Policy Search

CMPT 729 G100

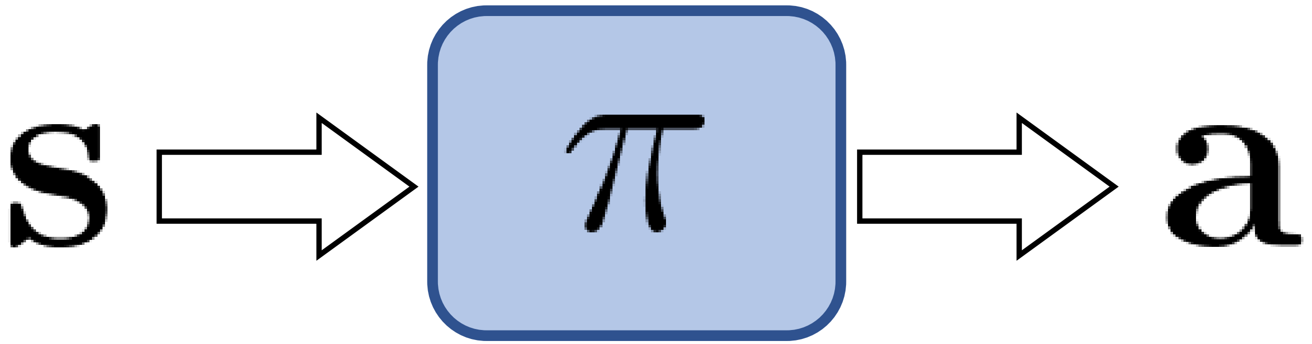
Jason Peng

Overview

- Policy Optimization
- Black Box Optimization
- Evolutionary Methods
- Finite-Difference Methods

Policy

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



Supervised Learning

$$\{(\mathbf{o}_0, \underline{\mathbf{a}_0}), (\mathbf{o}_1, \underline{\mathbf{a}_1}), \ldots\}$$

$$\min_{\pi} \mathbb{E}_{(\mathbf{o},\mathbf{a})\sim\mathcal{D}}\left[-\log \pi(\mathbf{a}|\mathbf{o})\right]$$

Dataset

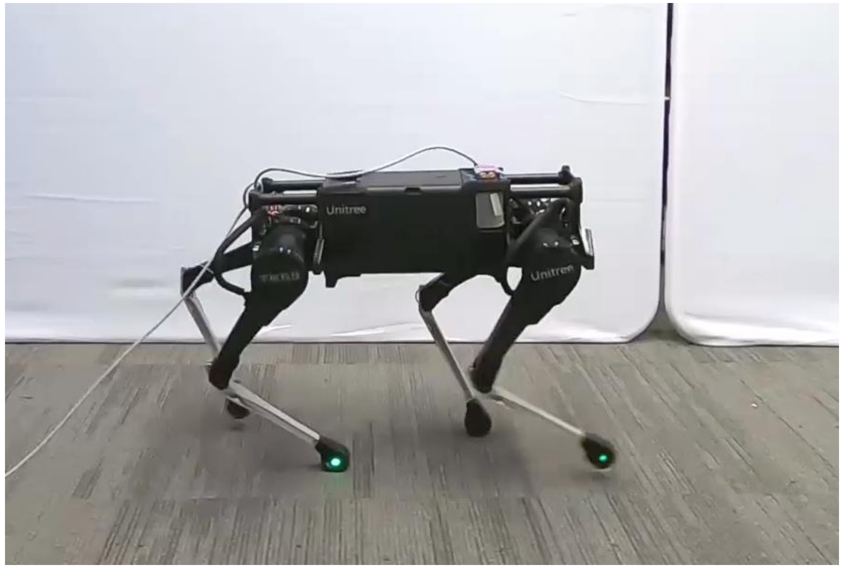
Behavioral Cloning

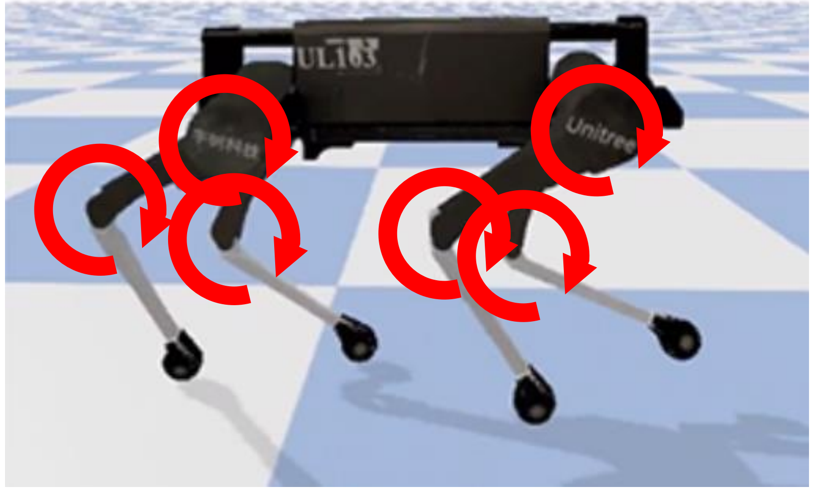
Supervised Learning

$$\{(\underline{\mathbf{x}_i}, \underline{y_i})\}$$

Robot State

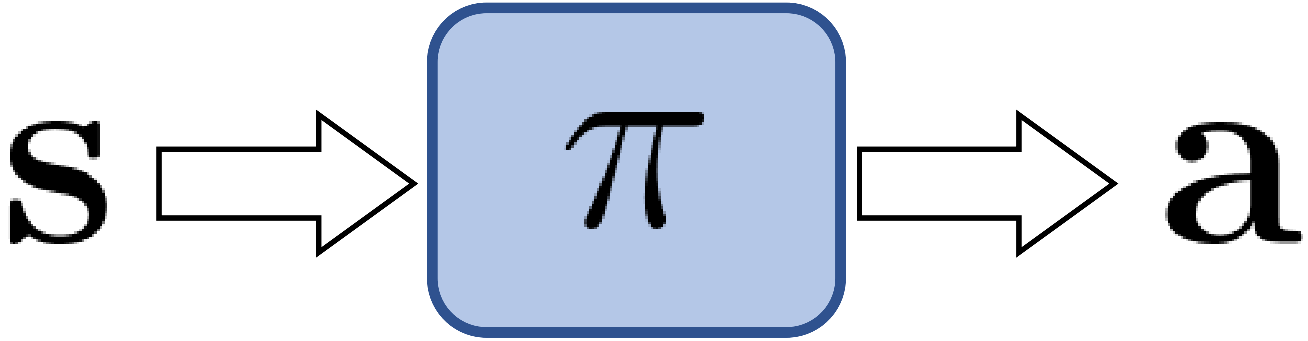
Motor Commands ?





