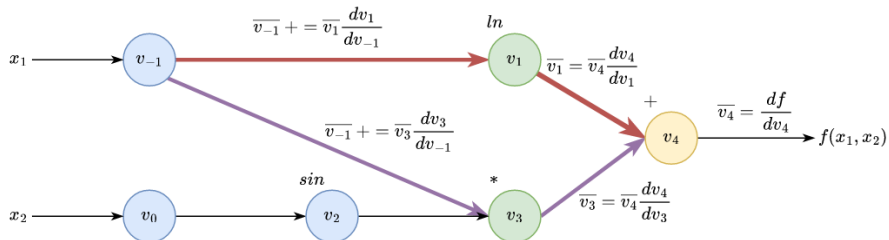
PG = select good data + increase likelihood of selected data

# How do we implement this?

REINFORCE algorithm:

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

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) d\tau \approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)$$



Compute gradients with autodiff

Sum up rewards in a trajectory



# How do we implement this?

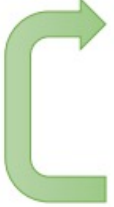
## Maximum likelihood:

```
# 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)
```

^Standard maximum likelihood training

# How do we implement this?

REINFORCE algorithm:

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

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 → Sum of rewards
# 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)
```

Formalizes the notion of trial and error

# How do we implement this?

$$\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

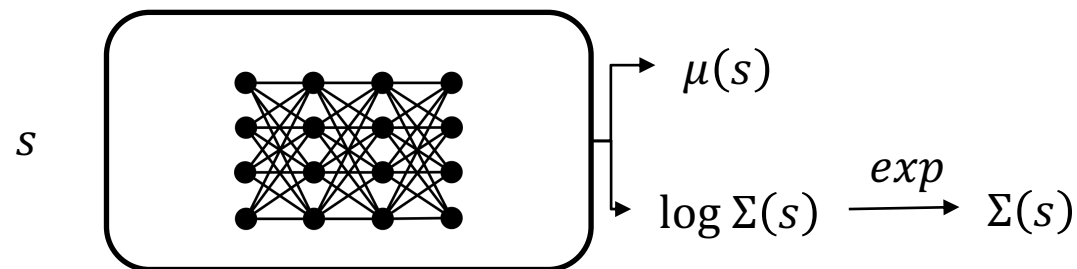
Let's try it for a Gaussian

$$\pi(\mathbf{a} | \mathbf{s})$$

$$= \pi(\mathbf{a} | \boldsymbol{\mu}_{\theta}(\mathbf{s}), \boldsymbol{\Sigma}_{\theta}(\mathbf{s}))$$

$$= \pi(\mathbf{a} | \boldsymbol{\mu}_{\theta}(\mathbf{s}), \boldsymbol{\Sigma}_{\theta}(\mathbf{s})) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}_{\theta}(\mathbf{s})|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_{\theta}(\mathbf{s}))^{\top} \boldsymbol{\Sigma}_{\theta}(\mathbf{s})^{-1} (\mathbf{x} - \boldsymbol{\mu}_{\theta}(\mathbf{s}))\right)$$

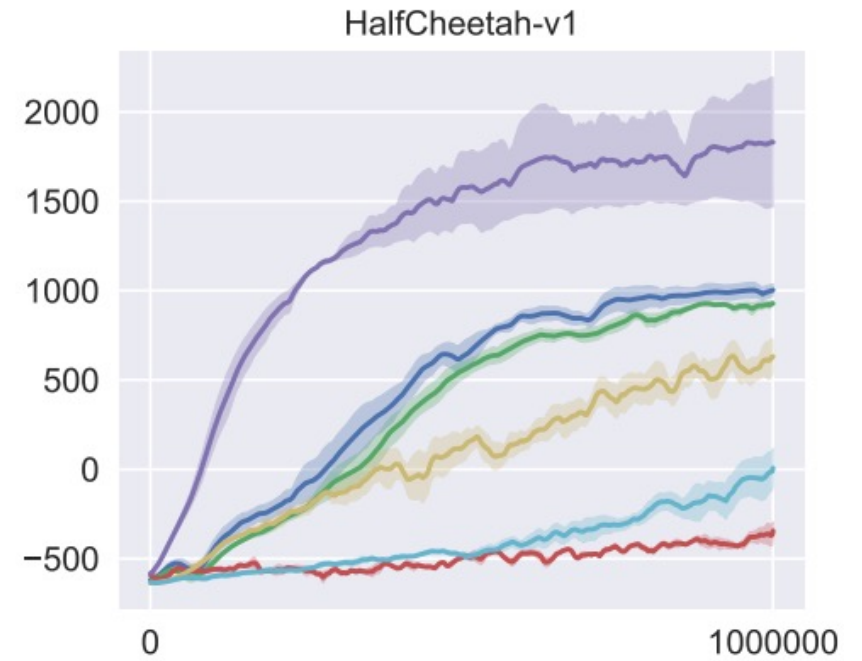
Similar for categorical or other distributions



Easier for distributions where likelihood can be expressed symbolically

# Does this work?

Comparison of  
RL algorithms  
in Humanoid-v2  
using CleanRL



Kind of?

# Lecture outline

---

## **Recap: Deriving the Policy Gradient**



What makes the Policy Gradient Challenging? - Variance

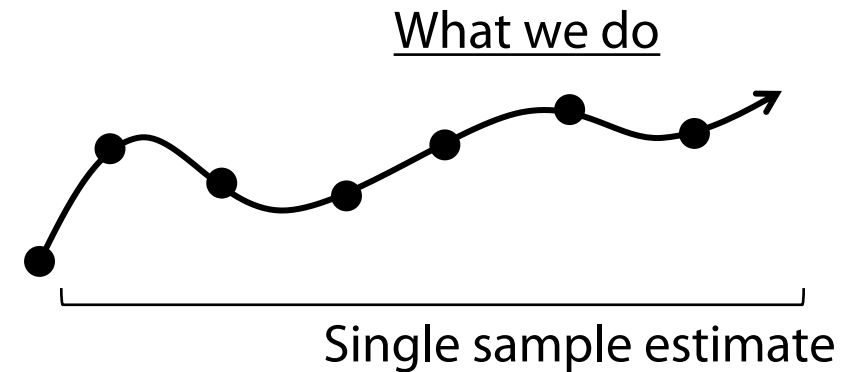


Natural Policy Gradients and Covariant Parameterization

# What makes policy gradient challenging?

Hard to tell what matters without many samples

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau \\ &\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i) \end{aligned}$$



For every (s, a) pair, weight by only the sum of rewards in the current trajectory

Couples together all actions

Susceptible to scale variations

Susceptible to lucky samples

Makes policy gradient unstable, requires huge numbers of samples and huge batch size

# What makes policy gradient challenging?

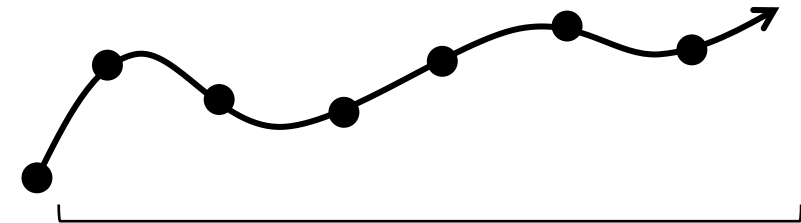
$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau$$

$$\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)$$

**High variance estimator!!**

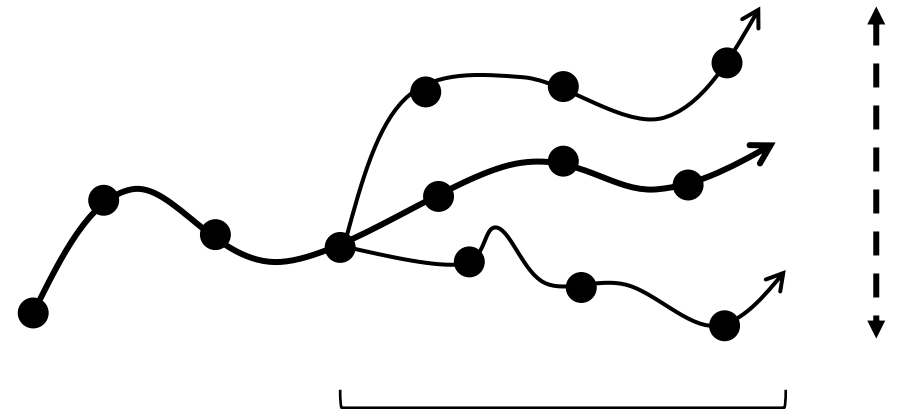
Hard to tell what matters without many samples

What we do



Single sample estimate

What we actually want

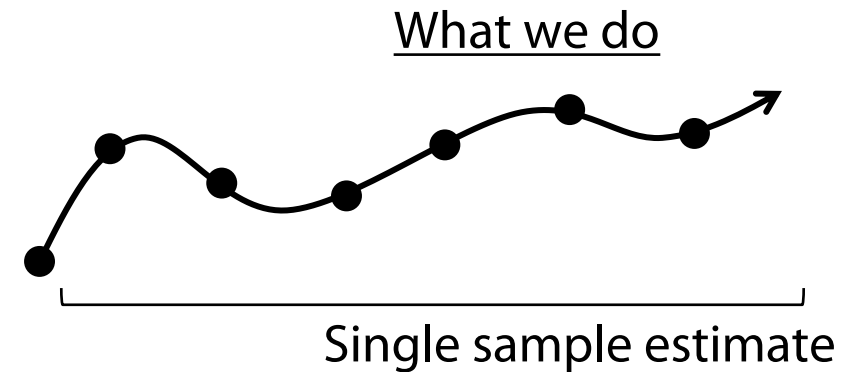


Averaged return estimate

