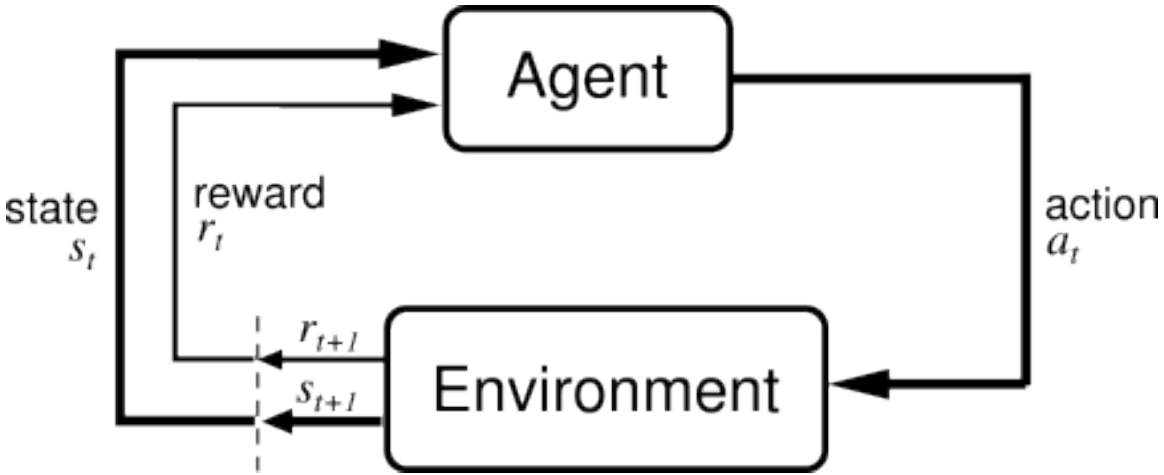
Derivative-free and Evolutionary Methods

Xi (Peter) Chen – UC Berkeley

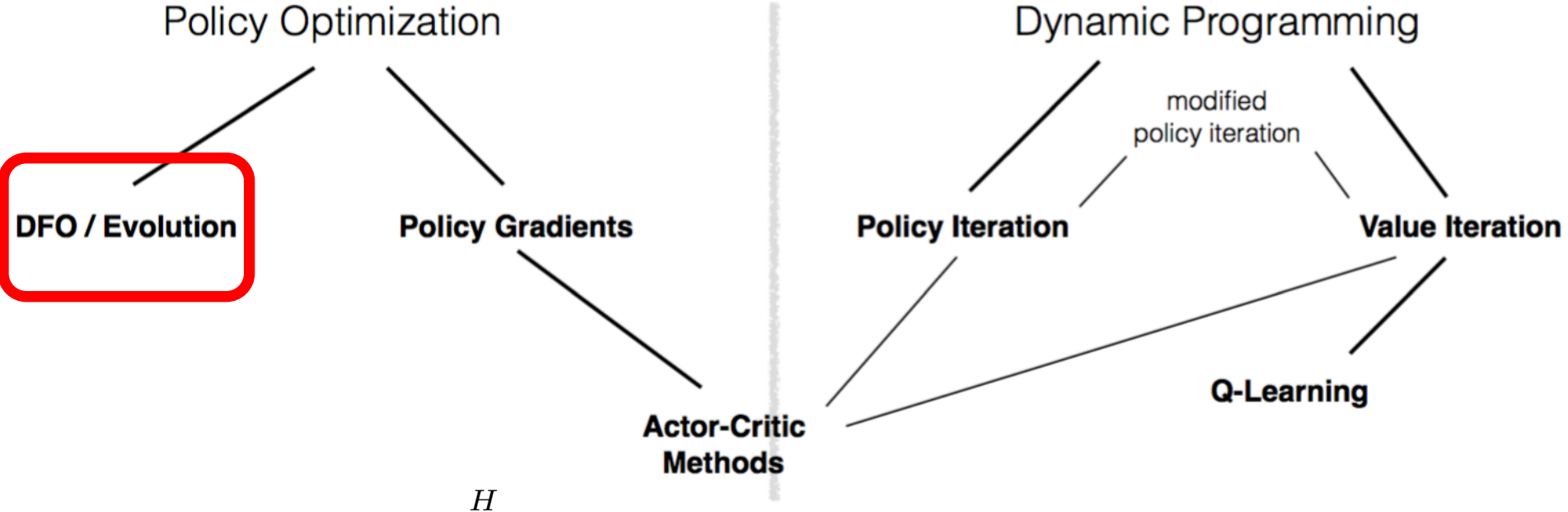
Slides authored with John Schulman (OpenAI) and Pieter Abbeel (OpenAI / UC Berkeley)

Reinforcement Learning



[Figure source: Sutton & Barto, 1998]

Policy Optimization in the RL Landscape



$$\max_\theta \; U(\theta) = \max_\theta \; \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

Evolutionary Methods

$$\max_{\theta} \; U(\theta) = \max_{\theta} \; \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

- General Algorithm:
  - Make some random change to the parameters
  - If the result improves, keep the change
  - Repeat

Cross-Entropy Method

CEM:

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d_{>0}$

for iteration = 1, 2, …

Sample n parameters $\theta_i \sim N(\mu, \mathrm{diag}(\sigma^2))$

For each θ_i, perform one rollout to get return $R(\tau_i)$

Select the top k% of θ, and fit a new diagonal Gaussian to those samples. Update μ, σ

endfor

Cross-Entropy Method

- Can work embarrassingly well

Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941

$$\mu \in \mathbb{R}^{22}$$

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon
INRIA Lille - Nord Europe, Team SequeL, FRANCE
victor.gabillon@inria.fr

Mohammad Ghavamzadeh*
INRIA Lille - Team SequeL & Adobe Research
mohammad.ghavamzadeh@inria.fr

Bruno Scherrer
INRIA Nancy - Grand Est, Team Maia, FRANCE
bruno.scherrer@inria.fr

[NIPS 2013]

Closely Related Approaches

CEM:

for iter i = 1, 2, …

for population member e = 1, 2, …

sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$

execute roll-outs under $\pi_{\theta^{(e)}}$

store $(\theta^{(e)}, U(e))$

endfor

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{(\bar{e})})$$

where $\bar{e}$ indexes over top p %

endfor

- Reward Weighted Regression (RWR)
  - Dayan & Hinton, NC 1997; Peters & Schaal, ICML 2007

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} q(U(e), P_{\mu}(\theta^{(e)})) \log P_{\mu}(\theta^{(e)})$$

- Policy Improvement with Path Integrals (PI^2)
  - PI2: Theodorou, Buchli, Schaal JMLR2010; Kappen, 2007; (PI2-CMA: Stulp & Sigaud ICML2012)

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} \exp(\lambda U(e)) \log P_{\mu}(\theta^{(e)})$$

- Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)
  - CMA: Hansen & Ostermeier 1996; (CMA-ES: Hansen, Muller, Koumoutsakos 2003)