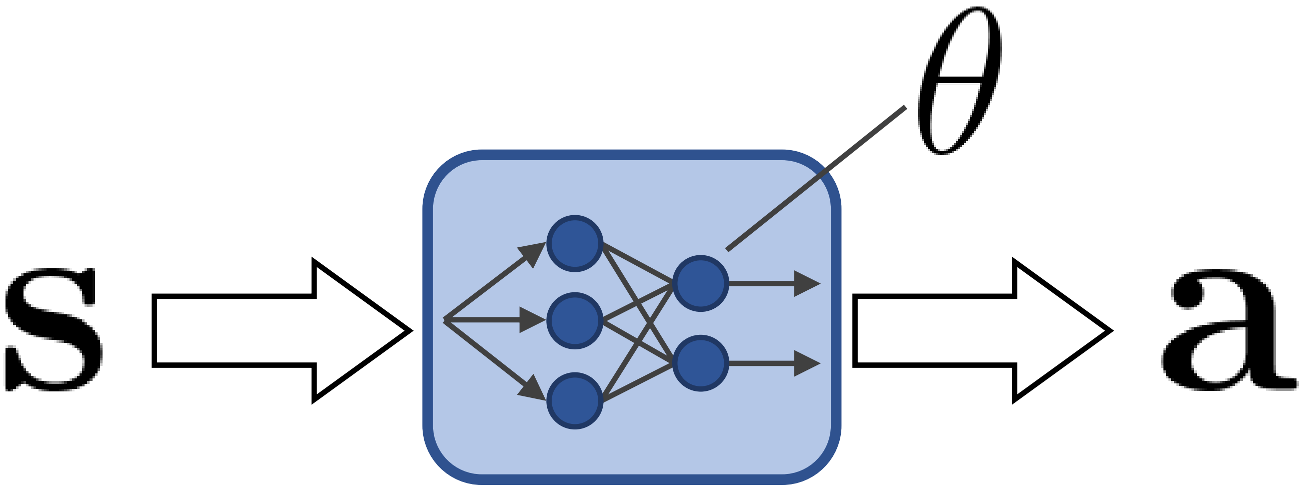
Policy

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



Policy

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



Optimization

$$\theta^* = \arg\max_{\theta} \; J(\pi_\theta)$$

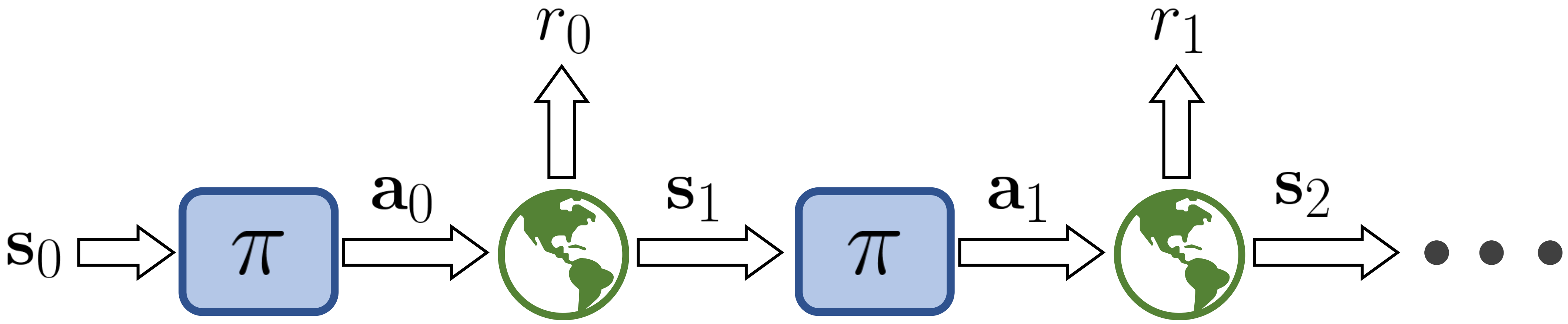
$$\Downarrow$$

Just use gradient ascent!

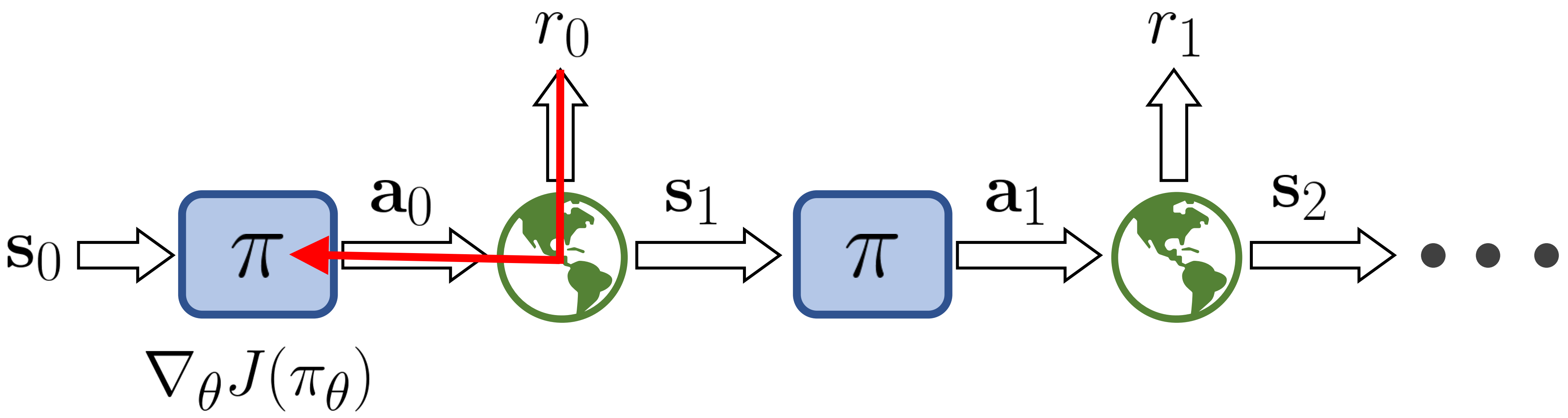
Objective is often **<u>NOT</u>** differentiable

$$\cancel{\nabla_\theta J(\pi_\theta)}$$

Optimization

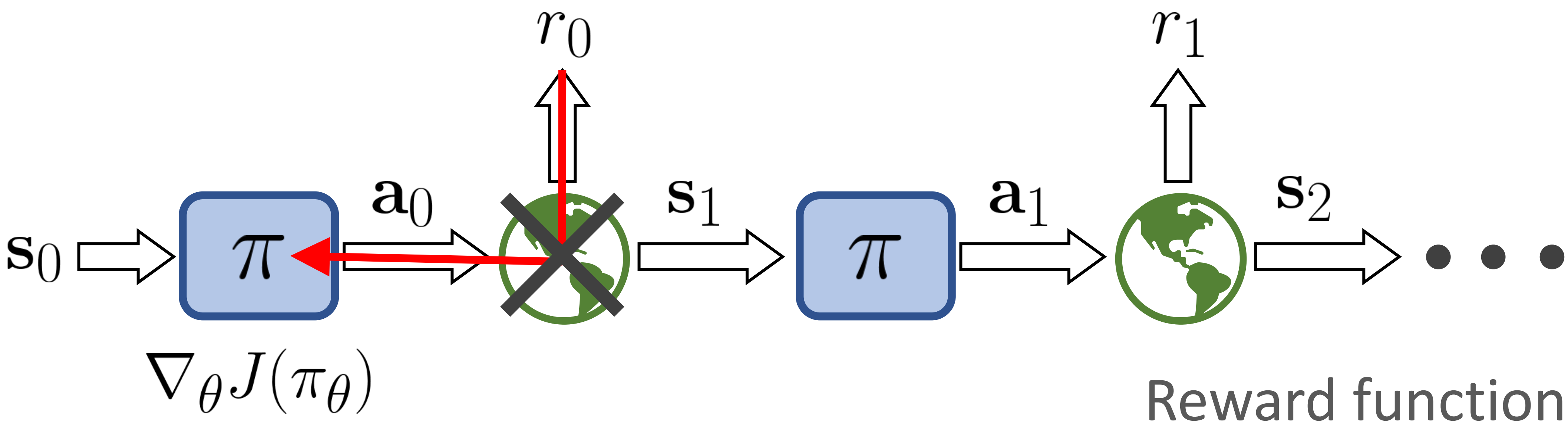


Optimization



$$\frac{\partial r_0}{\partial \theta}$$

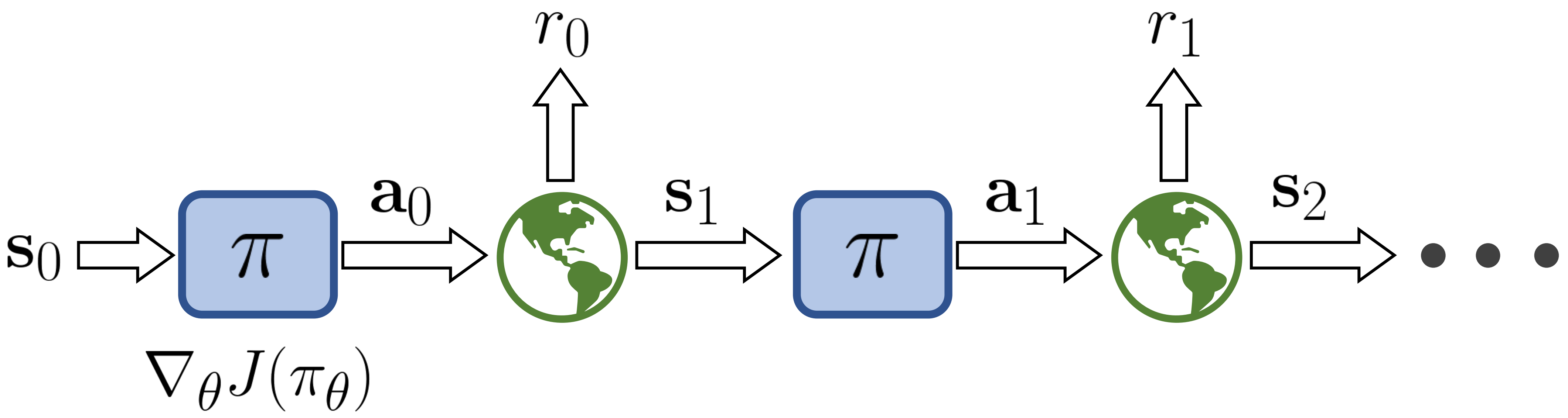
Optimization



Reward function may not be differentiable

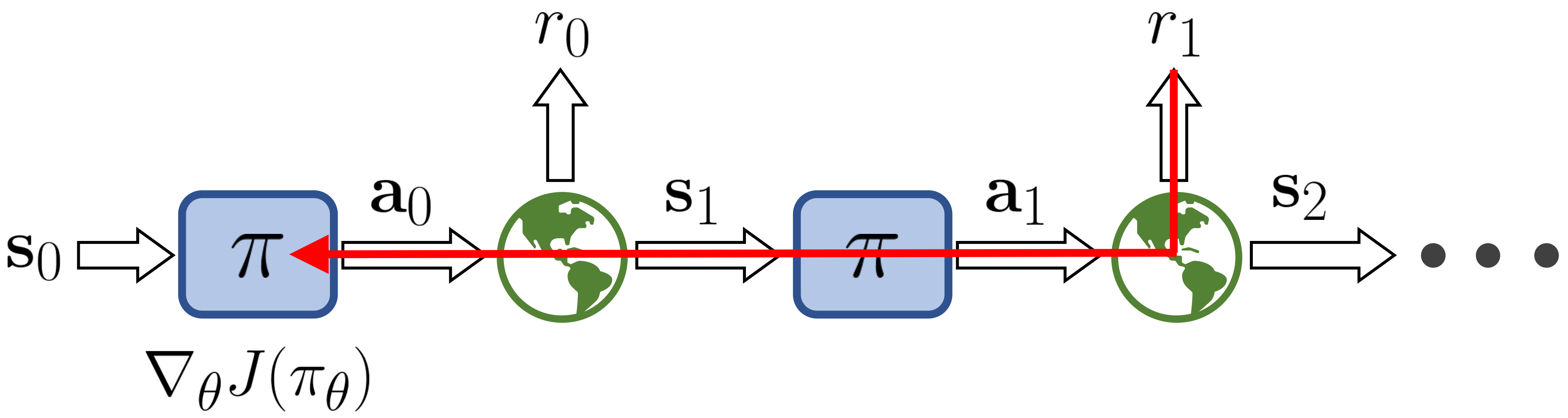
$$\frac{\partial r_0}{\partial \theta} = \frac{\partial r_0}{\partial \mathbf{a}_0} \frac{\partial \mathbf{a}_0}{\partial \theta}$$