# What makes policy gradient challenging?

Hard to tell what matters without many samples

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau \\ &\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i) \end{aligned}$$



For every (s, a) pair, weight by only the sum of rewards in the current trajectory

Couples together all actions

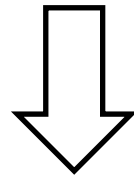
# Variance Reduction with Causality

Idea: Trajectory returns depend on past and future, but we only care about the future, since actions cannot affect the past. Instead, consider **“return-to-go”**

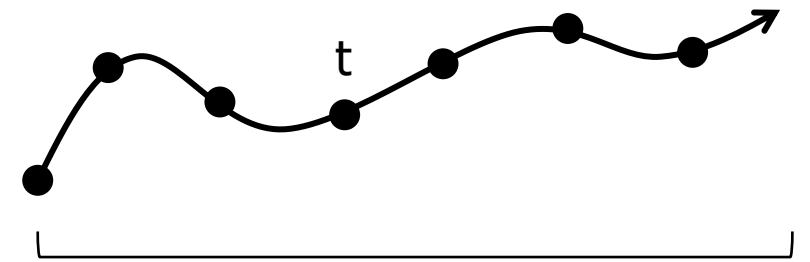
$$\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \underbrace{\sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)}$$

Includes  $t' < t$

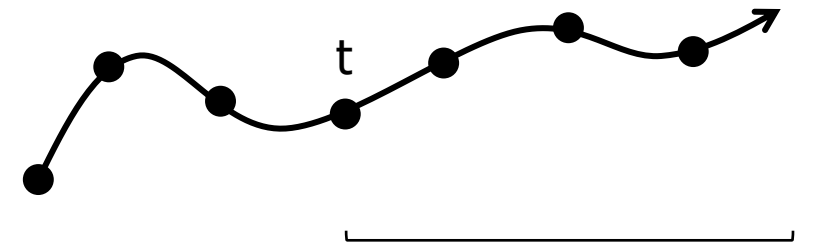
Ignore past terms



$$\frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=t}^T r(s_{t'}^i, a_{t'}^i)$$



Full trajectory return

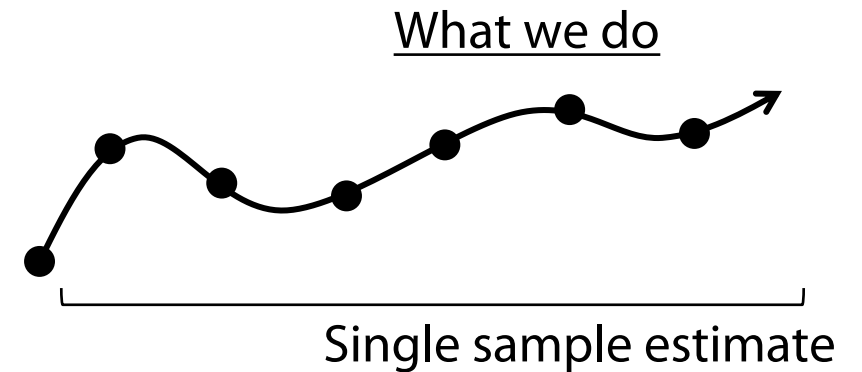


Return to go

# What makes policy gradient challenging?

Hard to tell what matters without many samples

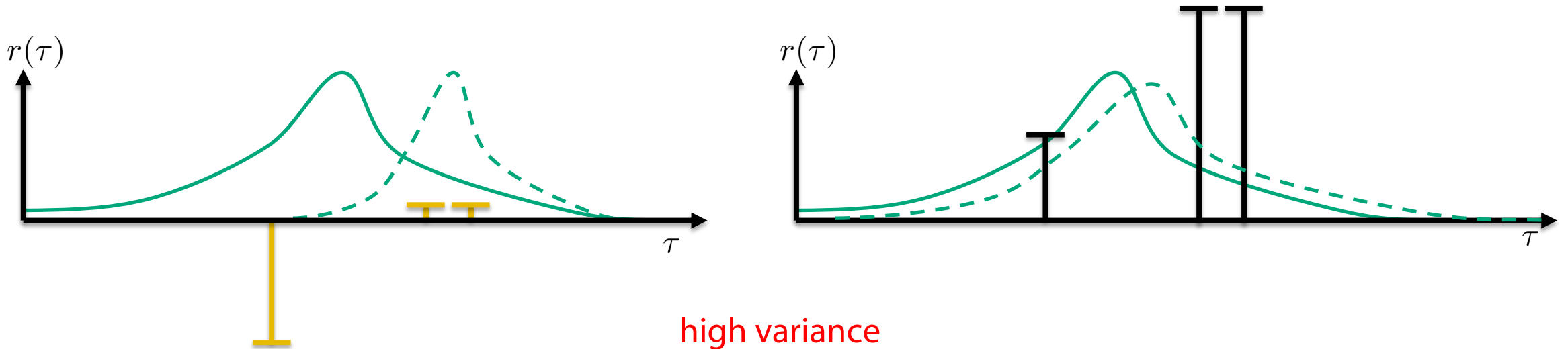
$$\begin{aligned} \nabla_{\theta} J(\theta) &= \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau \\ &\approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i) \end{aligned}$$



For every (s, a) pair, weight by only the sum of rewards in the current trajectory

Susceptible to scale variations

# Policy gradient is susceptible to scale variations




Arbitrarily uncentered, scaled returns can lead to huge variance:

- Imagine all rewards were positive, every action would be pushed up, some more than others
- What if instead, we pushed down some actions and pushed up some others (even if rewards are positive)



# Variance Reduction with a Baseline

Idea: We can reduce variance by subtracting a current state dependent function from the policy gradient return

$$\frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \left[ \sum_{t'=t}^T r(s_{t'}^i, a_{t'}^i) - b(s_t) \right]$$


Baseline: Centers the returns, reduces variance

But does this increase bias??

# Variance Reduction with a Baseline

$$\int_{\mathcal{S}} \int_{\mathcal{A}} p(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left[ \sum_{t'=t}^T r(s_{t'}, a_{t'}) - b(s_t) \right] ds_t da_t$$

$$\int_{\mathcal{S}} \int_{\mathcal{A}} p(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left[ \sum_{t'=t}^T r(s_{t'}, a_{t'}) \right] ds_t da_t - \int_{\mathcal{S}} \int_{\mathcal{A}} p(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) b(s_t) ds_t da_t$$

Let us show this is 0!