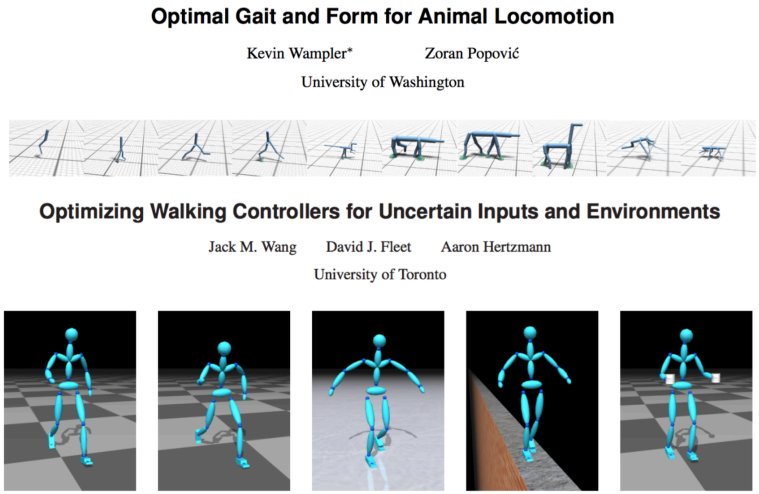
$$(\mu^{(i+1)}, \Sigma^{(i+1)}) = \arg\max_{\mu,\Sigma} \sum_{\bar{e}} w(U(\bar{e})) \log \mathcal{N}(\theta^{(\bar{e})}; \mu, \Sigma)$$

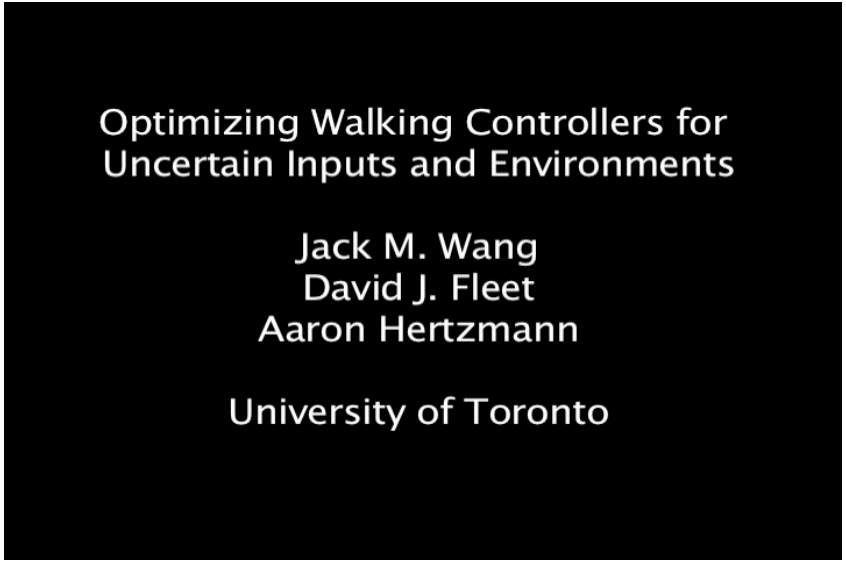
- PoWER
  - Kober & Peters, NIPS 2007 (also applies importance sampling for sample re-use)

$$\mu^{(i+1)} = \mu^{(i)} + \left(\sum_{e} (\theta^{(e)} - \mu^{(i)}) U(e) \right) / \left(\sum_{e} U(e) \right)$$

CMA-ES

Covariance Matrix Adaptation (CMA) has become standard in graphics [Hansen, Ostermeier, 1996]





$\mu \in \mathbb{R}^{120}$

However, DRL Humanoid policy usually have 100k+ params

Outline

- Evolutionary methods
  - Cross-Entropy Method works well on relatively low-dim problems
  - **Can ES work well on optimizing deep net policies?**

Learning a Distribution of Policies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
  - Goal: maximize $\mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta}[R(\tau)]$
  - Stochastic Gradient Ascent by Policy Gradient/LR:

$$\nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta}[\nabla_\mu \log P_\mu(\theta) R(\tau)]$$

  - Ignore most information about trajectory τ: states, actions, rewards..
    - Only specific params θ and return R(τ) are used
    - This turns out to enable very scalable distributed training (more on this later)

A Concrete Example

- Suppose $\theta \sim P_\mu(\theta)$ is a Gaussian distribution with mean μ, and covariance matrix $\sigma^2 I$

$$\log P_\mu(\theta) = -\frac{||\theta - \mu||^2}{2\sigma^2} + \text{const}$$

$$\nabla_\mu \log P_\mu(\theta) = \frac{\theta - \mu}{\sigma^2}$$

- If we draw two parameter samples θ_1, θ_2, and obtain two trajectories τ_1, τ_2:

$$\mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta} \left[\nabla_\mu \log P_\mu(\theta) R(\tau)\right] \approx \frac{1}{2}\left[R(\tau_1)\frac{\theta_1 - \mu}{\sigma^2} + R(\tau_2)\frac{\theta_2 - \mu}{\sigma^2}\right]$$

Connection to Finite Difference

- Antithetic sampling
  - Sample a pair of policies with mirror noise $(\theta_+ = \mu + \sigma\epsilon, \theta_- = \mu - \sigma\epsilon)$
  - Get a pair of rollouts from environment (τ_+, τ_-)
  - SPSA: Finite Difference with random direction

$$\nabla_\mu \mathbb{E}\left[R(\tau)\right] \approx \frac{1}{2}\left[R(\tau_+)\frac{\theta_+ - \mu}{\sigma^2} + R(\tau_-)\frac{\theta_- - \mu}{\sigma^2}\right]$$
$$= \frac{1}{2}\left[R(\tau_+)\frac{\sigma\epsilon}{\sigma^2} + R(\tau_-)\frac{-\sigma\epsilon}{\sigma^2}\right]$$
$$= \frac{\epsilon}{2\sigma}\left[R(\tau_+) - R(\tau_-)\right]$$

vs

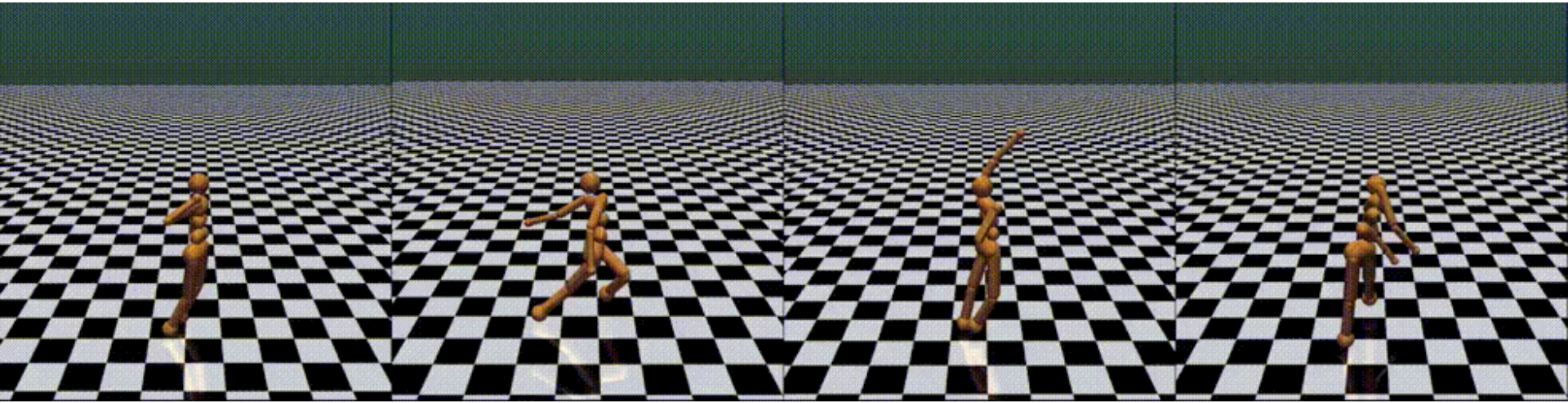
Finite Difference

$$\frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon e_j) - U(\theta - \epsilon e_j)}{2\epsilon}$$