Optimization



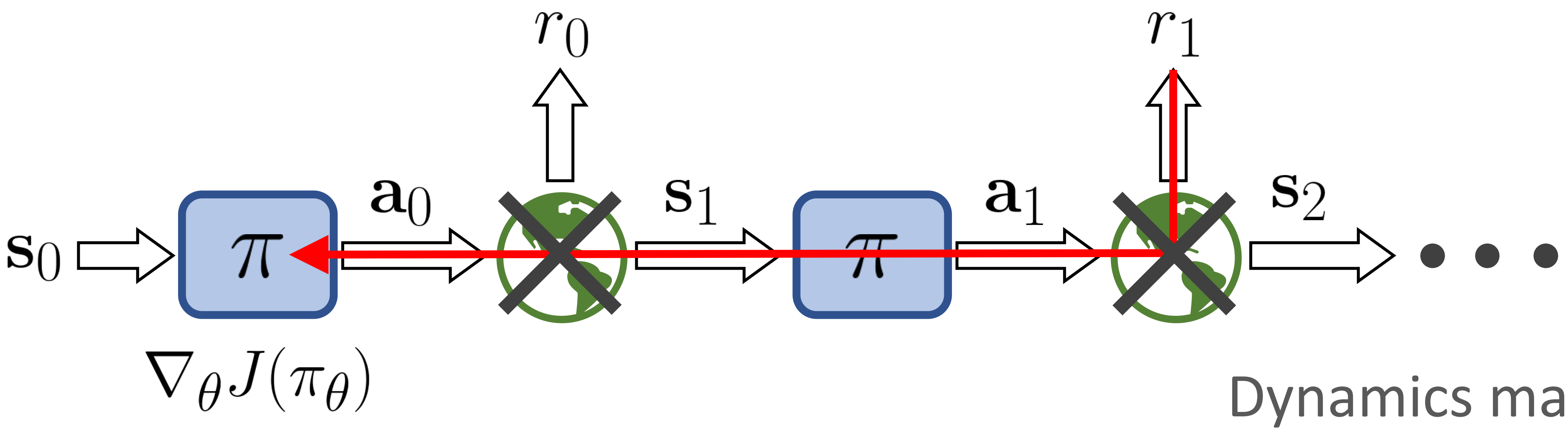
$$\frac{\partial r_1}{\partial \theta} = \frac{\partial r_1}{\partial \mathbf{a}_1}\frac{\partial \mathbf{a}_1}{\partial \theta}$$

Optimization



$$\frac{\partial r_1}{\partial \theta} = \frac{\partial r_1}{\partial \mathbf{a}_1}\frac{\partial \mathbf{a}_1}{\partial \theta}$$

Optimization



Dynamics may not be differentiable

$$\frac{\partial r_1}{\partial \theta} = \frac{\partial r_1}{\partial \mathbf{a}_1}\frac{\partial \mathbf{a}_1}{\partial \theta} + \frac{\partial r_1}{\partial \mathbf{a}_1}\frac{\partial \mathbf{a}_1}{\partial \mathbf{s}_1}\boxed{\frac{\partial \mathbf{s}_1}{\partial \mathbf{a}_0}}\frac{\partial \mathbf{a}_0}{\partial \theta} + \dots$$

Dynamics

Nondifferentiable Objective

$$\theta^* = \arg\max_{\theta} \; J(\pi_\theta)$$

Just use gradient ascent!

Objective is often NOT differentiable

$$\cancel{\nabla_\theta J(\pi_\theta)}$$

Black Box Optimization

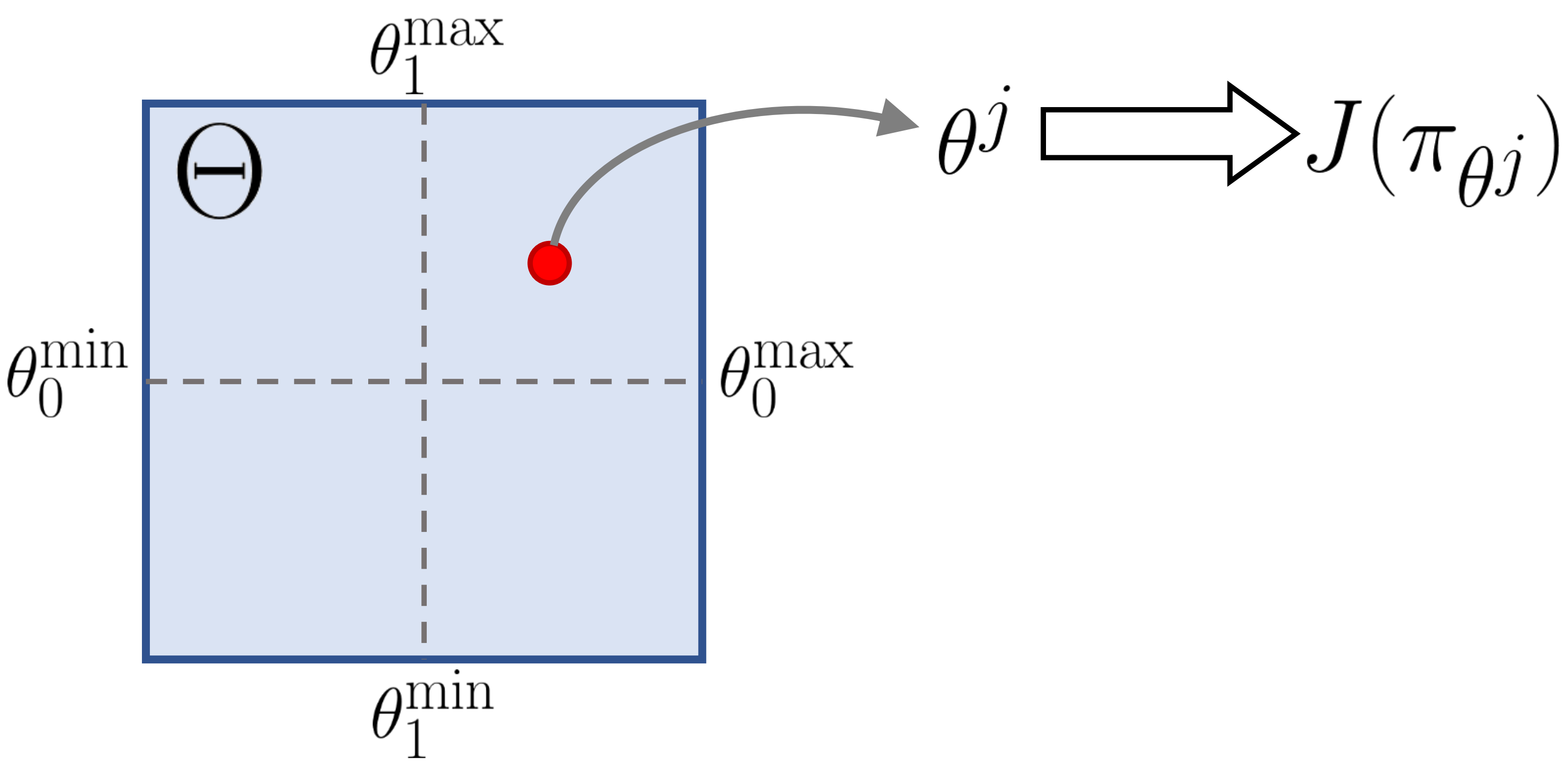
$$\theta^* = \arg\max_{\theta} \; \underline{J(\pi_\theta)}$$