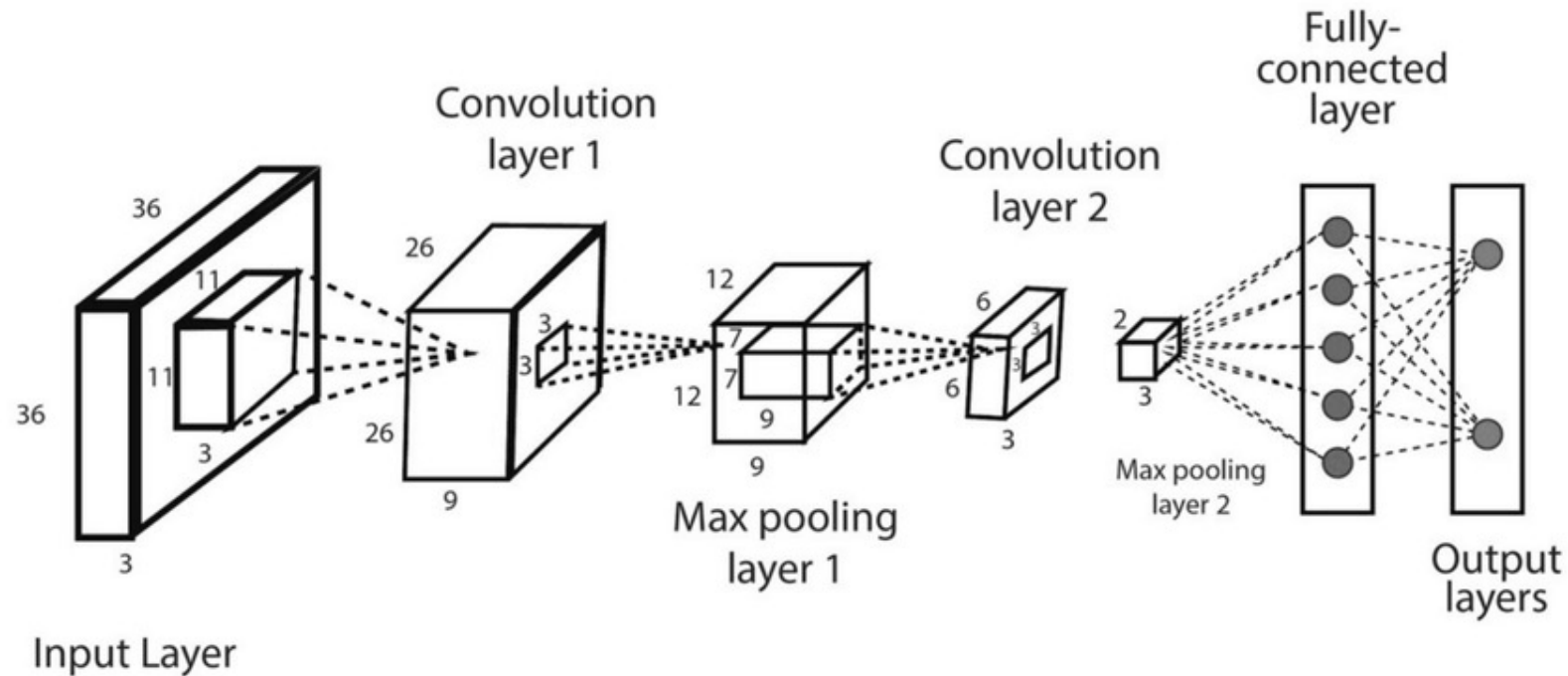
# Variance Reduction with a Baseline

$$\begin{aligned}\int \int p(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) [b(s_t)] ds_t da_t &= \int \int p(s_t) \pi_{\theta}(a_t | s_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) [b(s_t)] ds_t da_t \\ &= \int p(s_t) b(s_t) \int \pi_{\theta}(a_t | s_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) da_t ds_t \\ &= \int p(s_t) b(s_t) \int \nabla_{\theta} \pi_{\theta}(a_t | s_t) da_t ds_t \\ &= \int p(s_t) b(s_t) \nabla_{\theta} \int \pi_{\theta}(a_t | s_t) da_t ds_t = \int p(s_t) b(s_t) \nabla_{\theta} (1) ds_t = 0\end{aligned}$$

Unbiased!

# Learning Baselines

Baselines are typically learned as deep neural nets from  $\mathbb{R}^s \rightarrow \mathbb{R}^1$



$$\arg \min_{\hat{V}} \frac{1}{N} \sum_{j=1}^N \left\| \hat{V}(s_t^j) - \sum_{t=1}^H r(s_t^j, a_t^j) \right\| \quad \nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left( \sum_{t'=t}^T r(s_{t'}, a_{t'}) - \hat{V}(s_t) \right) \right]$$

Minimize with Monte-Carlo regression at every iteration, club with policy gradient

# Why do baselines really reduce variance?

---

Let's define variance:  $\text{Var}[x] = E[x^2] - E[x]^2$        $\nabla_{\theta} J(\theta) = E_{\tau \sim p_{\theta}(\tau)}[\nabla_{\theta} \log p_{\theta}(\tau)(r(\tau) - b)]$

Whiteboard

# Lecture outline

---

**Recap: Deriving the Policy Gradient**



**What makes the Policy Gradient Challenging? - Variance**



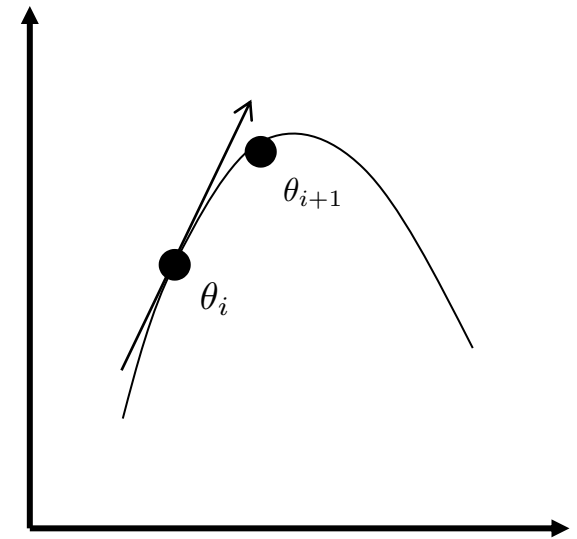
Natural Policy Gradients and Covariant Parameterization

# Take a deeper look at REINFORCE

$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau \approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)$$

Gradient ascent is steepest ascent on linear approximation under the Euclidean metric!

$$\begin{aligned} \max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T r(s_t, a_t) \right] \\ = J(\theta) \end{aligned}$$



# Take a deeper look at REINFORCE

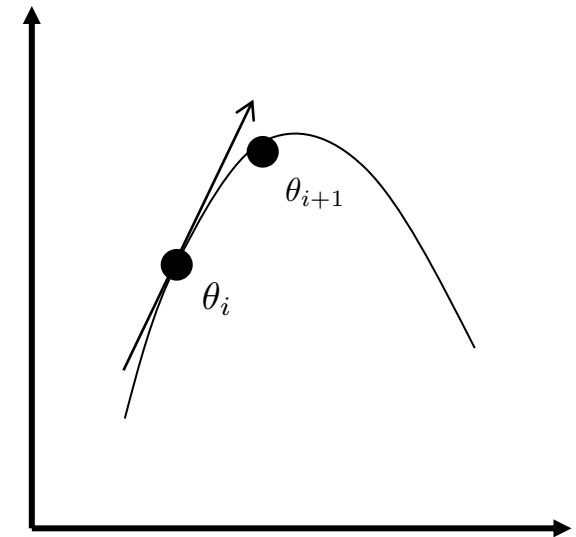
$$\nabla_{\theta} J(\theta) = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) d\tau \approx \frac{1}{N} \sum_{i=0}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \sum_{t'=0}^T r(s_{t'}^i, a_{t'}^i)$$

Gradient ascent is steepest ascent on linear approximation under the Euclidean metric!