Practical Tricks

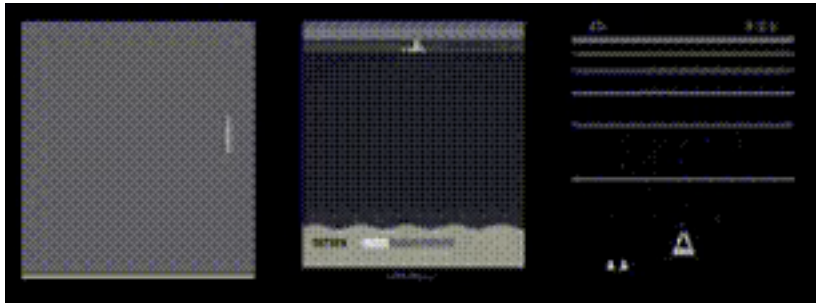
- Loss Transformation to be robust against reward scaling
  - Rank-based: $R(\tau) = \text{percentile of } \tau$
  - Learn from the best: $R(\tau) = \mathbb{1}[\tau \text{ in best k\%}]$
    - → Cross-Entropy Method!
- Parametrizations matter
  - Normal deep nets that work with SGD might not work well with ES [Duan et al., ICML 2016]
  - Virtual Batch Norm leads to competitive performance on modern Deep RL benchmark [Salimans et al., NIPS 2016]
  - Other tricks that helped: discretization of continuous action space, DenseNet [Huang et al., 2016]...

ES Competitive on DeepRL Tasks



[Salimans, Ho, Chen, Sutskever, 2017]

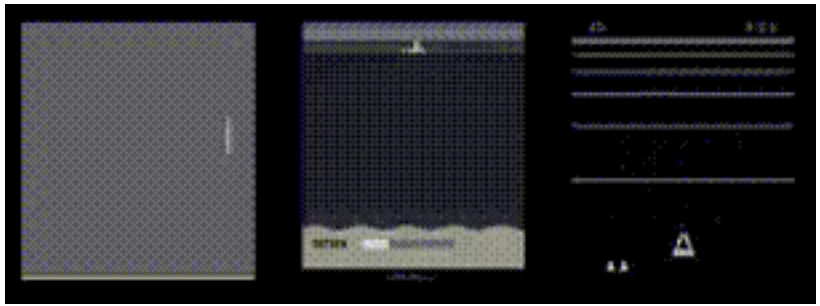
Considerations



- Pros:
  - Work with arbitrary parameterization, even non-differentiable
  - Embarrassingly easy to parallelize
- Cons:
  - Not very sample efficient since ignores temporal structure
  - Harder to tune

[Salimans, Ho, Chen, Sutskever, 2017]

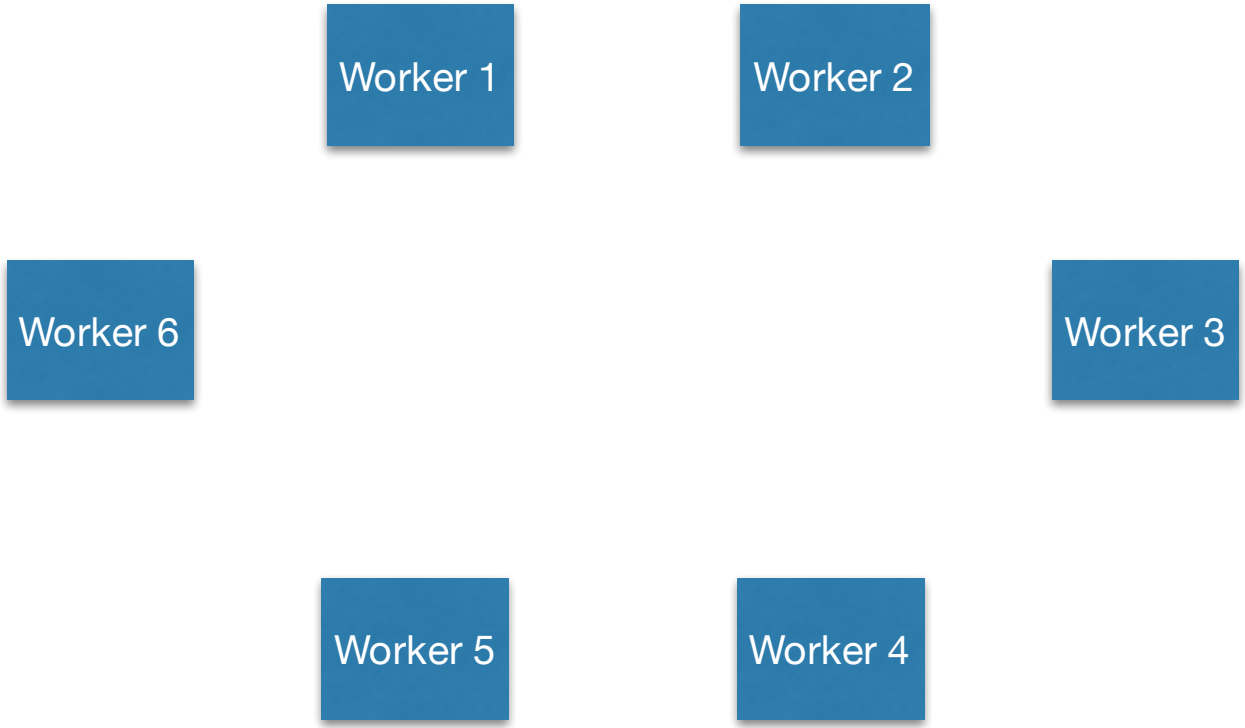
Considerations



- Pros:
  - Work with arbitrary parameterization, even non-differentiable
  - **Embarrassingly easy to parallelize**
- Cons:
  - Not very sample efficient since ignores temporal structure
  - Harder to tune