black box

$$\theta \Rightarrow \boxed{J} \Rightarrow J(\pi_\theta)$$

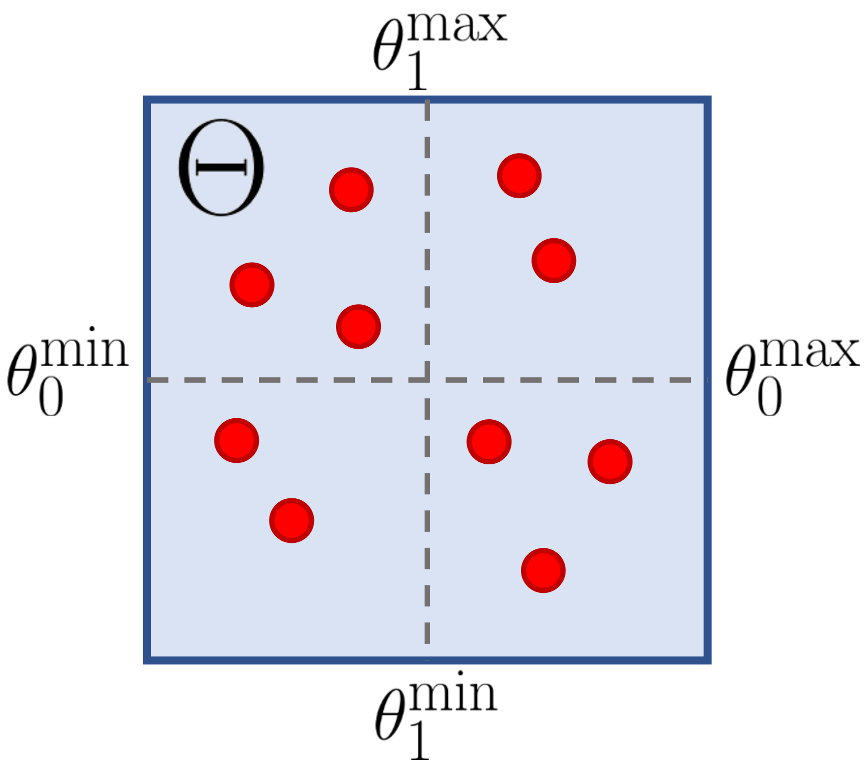
Random Search

- Parameter space: $\theta \in \Theta$
- Each parameter is bounded: $\theta_i \in [\theta_i^{\min}, \theta_i^{\max}]$
- Idea: just sample randomly



Random Search

- Parameter space: $\theta \in \Theta$
- Each parameter is bounded: $\theta_i \in [\theta_i^{\min}, \theta_i^{\max}]$
- Idea: just sample randomly

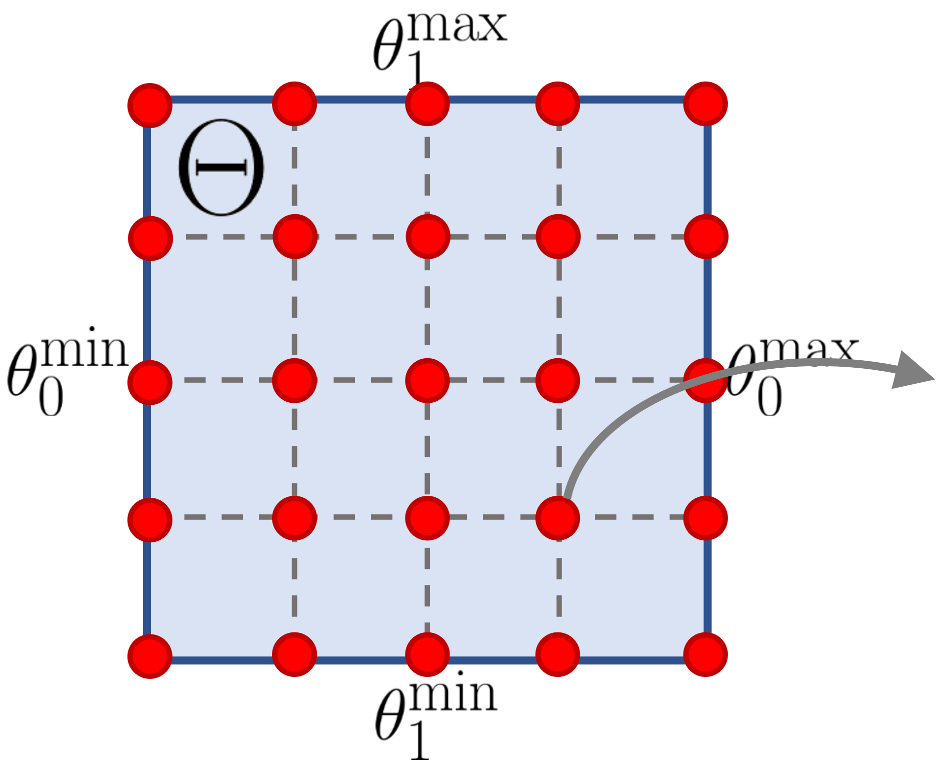


Pick θ with highest expected return

Random Search for Hyper-Parameter Optimization
[Bergstra and Bengio 2012]

Grid Search

- Parameter space: $\theta \in \Theta$
- Each parameter is bounded: $\theta_i \in [\theta_i^{\min}, \theta_i^{\max}]$
- Idea: discretize space and evaluate



Pick θ with highest expected return

$$\theta^j \Longrightarrow J(\pi_{\theta^j})$$

Random Search for Hyper-Parameter Optimization
[Bergstra and Bengio 2012]

Random Search

✓ Extremely simple

✓ Trivially parallel

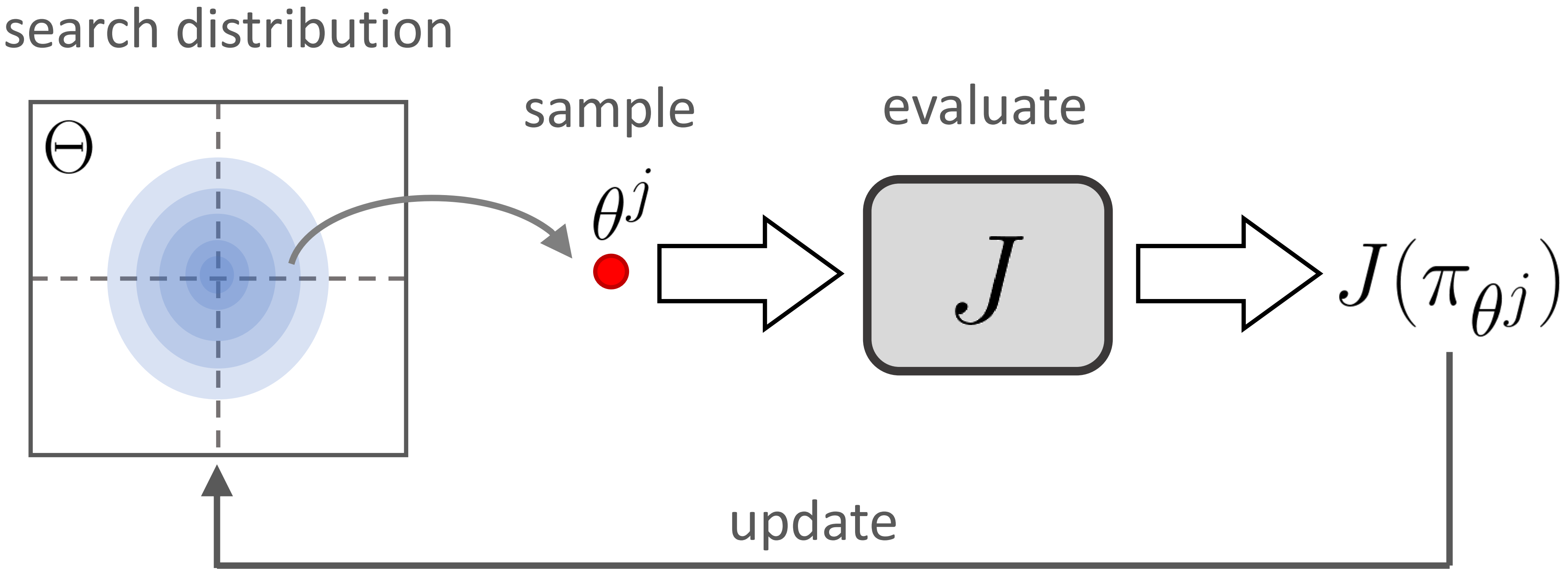
✓ Can work well for small number of parameters (< 10)