$$\begin{aligned} \max \quad & J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i} (\theta - \theta_i) && \text{Linear approximation} \\ & (\theta - \theta_i)^T (\theta - \theta_i) \leq \epsilon && \text{Quadratic Constraint} \end{aligned}$$

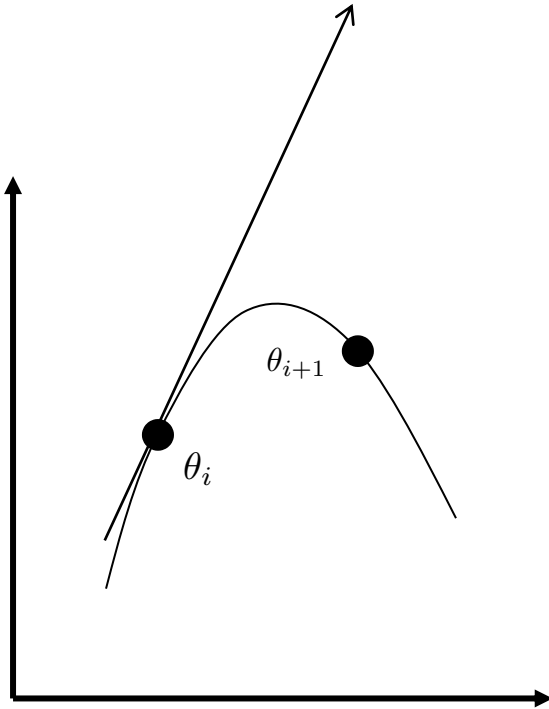


$$\theta = \theta_i + \alpha \nabla_{\theta} J(\theta)|_{\theta=\theta_i}$$



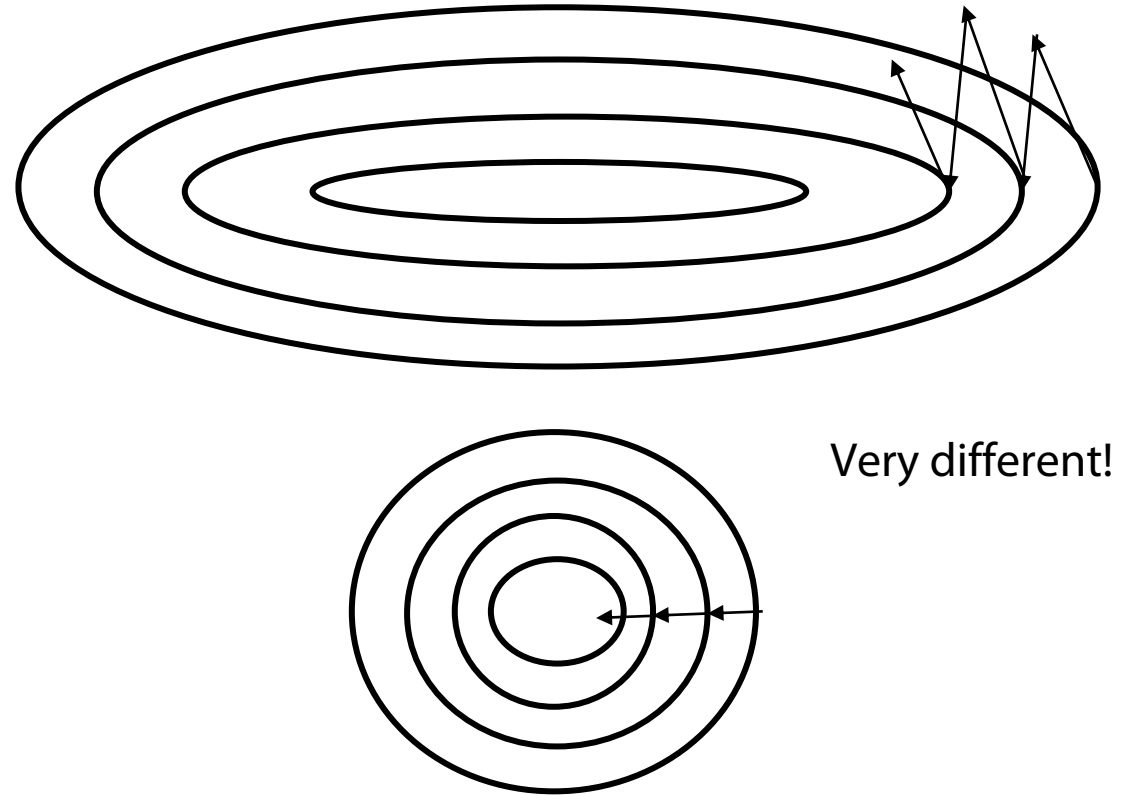
# When might this fail?

Large step sizes may cause collapse



Must use very small step sizes, slow!

Sensitive to Policy Parameterization



Can struggle for a deep neural network!

# Parameterization dependence of PG

## Sensitive to Policy Parameterization

$$L(\theta) = \theta_1 + \theta_2$$

$$\nabla_{\theta_1} L = 1$$

$$\nabla_{\theta_2} L = 1$$

$$L(\phi) = \phi_1^{0.5} + \phi_2^{-1}$$

$$\phi_1 = \theta_1^2$$

$$\phi_2 = \theta_2^{-1}$$

$$\nabla_{\phi_1} L = 0.5\phi_1^{-0.5} = 0.5\theta_1^{-1}$$

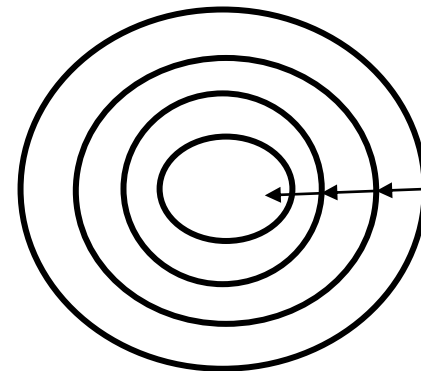
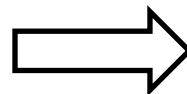
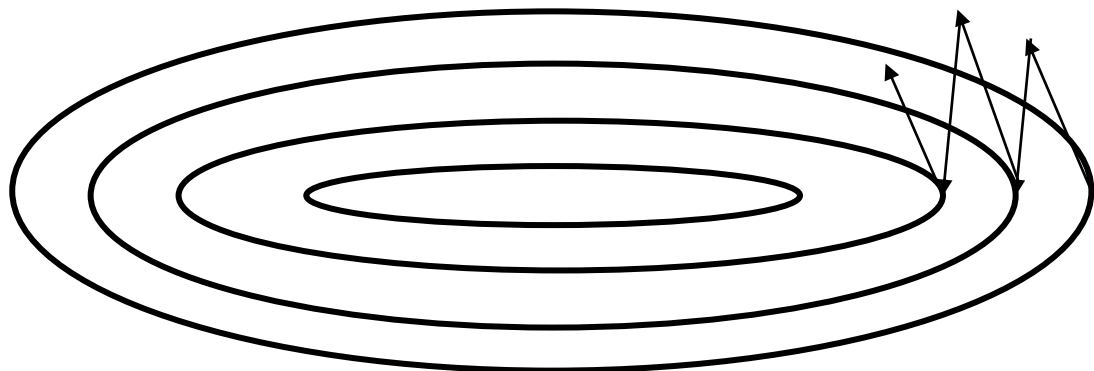
$$\nabla_{\phi_2} L = -\phi_2^{-2} = -\theta_2^2$$

←————→  
Not covariant!

# Modified Constraint on Policy Gradient

$$\begin{aligned} \max \quad & J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i}(\theta - \theta_i) \\ & (\theta - \theta_i)^T (\theta - \theta_i) \leq \epsilon \end{aligned}$$

$$\begin{aligned} \max \quad & J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i}(\theta - \theta_i) \\ & (\theta - \theta_i)^T G(\theta - \theta_i) \leq \epsilon \end{aligned}$$



$$\theta_{i+1} = \theta_i + \alpha G^{-1} \nabla_{\theta} J(\theta)|_{\theta=\theta_i}$$

Rescales according to  $G^{-1}$

Adaptive choice of  $G$  can avoid sensitivity to policy parameterization!

# Covariant Policy Gradient Updates

---

$$\begin{aligned} \max \quad & J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i} (\theta - \theta_i) \\ & (\theta - \theta_i)^T G (\theta - \theta_i) \leq \epsilon \end{aligned}$$

What should G be?

$$\begin{aligned} \max \quad & J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i} (\theta - \theta_i) \\ & D_{\text{KL}}(\pi_{\theta} || \pi_{\theta_i}) \leq \epsilon \end{aligned}$$

Let us use the constraint as  
KL divergence on the policy  
(2<sup>nd</sup> order Taylor expansion)

Measures functional distance, not parameter distance

# Resulting “Natural” Policy Gradient

$$\max J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i} (\theta - \theta_i)$$

$$D_{\text{KL}}(\pi_{\theta} || \pi_{\theta_i}) \leq \epsilon$$

2<sup>nd</sup> order approximation of KL  $\rightarrow$  Fisher Information Metric

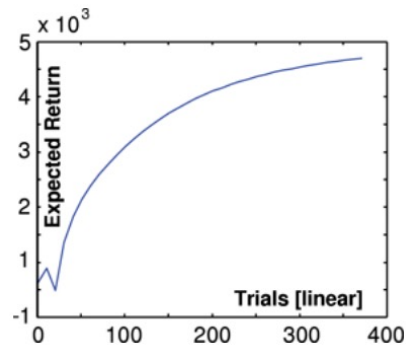
$$F = \mathbb{E}_{\pi_{\theta}} [(\nabla_{\theta} \log \pi_{\theta})(\nabla_{\theta} \log \pi_{\theta})^T]$$

$$\max J(\theta_i) + \nabla_{\theta} J(\theta)|_{\theta=\theta_i} (\theta - \theta_i)$$

$$(\theta - \theta_i)^T F (\theta - \theta_i) \leq \epsilon$$

Resulting update  $\theta_{i+1} = \theta_i + \alpha F^{-1} \nabla_{\theta} J(\theta)|_{\theta=\theta_i}$  Covariant to parameterization

# Natural Policy Gradient in Action



(a) Performance.