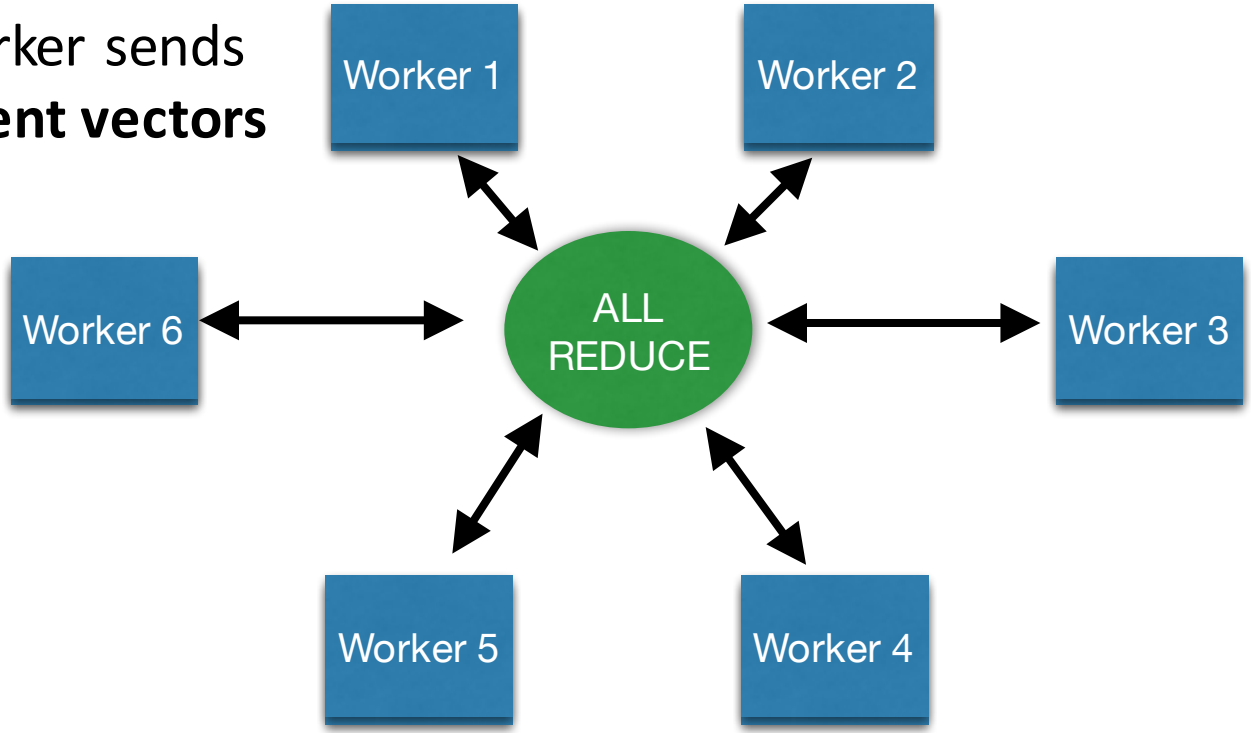
[Salimans, Ho, Chen, Sutskever, 2017]

Distributed Deep Learning

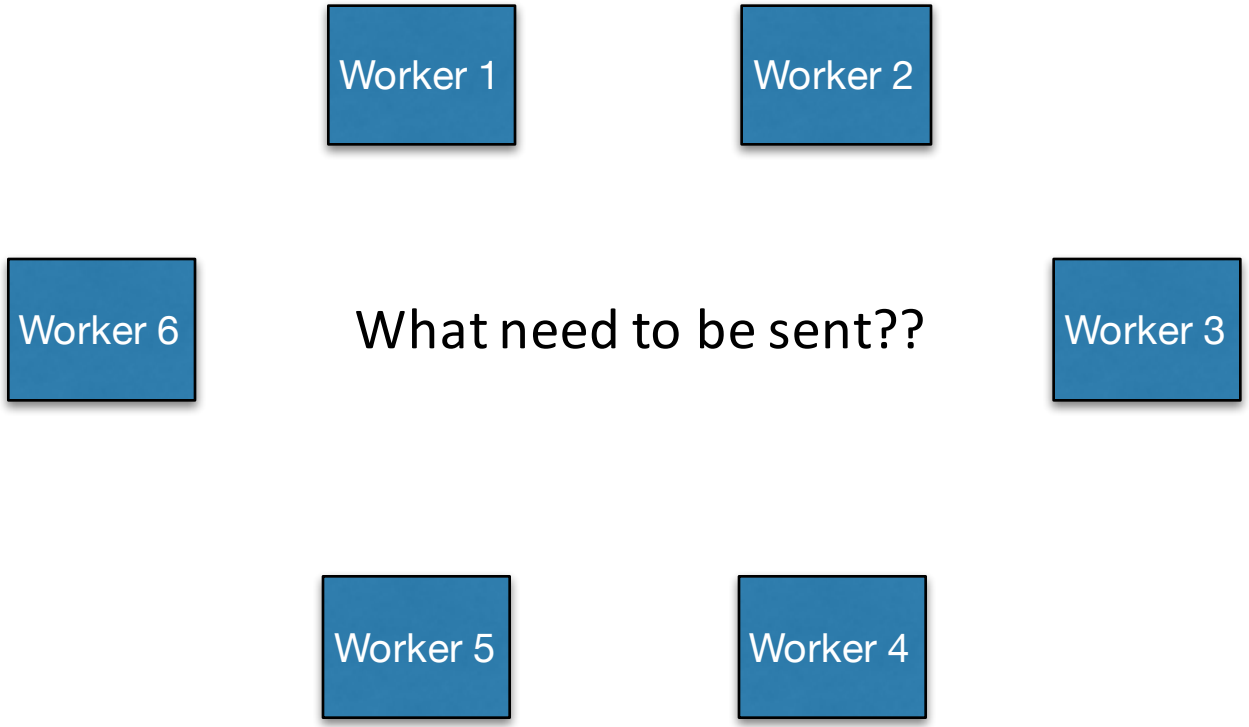


Distributed Deep Learning

Each worker sends
big gradient vectors



Distributed Evolution



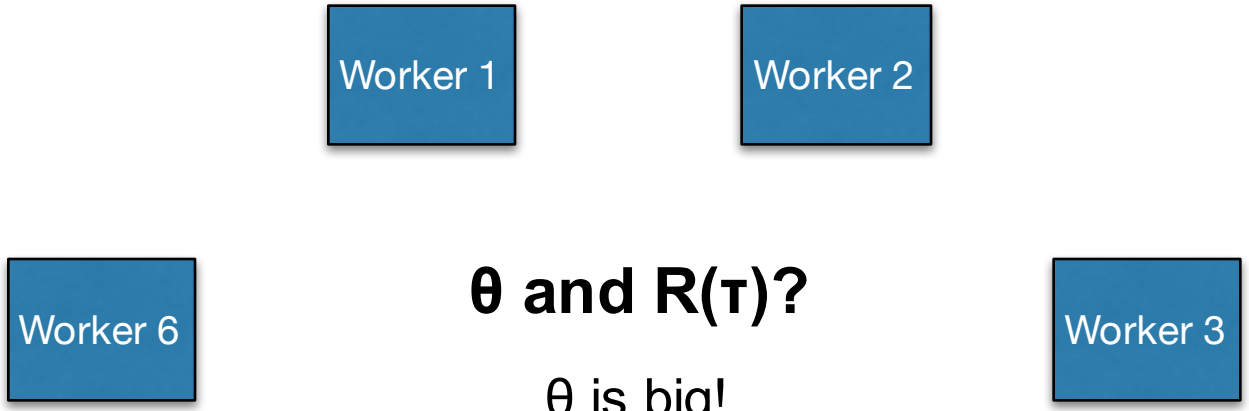
Recap: Learning a Distribution of Policies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
  - Goal: maximize $\mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta}[R(\tau)]$
  - Stochastic Gradient Ascent by Policy Gradient/LR:

$$\nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\theta \sim P_\mu(\theta), \tau \sim \pi_\theta}[\nabla_\mu \log P_\mu(\theta) R(\tau)]$$