✗ Curse of dimensionality

✗ Probably won't find an optimum

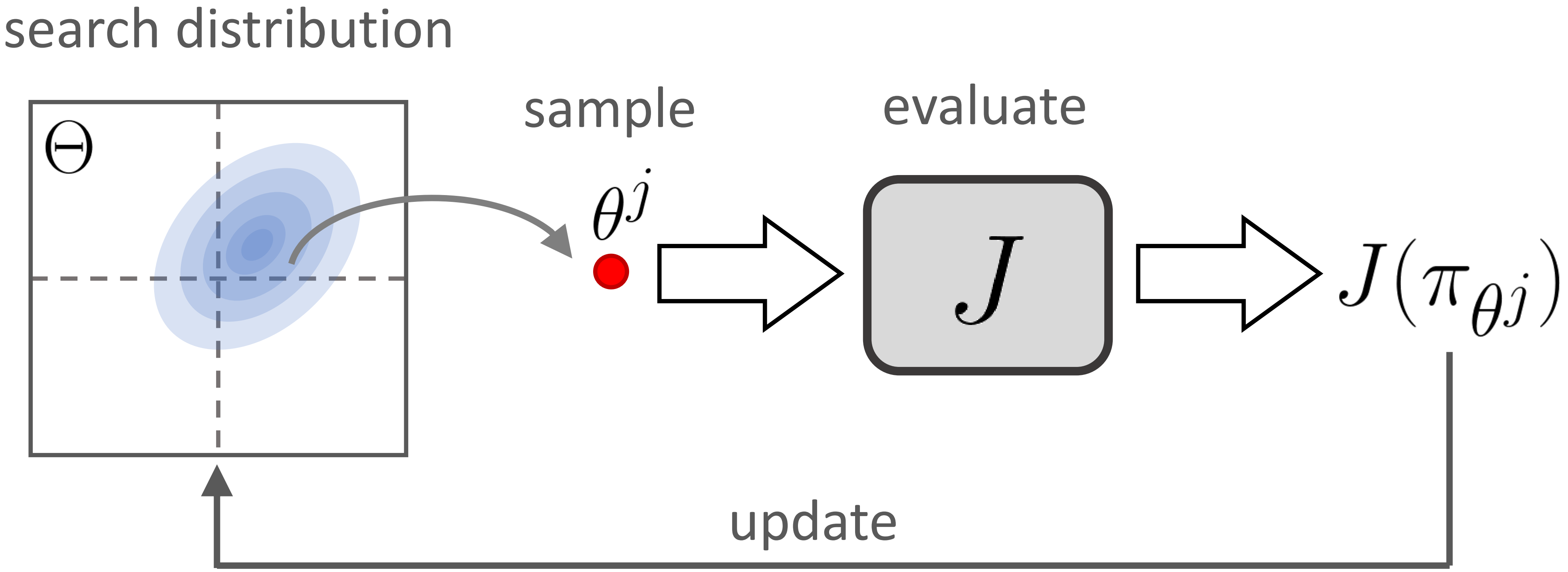
Smarter Search

- Adapt search samples base on objective



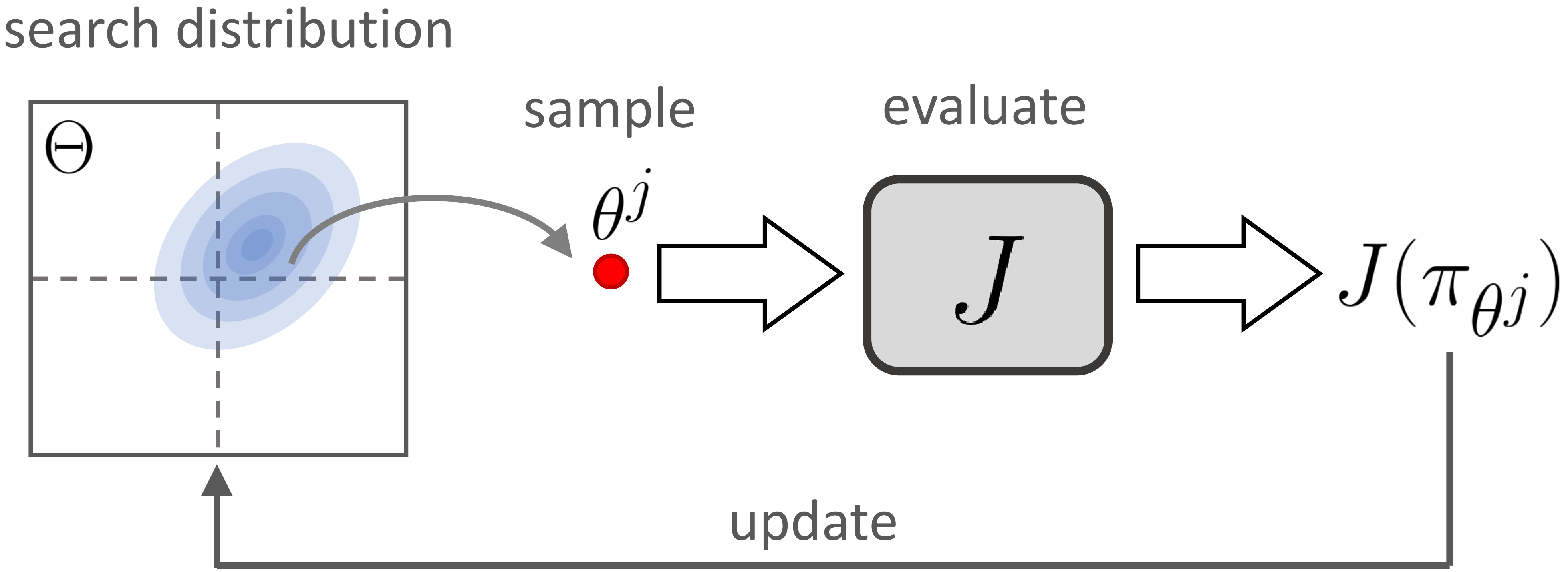
Smarter Search

- Adapt search samples base on objective



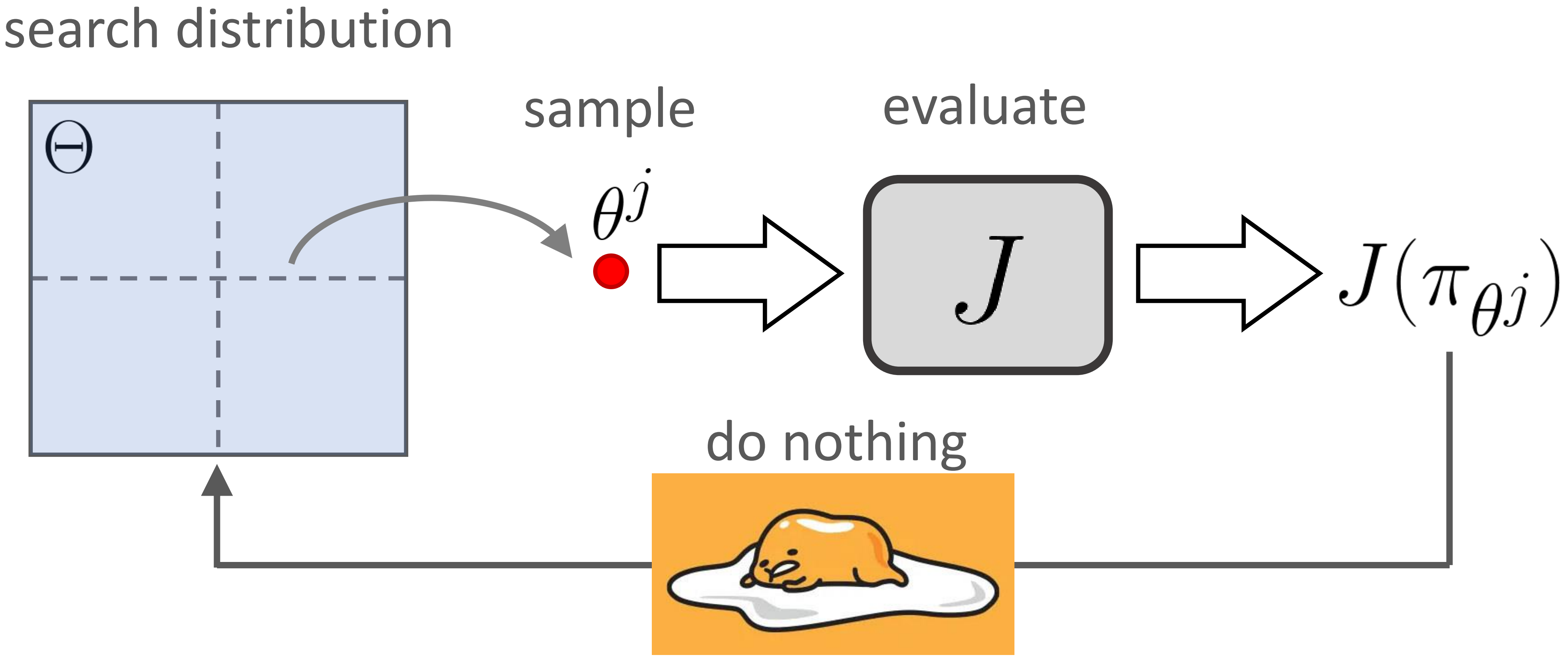