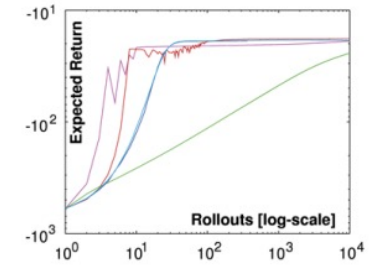
(b) Imitation learning.



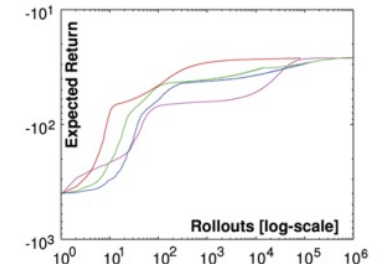
(c) Initial reproduction.



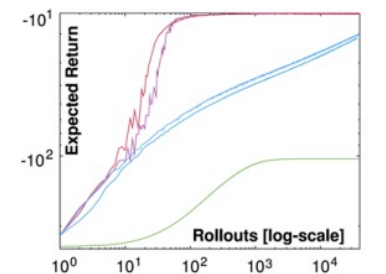
(d) After reinforcement learning.



(b) Minimum motor command with motor primitives



(c) Passing through a point with splines



(d) Passing through a point with motor primitives

- Finite Difference Gradient
- Vanilla Policy Gradient with constant baseline
- Vanilla Policy Gradient with time-variant baseline
- Episodic Natural Actor-Critic with single offset basis functions
- Episodic Natural Actor-Critic with time-variant offset basis functions

# Lecture outline

---

**Recap: Deriving the Policy Gradient**



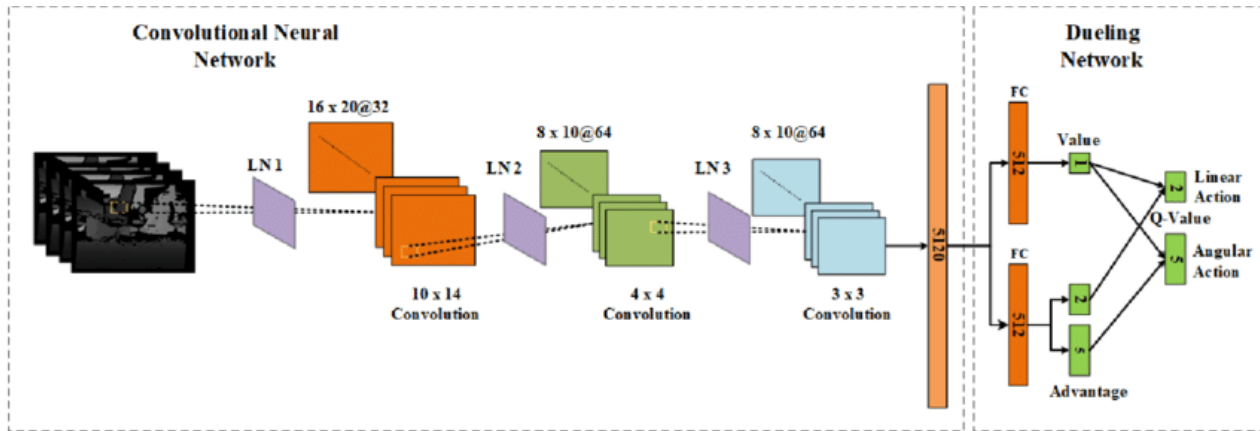
**What makes the Policy Gradient Challenging? - Variance**



**Natural Policy Gradients and Covariant Parameterization**

# Natural Policy Gradient - is it enough?

## Huge matrix inversion

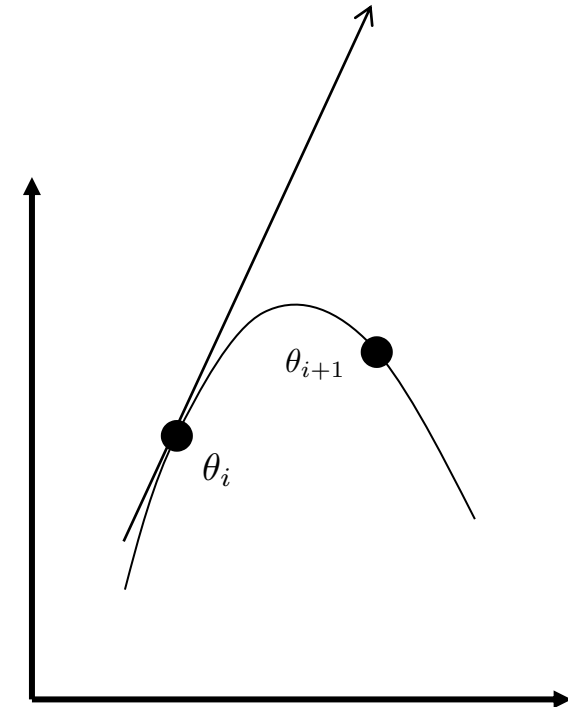


$$F - R^{d \times d}$$

For a standard convnet – d is in the millions

Hessian is way out of memory / hard to invert!

## Step-size?



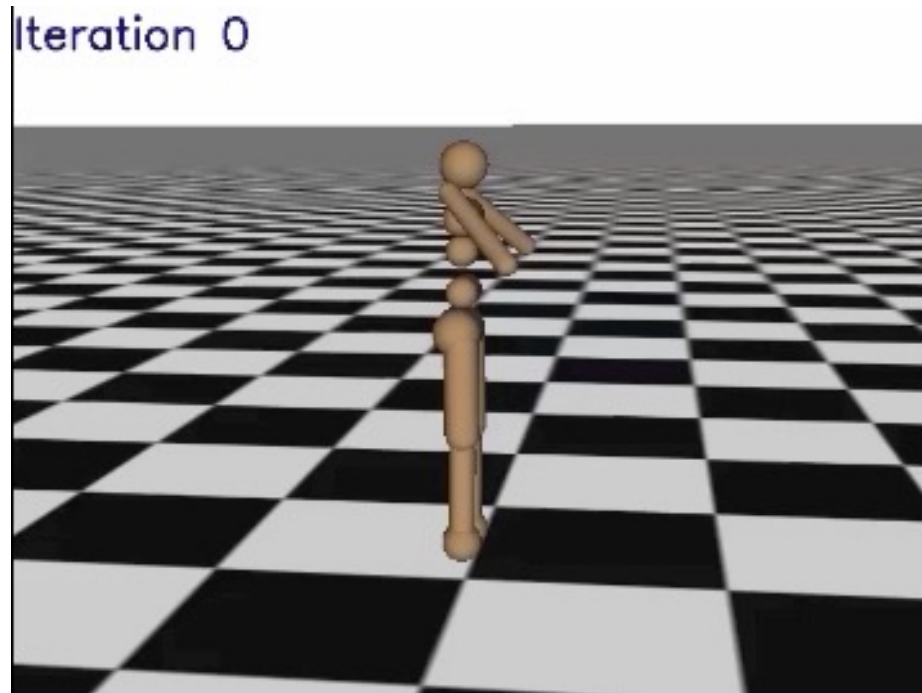
Can easily overstep and collapse performance

Also, only a single gradient step at a time before recollecting data!