  - Ignore most information about trajectory τ: states, actions, rewards..
    - **Only specific params θ and return R(τ) are used**
    - This turns out to enable very scalable distributed training (more on this later)

Distributed Evolution



Scalability

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

[Salimans, Ho, Chen, Sutskever, 2017]

Scalability

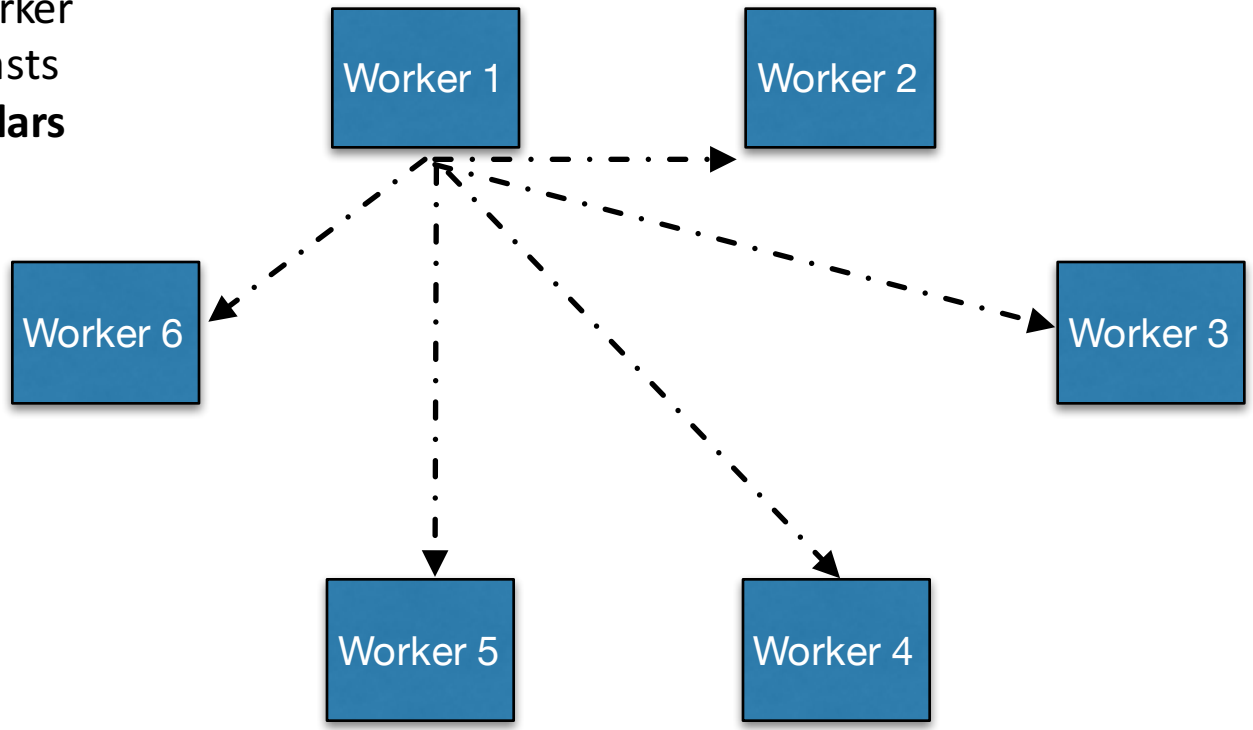
Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

[Salimans, Ho, Chen, Sutskever, 2017]