Random Search

- Random Search



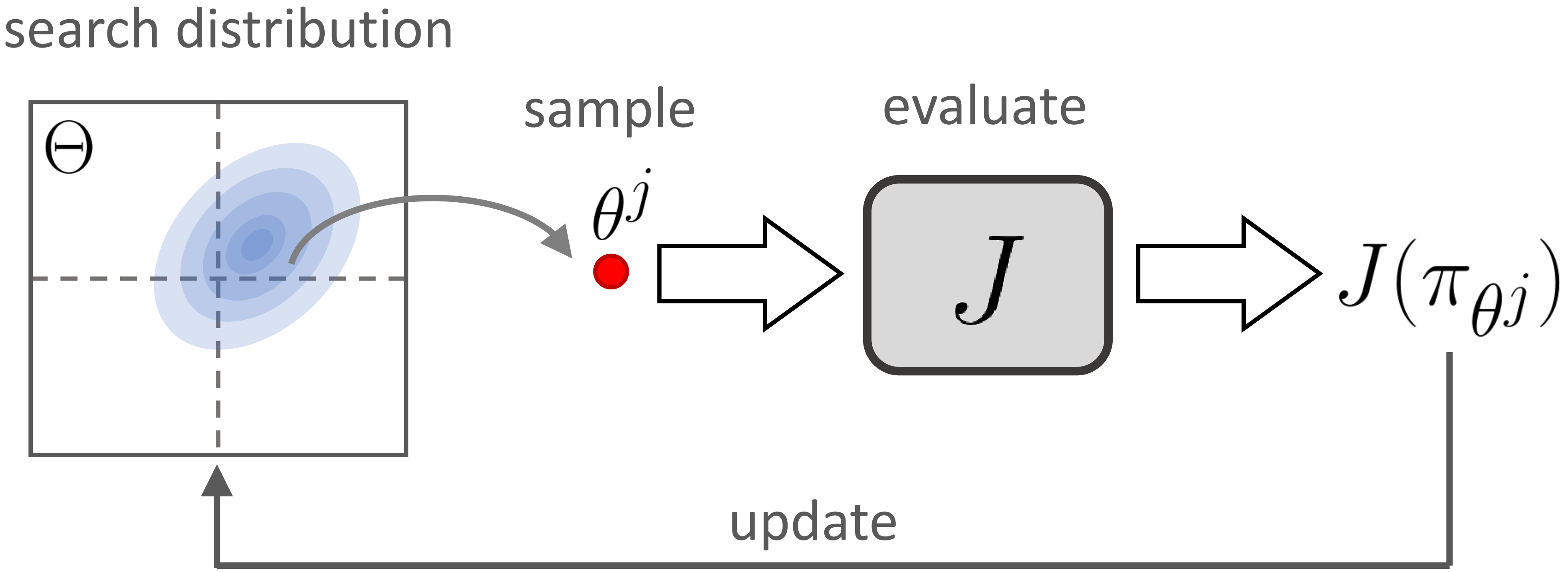
Random Search

- Random Search

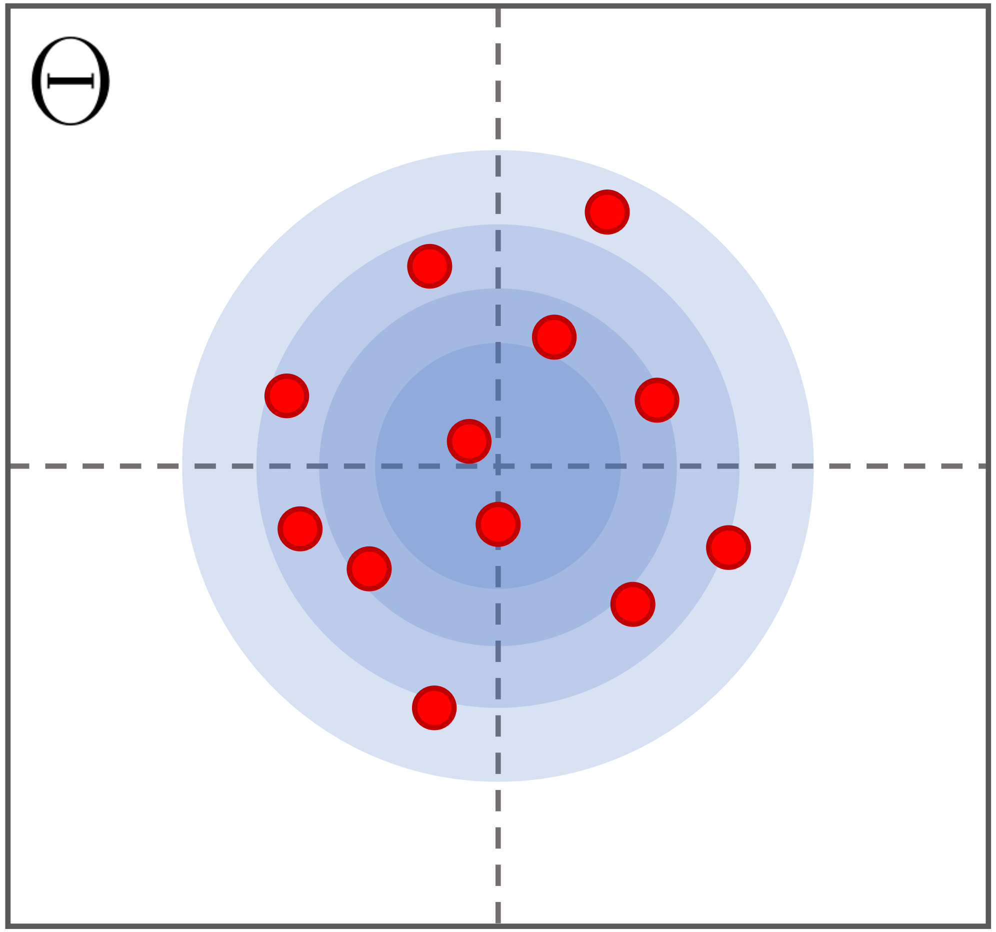


Evolutionary Methods

- Adapt search samples base on objective

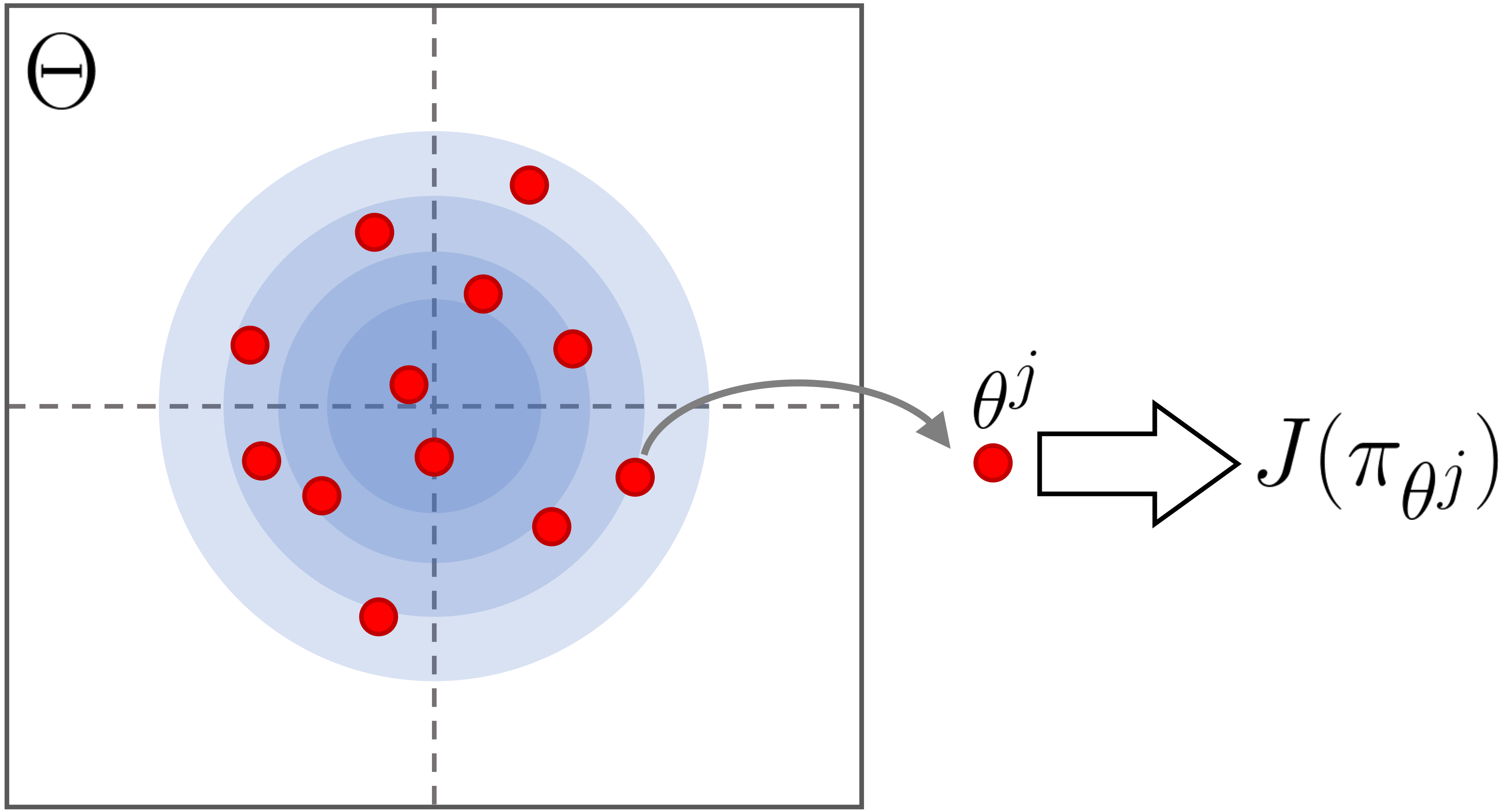


Cross-Entropy Method