# Trust Region Policy Optimization

3 key ideas:

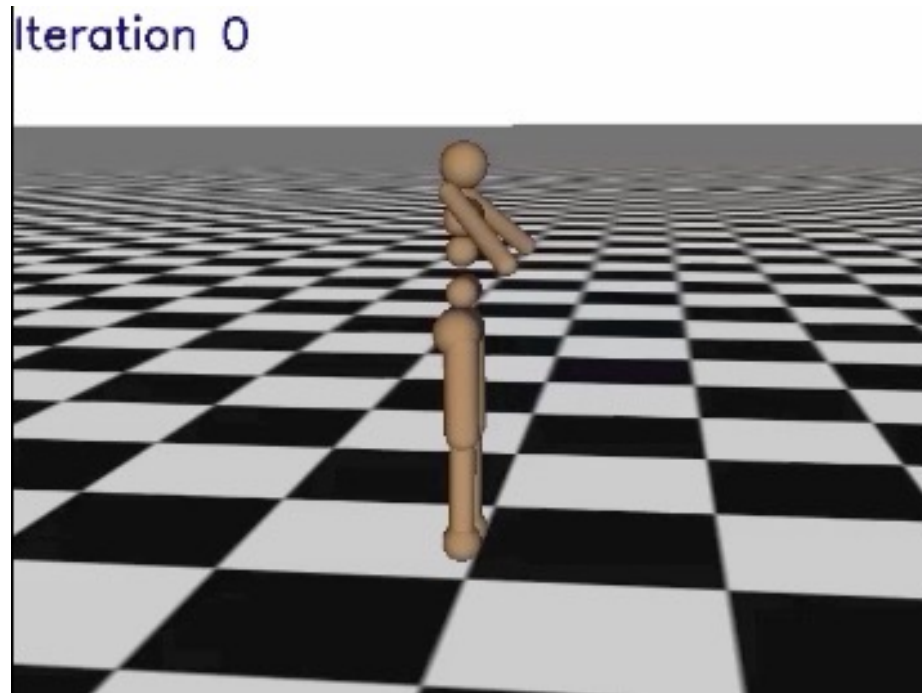
1. On-policy updates  $\rightarrow$  importance sampled objective
2. Huge matrix inversion  $\rightarrow$  conjugate gradient method
3. Step size may be too large  $\rightarrow$  backtracking line search



# Trust Region Policy Optimization

3 key ideas:

1. On-policy updates  $\rightarrow$  importance sampled objective
2. Huge matrix inversion  $\rightarrow$  conjugate gradient method
3. Step size may be too large  $\rightarrow$  backtracking line search



# Trust Region Policy Optimization – Importance Sampling

Cannot evaluate without resampling

Original Objective

$$J(\theta) = \mathbb{E}_{s \sim d_{\pi}(\theta), a \sim \theta} [A(s, a)]$$

$$= \mathbb{E}_{s \sim d_{\pi}(\theta), a \sim \theta_i} \left[ \frac{\pi_{\theta}}{\pi_{\theta_i}} A(s, a) \right]$$

Importance Sampling(ish)

$$\approx \mathbb{E}_{s \sim d_{\pi}(\theta_i), a \sim \theta_i} \left[ \frac{\pi_{\theta}}{\pi_{\theta_i}} A(s, a) \right]$$

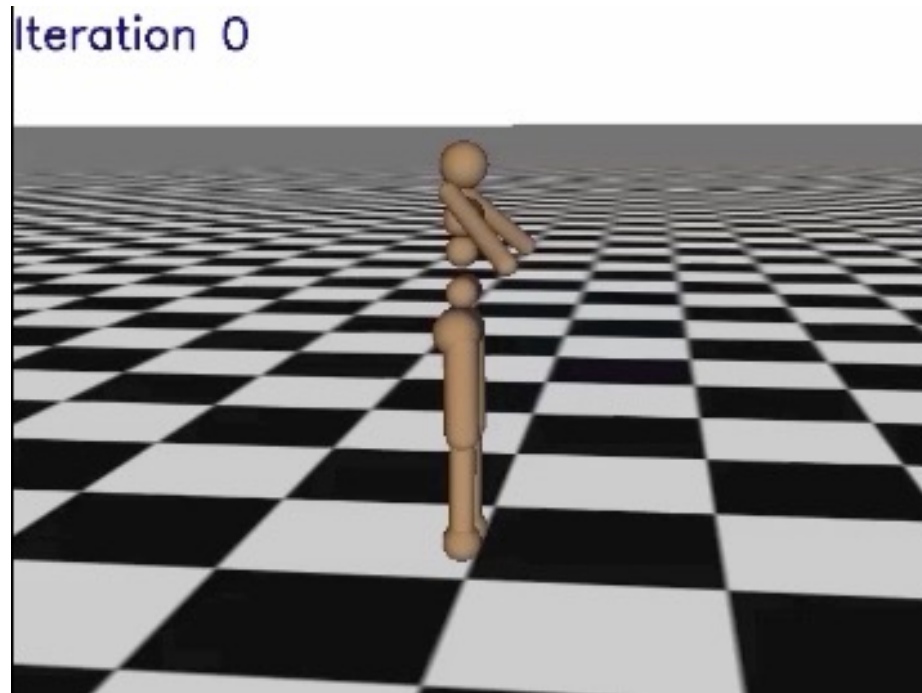


If policies are close, we can show that this is not so bad!

# Trust Region Policy Optimization

3 key ideas:

1. On-policy updates  $\rightarrow$  importance sampled objective
2. Huge matrix inversion  $\rightarrow$  conjugate gradient method
3. Step size may be too large  $\rightarrow$  backtracking line search



# Trust Region Policy Optimization – Conjugate Gradient

---

Challenging to compute  $F^{-1}$  and then get  $F^{-1}g$



Convert into an iterative minimization problem!

Solution to

$$Fx = g$$

same as

Solution to

$$\min_x \frac{1}{2} x^T F x - x^T g + c$$

# Trust Region Policy Optimization – Conjugate Gradient

Challenging to compute  $F^{-1}$  and then get  $F^{-1}g$

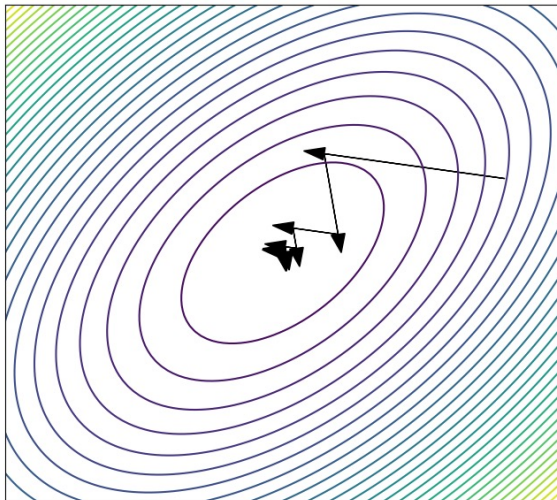


Convert into an iterative minimization problem!