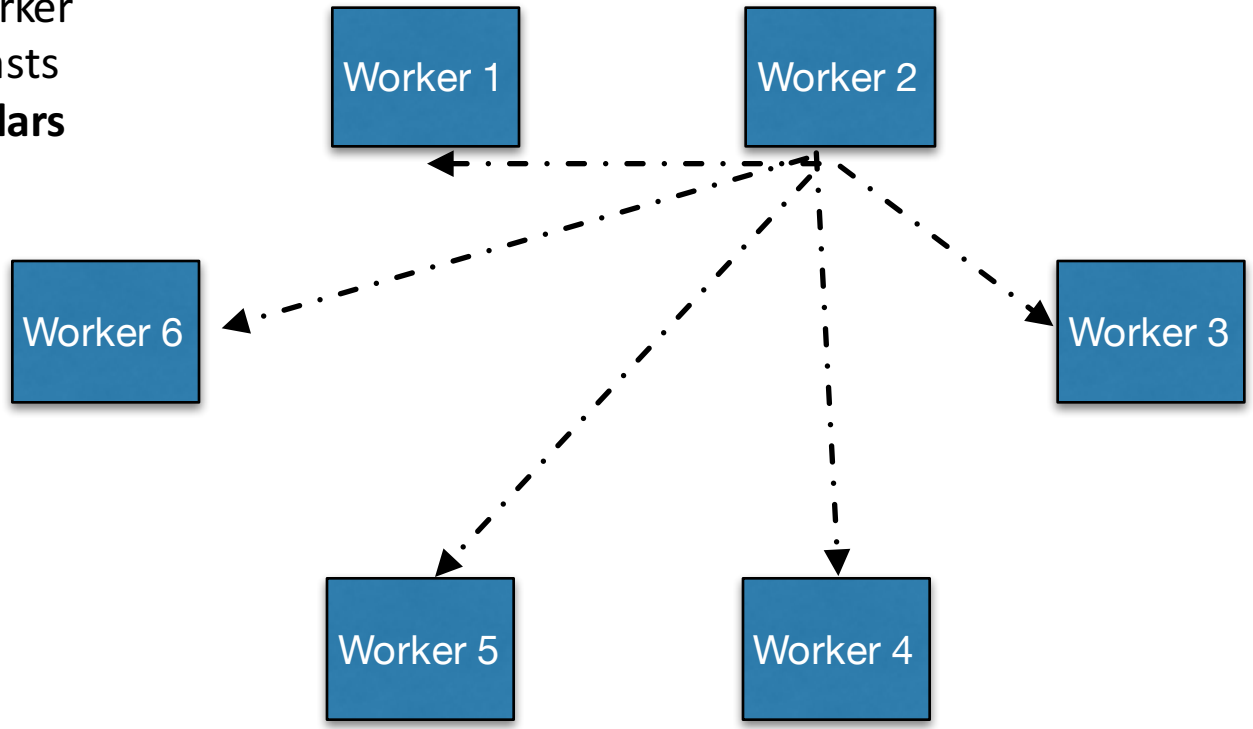
Distributed Evolution

Each worker broadcasts **tiny scalars**



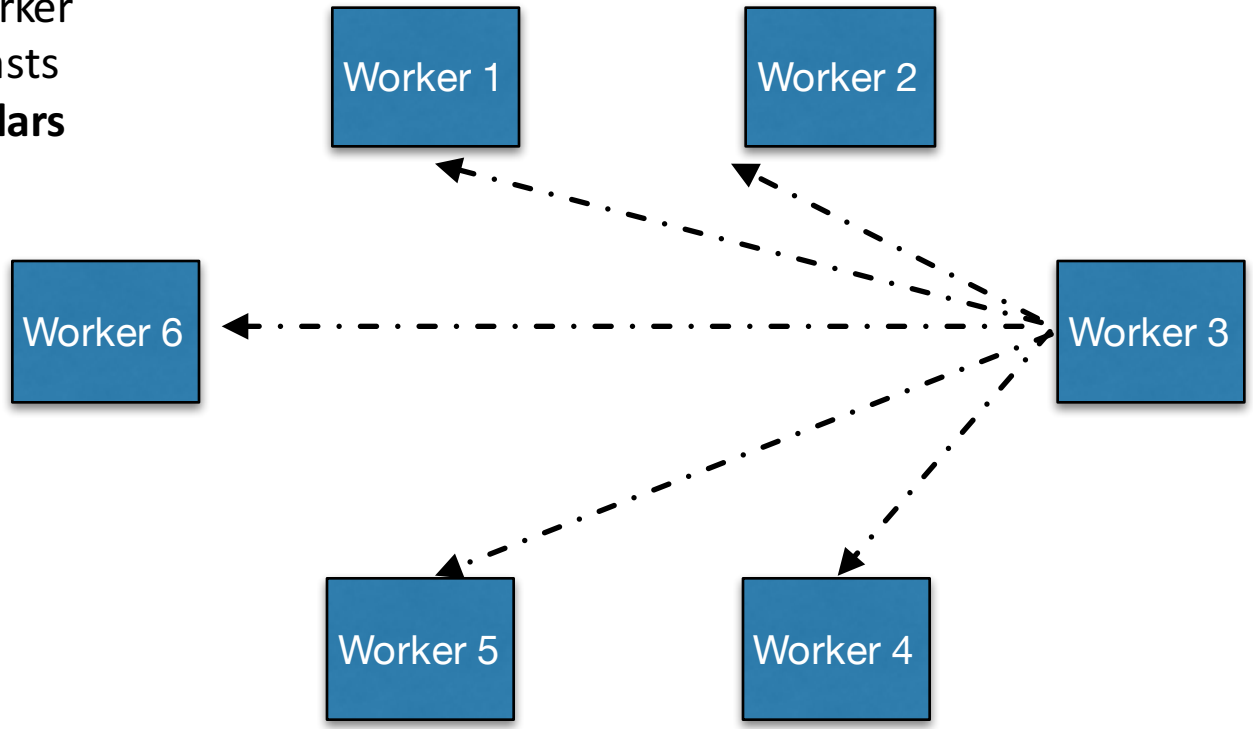
Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution

Each worker broadcasts **tiny scalars**



Pro: Scalability

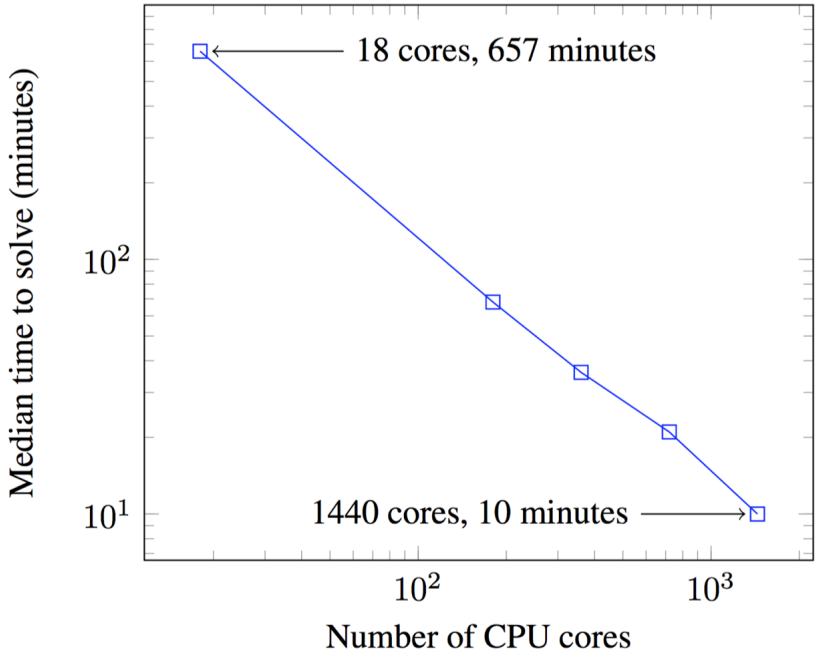
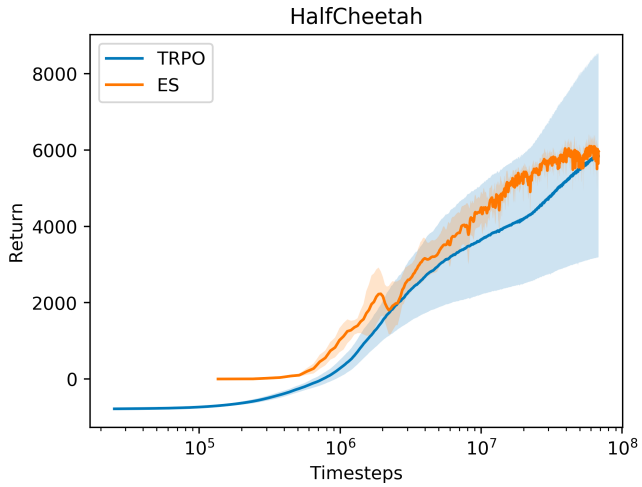
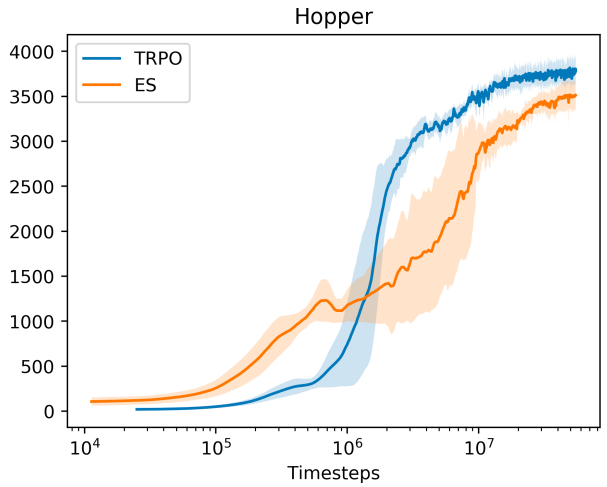