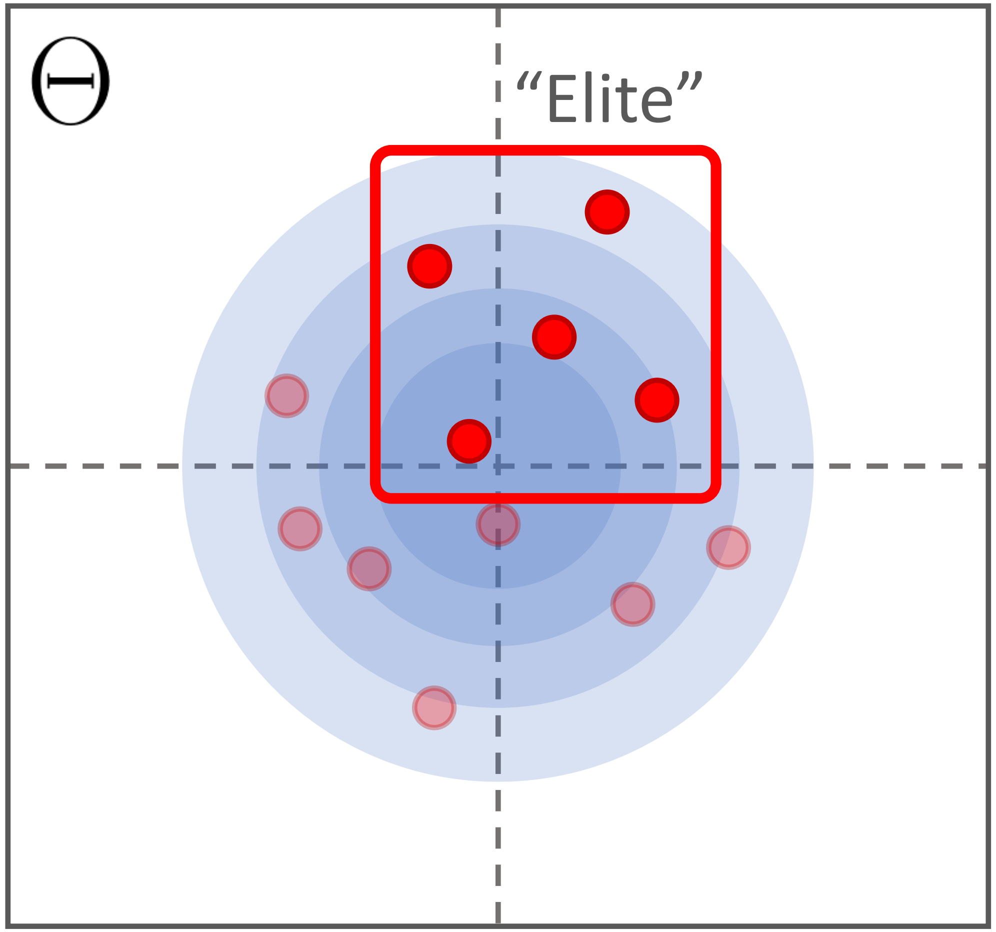
The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method



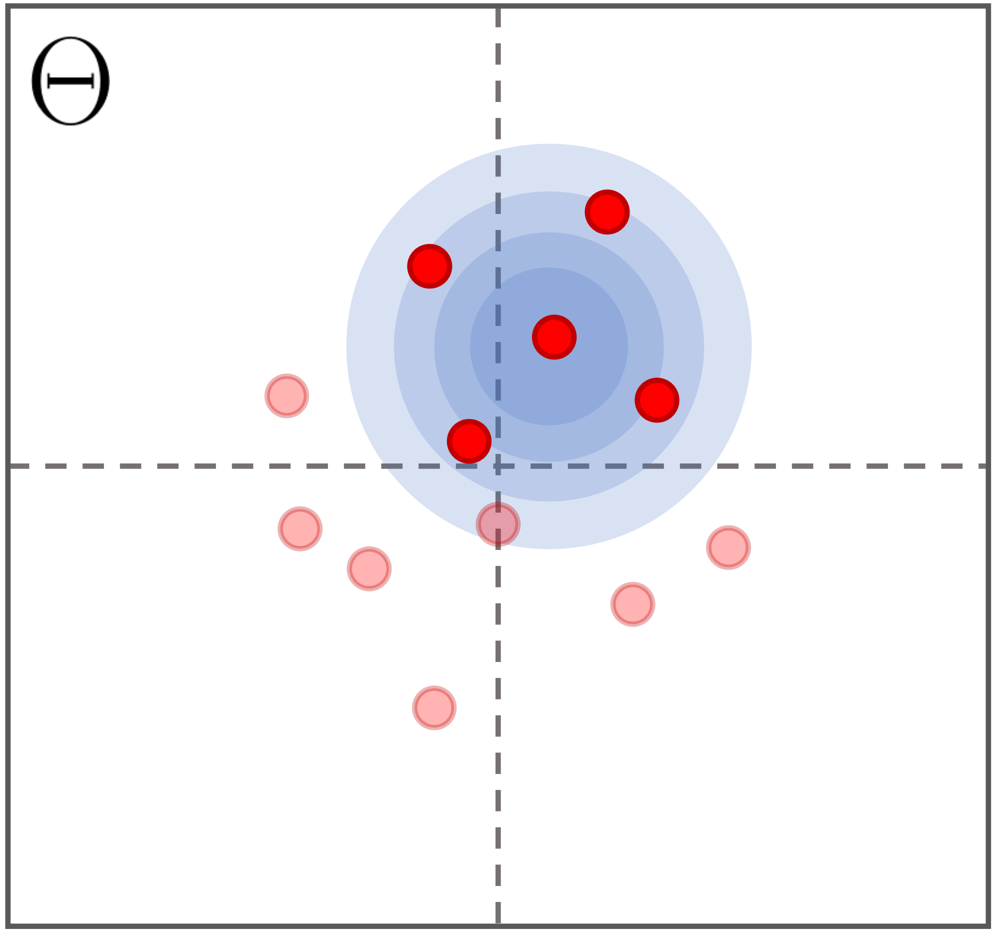
The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method



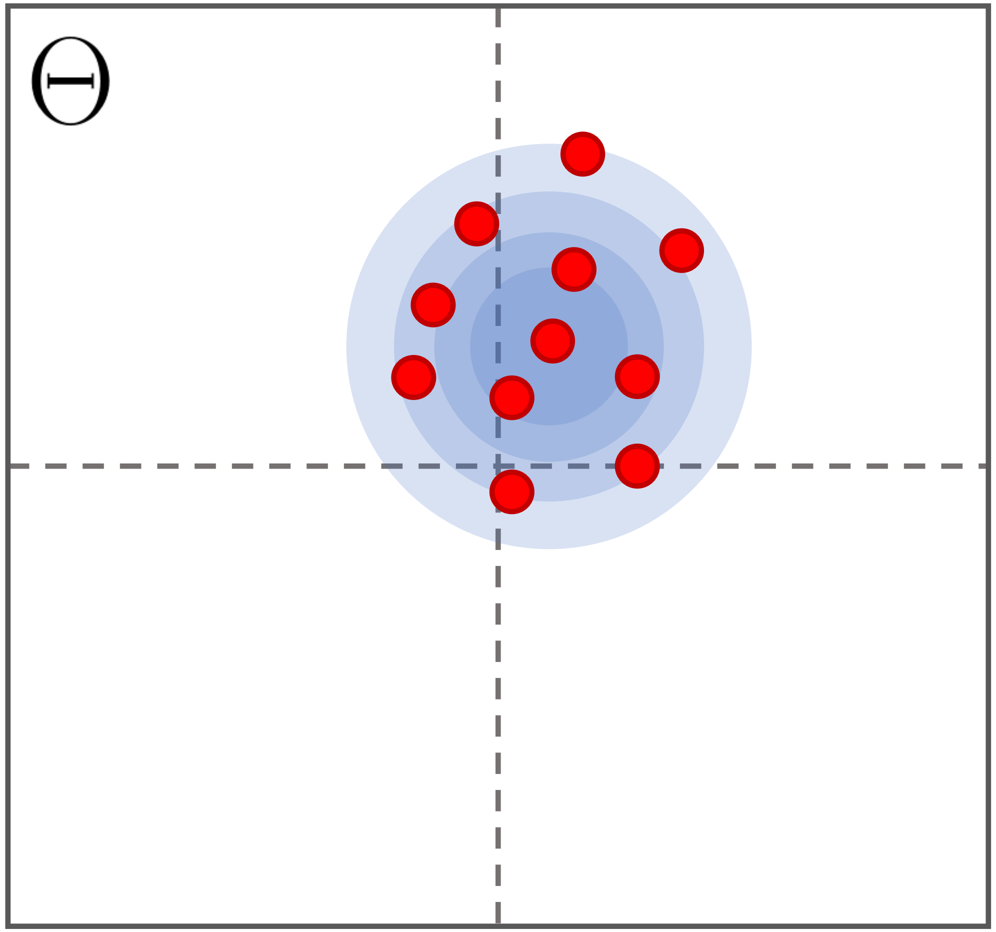
The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method