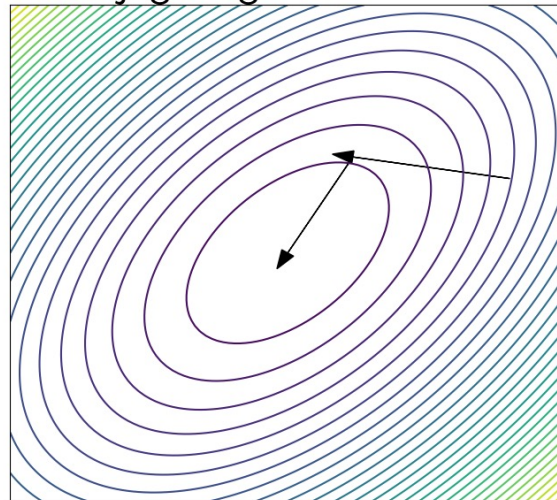


Solve with conjugate gradient

Gradient descent

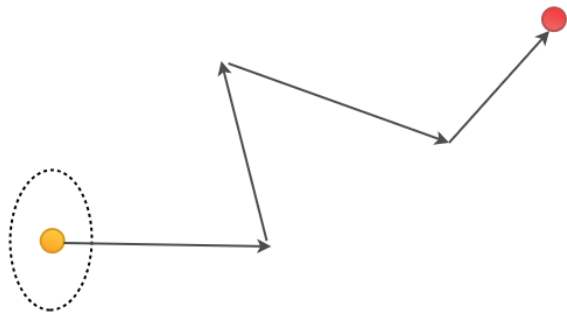


Conjugate gradient descent



Do coordinate descent in geometry aligned orthogonal directions

# Trust Region Policy Optimization – Conjugate Gradient



Gradient ascent

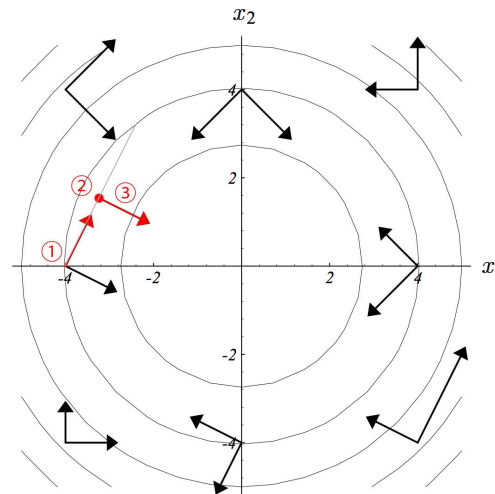
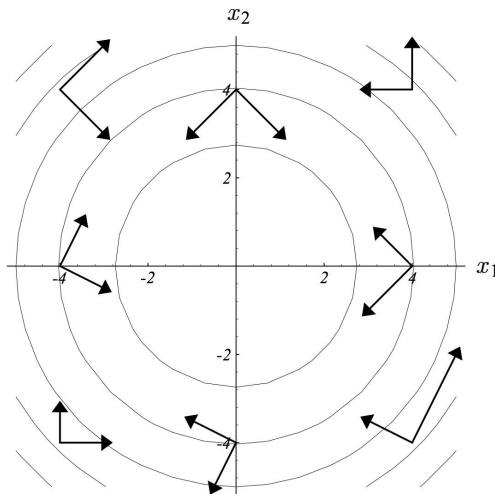


Conjugate gradient

$$\min_x \frac{1}{2} x^T F x - x^T g + c$$

Find search directions at every step that are F-orthogonal with previous directions

$$d_{(i)}^T F d_{(i)} = 0$$



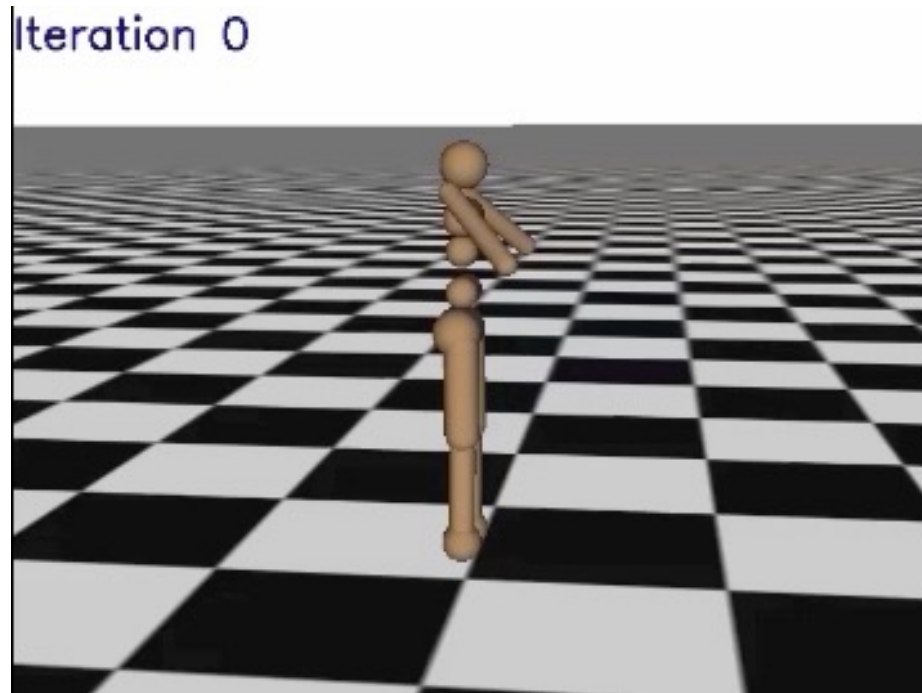
Converges in approx N steps!

Only requires matrix-vector product

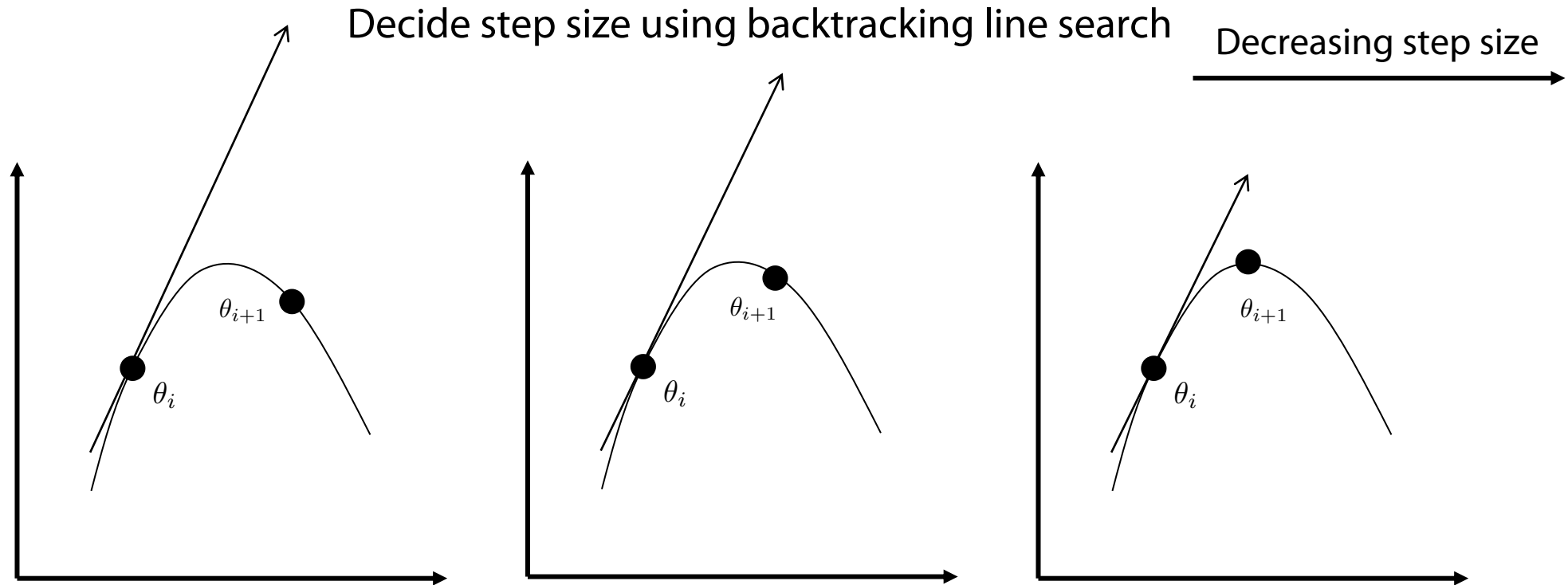
# Trust Region Policy Optimization

3 key ideas:

1. On-policy updates  $\rightarrow$  importance sampled objective
2. Huge matrix inversion  $\rightarrow$  conjugate gradient method
3. Step size may be too large  $\rightarrow$  backtracking line search



# Trust Region Policy Optimization – Backtracking line search



1. Choose parameter  $\beta \in (0, 1)$ , given search direction  $s = F^{-1}g$
2. Compute maximal step size such that constraint is satisfied -  $\frac{1}{2}(ts)^T F(ts) = \epsilon \rightarrow t = \sqrt{\frac{2\epsilon}{s^T F s}}$
3. While  $J(\theta_i + ts) < J(\theta_i)$ , set  $t = \beta t$

Backtracking

# Trust Region Policy Optimization

3 key ideas:

1. On-policy updates  $\rightarrow$  importance sampled objective
2. Huge matrix inversion  $\rightarrow$  conjugate gradient method
3. Step size may be too large  $\rightarrow$  backtracking line search

---

**Algorithm 3** Trust Region Policy Optimization

---

Input: initial policy parameters  $\theta_0$

**for**  $k = 0, 1, 2, \dots$  **do**

Collect set of trajectories  $\mathcal{D}_k$  on policy  $\pi_k = \pi(\theta_k)$

Estimate advantages  $\hat{A}_t^{\pi_k}$  using any advantage estimation algorithm

Form sample estimates for

- policy gradient  $\hat{g}_k$  (using advantage estimates)
- and KL-divergence Hessian-vector product function  $f(v) = \hat{H}_k v$

Use CG with  $n_{cg}$  iterations to obtain  $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

Estimate proposed step  $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

**end for**