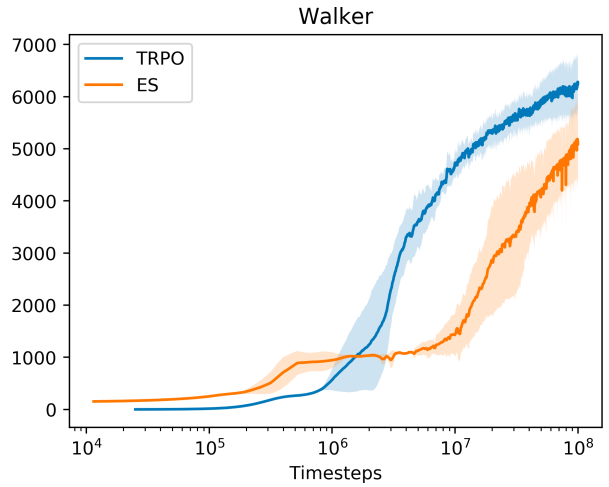
Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

[Salimans, Ho, Chen, Sutskever, 2017]

Con: Sample Efficiency







[Salimans, Ho, Chen, Sutskever, 2017]

Cross-Entropy / Evolutionary Methods

- Full episode evaluation, parameter perturbation
  - More structured exploration
  - Not biased by discount factor
  - Easy to scale
- Open Question: Policy Gradient at action level or parameter level?
  - Important to remember both are doing finite difference / random search
  - When episode is very long, maybe finite difference in parameter space is more efficient?
  - Maybe there is a better space to be discovered

Outline

- Evolutionary methods
- **Finite difference gradients**

Black Box Gradient Computation

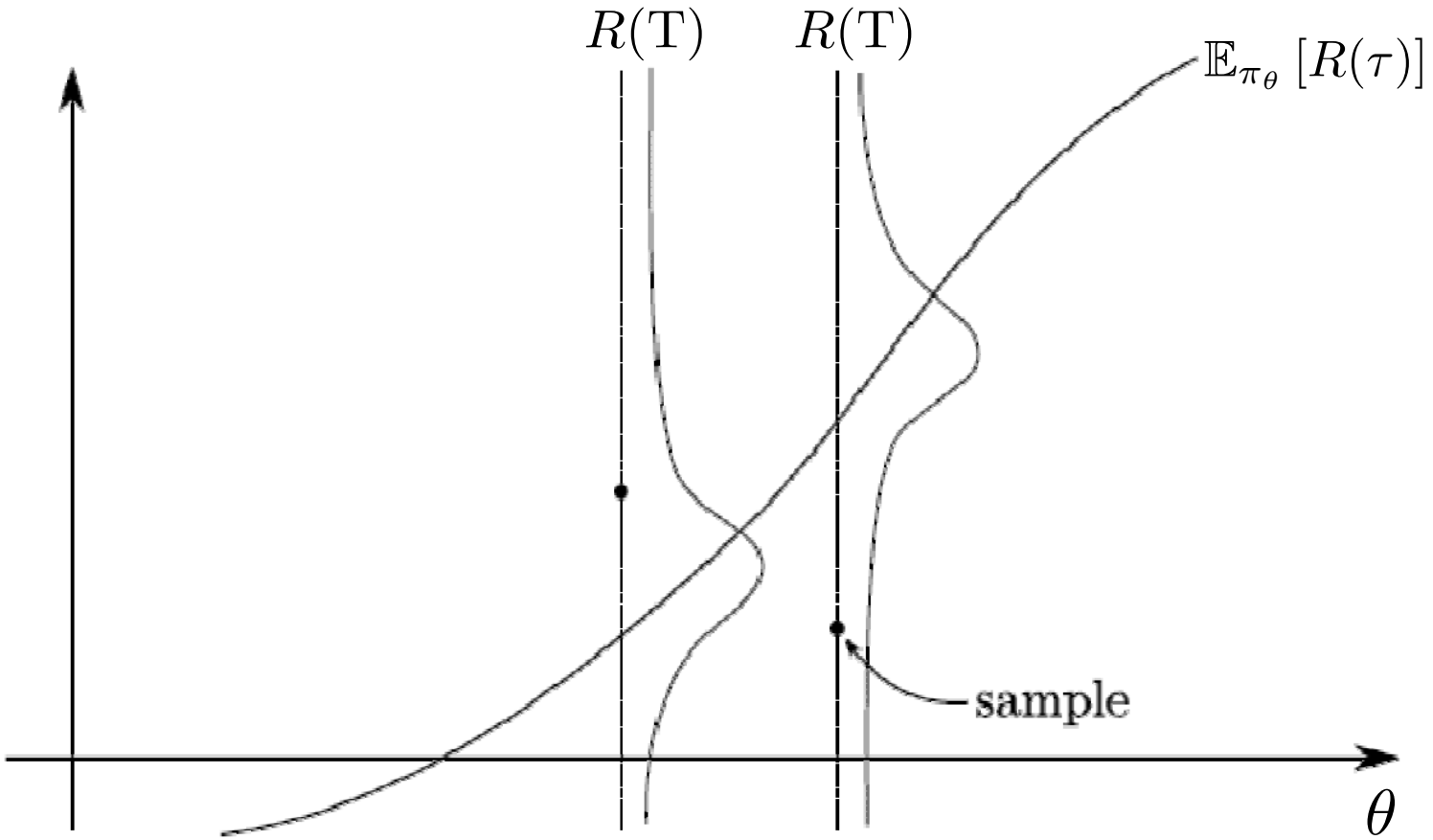
We can compute the gradient g using standard finite difference methods, as follows:

$$\frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon e_j) - U(\theta - \epsilon e_j)}{2\epsilon}$$

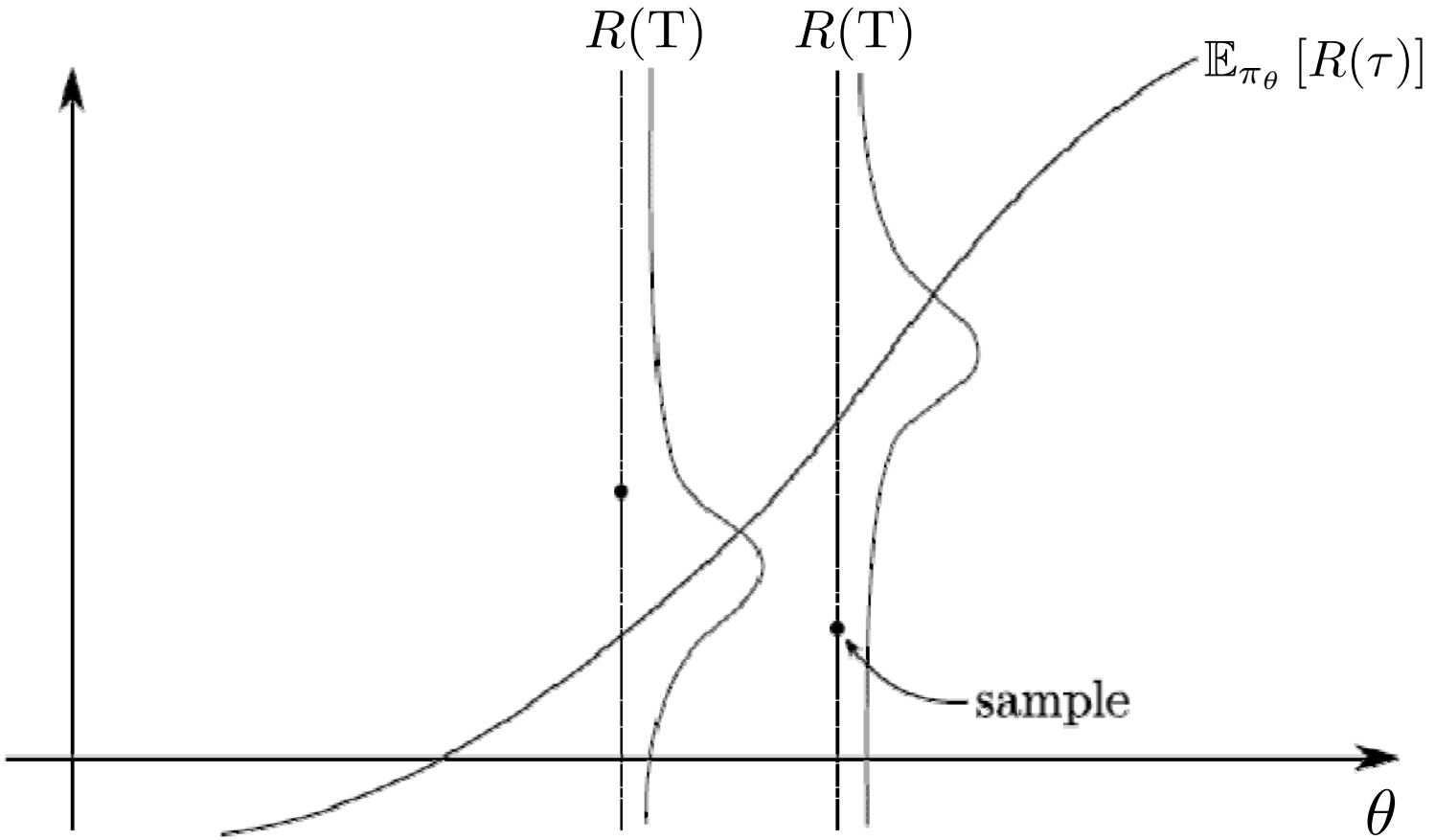
Where:

$$e_j = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow j\text{'th entry}$$

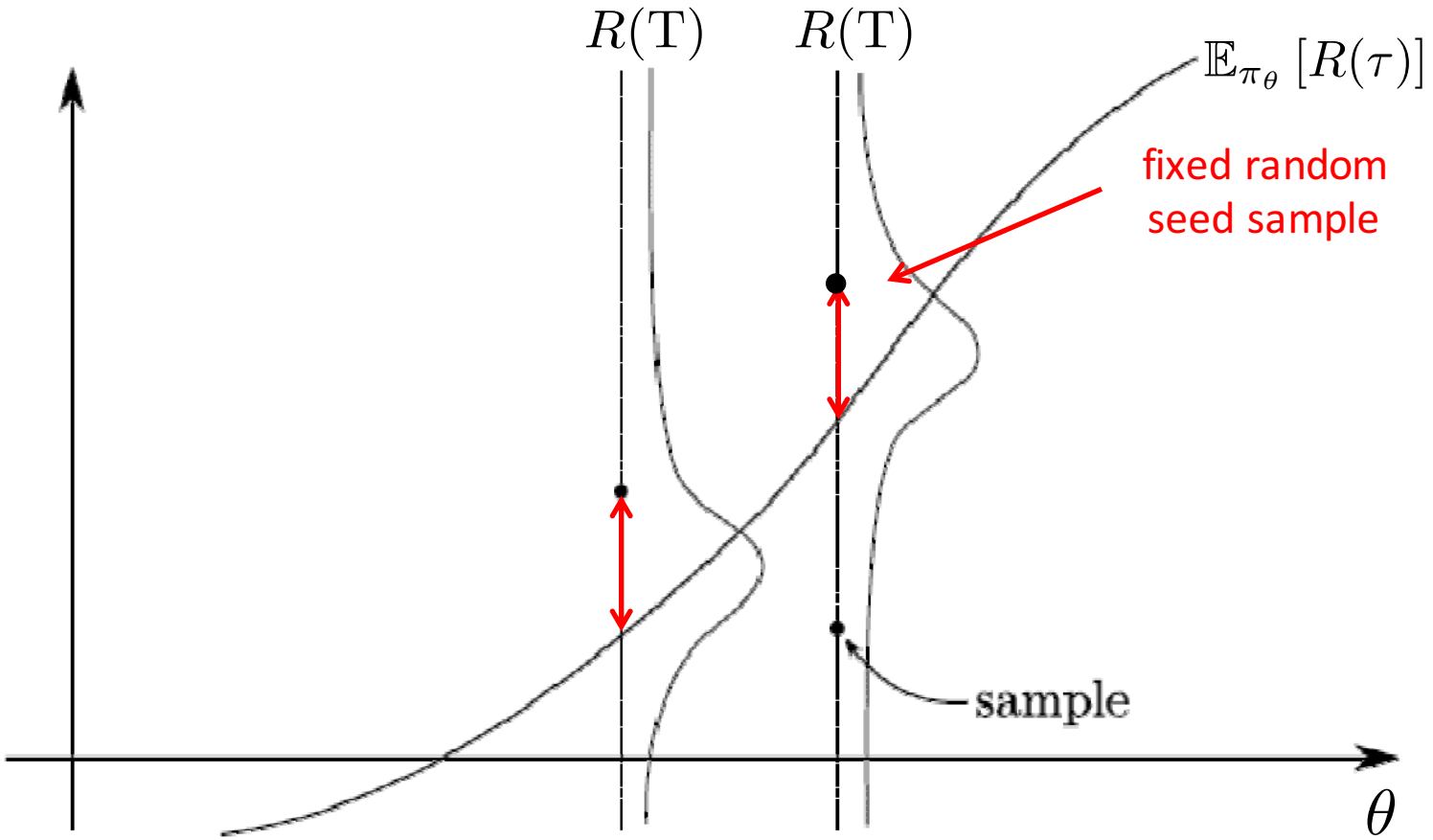
Challenge: Noise Can Dominate



Solution 1: Average over many samples



Solution 2: Fix random seed



Solution 2: Fix random seed

- Randomness in policy and dynamics
  - But can often only control randomness in policy..
- Example: wind influence on a helicopter is stochastic, but if we assume the same wind pattern across trials, this will make the different choices of θ more readily comparable
- *Note: equally applicable to evolutionary methods*

[Ng & Jordan, 2000] provide theoretical analysis of gains from fixing randomness ("pegasus")

Learning to Hover

x, y, z: x points forward along the helicopter, y sideways to the right, z downward.

n_x, n_y, n_z: rotation vector that brings helicopter back to "level" position (expressed in the helicopter frame).

$$u_{collective} = \theta_1 \cdot f_1(z^* - z) + \theta_2 \cdot \dot{z}$$

$$u_{elevator} = \theta_3 \cdot f_2(x^* - x) + \theta_4 f_4(\dot{x}) + \theta_5 \cdot q + \theta_6 \cdot n_y$$

$$u_{aileron} = \theta_7 \cdot f_3(y^* - y) + \theta_8 f_5(\dot{y}) + \theta_9 \cdot p + \theta_{10} \cdot n_x$$

$$u_{rudder} = \theta_{11} \cdot r + \theta_{12} \cdot n_z$$