# Can we say anything formal about updates?

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{\text{KL}}^{\max}(\pi, \tilde{\pi}),$$
$$\text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}.$$

Ensures that policies are non-decreasing in performance

Performance difference  
lemma

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \tilde{\pi}} \left[ \sum_{t=0}^{\infty} \gamma^t A_{\pi}(s_t, a_t) \right]$$

Express advantage  
in terms of TVD

**Theorem 1.** Let  $\alpha = D_{\text{TV}}^{\max}(\pi_{\text{old}}, \pi_{\text{new}})$ . Then the following bound holds:

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{4\epsilon\gamma}{(1-\gamma)^2} \alpha^2$$

where  $\epsilon = \max_{s,a} |A_{\pi}(s, a)|$  (8)

Key idea: by bounding how different the policies are, we can bound how different returns are

# TRPO in action

---

Trust Region Policy Optimization

# Why might TRPO not be enough?

---

## Algorithm 3 Trust Region Policy Optimization

---

Input: initial policy parameters  $\theta_0$

**for**  $k = 0, 1, 2, \dots$  **do**

Collect set of trajectories  $\mathcal{D}_k$  on policy  $\pi_k = \pi(\theta_k)$

Estimate advantages  $\hat{A}_t^{\pi_k}$  using any advantage estimation algorithm

Form sample estimates for

- policy gradient  $\hat{g}_k$  (using advantage estimates)
- and KL-divergence Hessian-vector product function  $f(v) = \hat{H}_k v$

Use CG with  $n_{cg}$  iterations to obtain  $x_k \approx \hat{H}_k^{-1} \hat{g}_k$

Estimate proposed step  $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$

Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

**end for**

---

Advantage estimation is too high variance

Optimization expensive/unstable

# Better Advantage Estimation - Generalized Advantage Estimation

Advantage estimator

$$A_N^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^{N-1} r_N - V(s_1)$$

High variance!

N step advantage estimator

$$A_N^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^{N-1} r_N - V(s_1)$$

N-1 step advantage estimator

$$A_{N-1}^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^{N-2} V(s_{N-1}) - V(s_1)$$

⋮

2 step advantage estimator

$$A_2^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^2 V(s_3) - V(s_1)$$

1 step advantage estimator

$$A_1^\theta(s_1, a_1) = r_1 + \gamma V(s_2) - V(s_1)$$

↑  
Variance

↓  
Bias

# Generalized Advantage Estimation

Sum up all the estimators in a geometric sum

$$A_N^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^{N-1} r_N - V(s_1)$$

$$A_{N-1}^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^{N-2} V(s_{N-1}) - V(s_1)$$

$$A_2^\theta(s_1, a_1) = r_1 + \gamma r_2 + \dots + \gamma^2 V(s_3) - V(s_1)$$

$$A_1^\theta(s_1, a_1) = r_1 + \gamma V(s_2) - V(s_1)$$

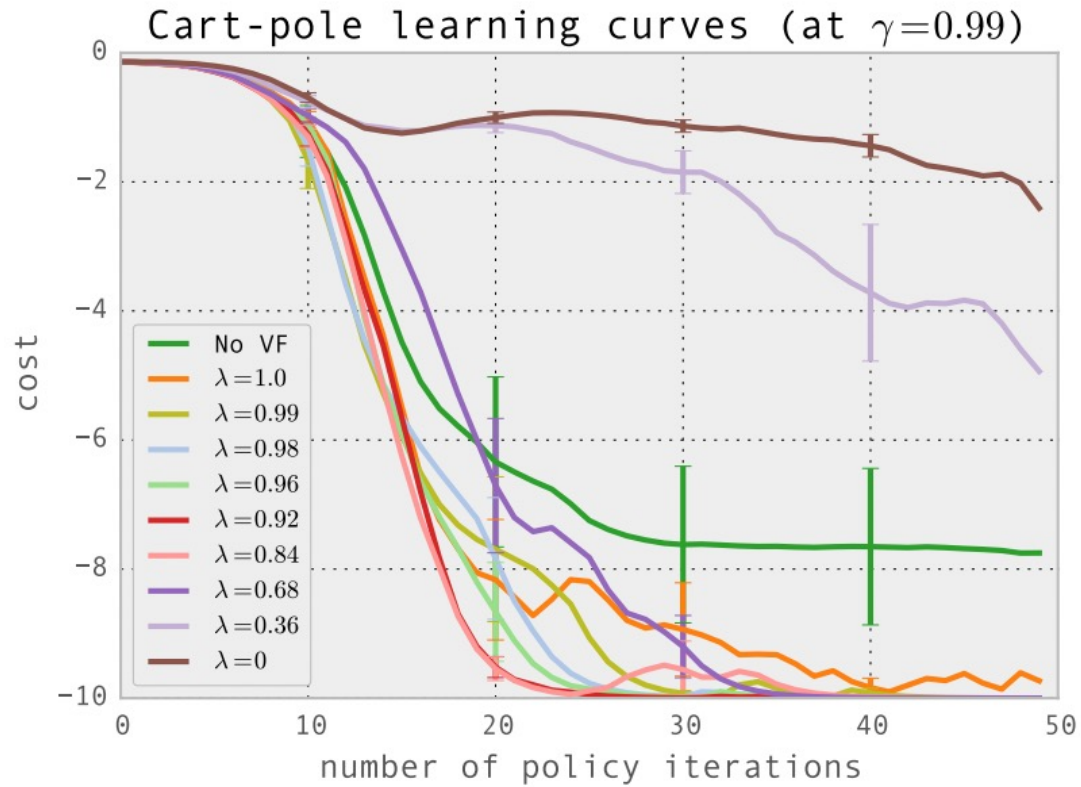
Geometric sum

$$A_\lambda^\theta(s_1, a_1) = \sum_{j=1}^N \lambda^j A_j^\theta(s, a)$$

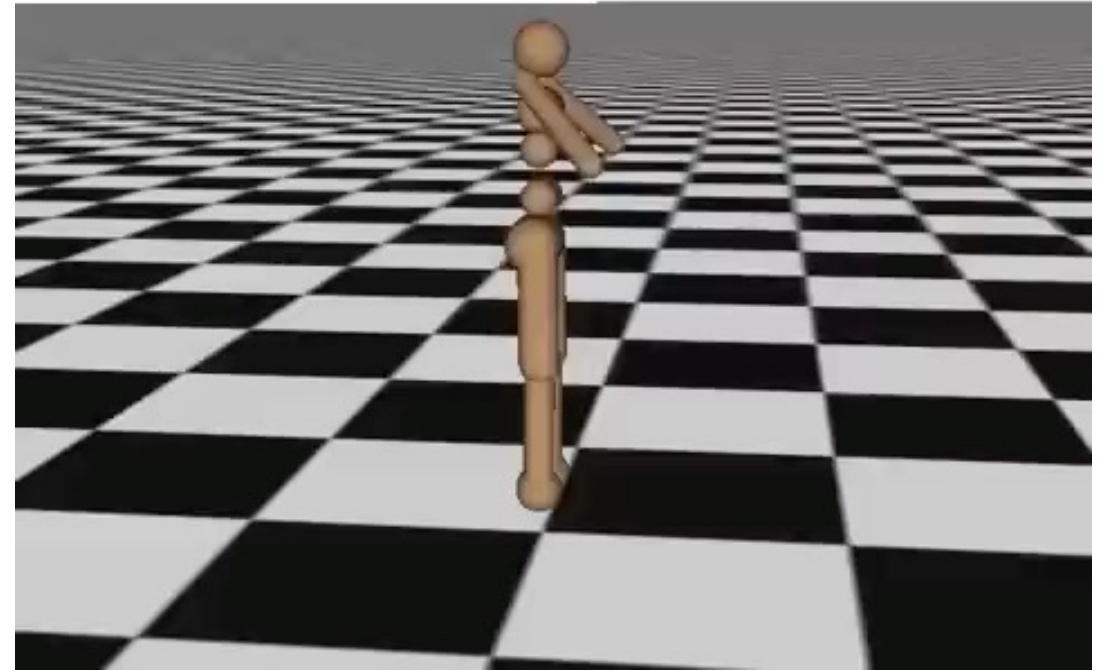
$\lambda$  controls bias-variance tradeoff

Best of both worlds – very similar idea to eligibility traces

# Generalized Advantage Estimation in Action



Iteration 0



---

More on the empirical details next week!

Fin.