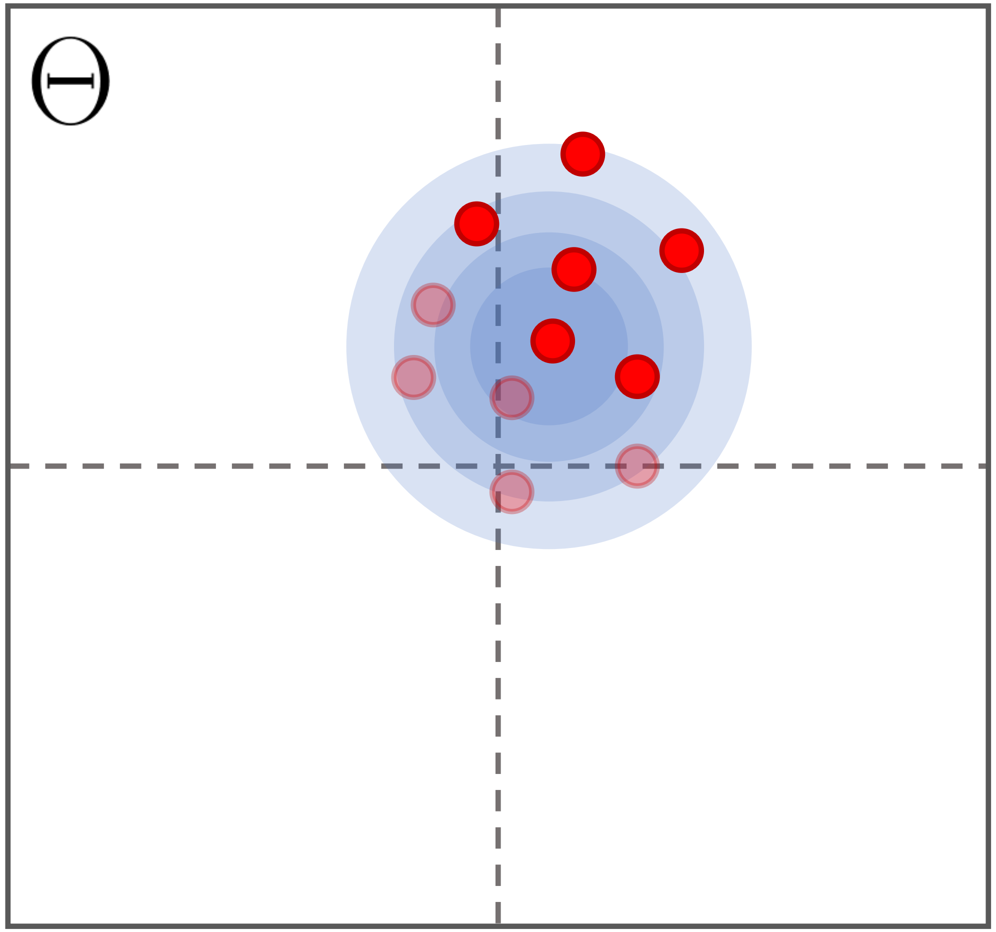
The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method



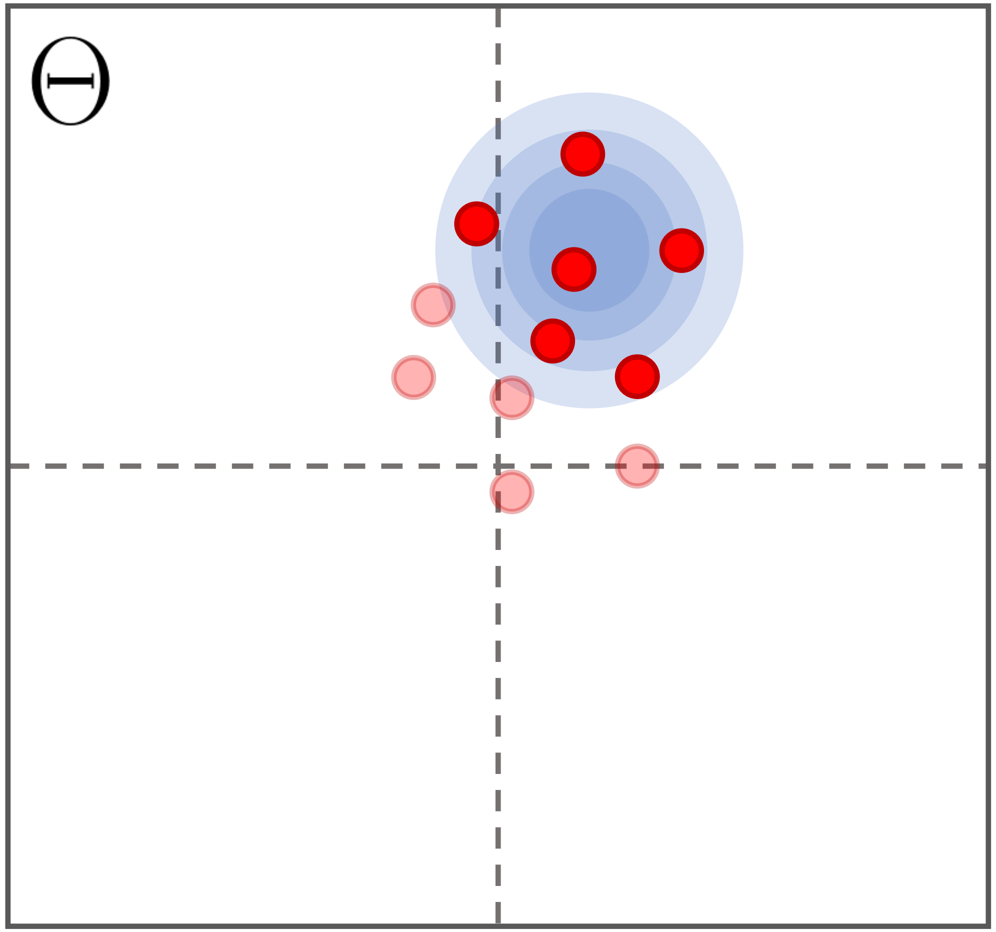
The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method



The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method



The Cross-Entropy Method for Combinatorial and Continuous Optimization [Rubinstein 1999]
The Cross-Entropy Method for Optimization [Botev et al. 2013]

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, \ldots, k-1$ **do**
3: Sample parameters $\theta_1, \ldots, \theta_n \sim q^i(\theta)$
4: Evaluate performance of samples $J(\theta_1), \ldots, J(\theta_n)$
5: Select elite samples with highest performance $\hat{\theta}_1, \ldots, \hat{\theta}_m$

6: Update search distribution with elite samples:
$q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)$
7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, ..., k-1$ **do**

3: Sample parameters $\theta_1, ..., \theta_n \sim q^i(\theta)$

4: Evaluate performance of samples $J(\theta_1), ..., J(\theta_n)$

5: Select elite samples with highest performance $\hat{\theta}_1, ..., \hat{\theta}_m$

6: Update search distribution with elite samples:

$q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)$

7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, ..., k-1$ **do**

3: Sample parameters $\theta_1, ..., \theta_n \sim q^i(\theta)$

4: Evaluate performance of samples $J(\theta_1), ..., J(\theta_n)$

5: Select elite samples with highest performance $\hat{\theta}_1, ..., \hat{\theta}_m$

6: Update search distribution with elite samples:

$q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)$

7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, ..., k-1$ **do**

3: Sample parameters $\theta_1, ..., \theta_n \sim q^i(\theta)$

4: Evaluate performance of samples $J(\theta_1), ..., J(\theta_n)$

5: Select elite samples with highest performance $\hat{\theta}_1, ..., \hat{\theta}_m$

6: Update search distribution with elite samples:
$q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)$

7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, ..., k-1$ **do**
3: Sample parameters $\theta_1, ..., \theta_n \sim q^i(\theta)$
4: Evaluate performance of samples $J(\theta_1), ..., J(\theta_n)$
5: Select elite samples with highest performance $\hat{\theta}_1, ..., \hat{\theta}_m$

6: Update search distribution with elite samples:
$q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)$
7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, \ldots, k-1$ **do**

3: Sample parameters $\theta_1, \ldots, \theta_n \sim q^i(\theta)$

4: Evaluate performance of samples $J(\theta_1), \ldots, J(\theta_n)$

5: Select elite samples with highest performance $\hat{\theta}_1, \ldots, \hat{\theta}_m$

6: Update search distribution with elite samples:

$$\underline{q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)}$$

7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, ..., k-1$ **do**

3: Sample parameters $\theta_1, ..., \theta_n \sim q^i(\theta)$

4: Evaluate performance of samples $J(\theta_1), ..., J(\theta_n)$

5: Select elite samples with highest performance $\hat{\theta}_1, ..., \hat{\theta}_m$

6: Update search distribution with elite samples:
$q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)$

7: **end for**

8: return best performing θ

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, \ldots, k-1$ **do**
3: Sample parameters $\theta_1, \ldots, \theta_n \sim q^i(\theta)$
4: Evaluate performance of samples $J(\theta_1), \ldots, J(\theta_n)$
5: Select elite samples with highest performance $\hat{\theta}_1, \ldots, \hat{\theta}_m$

6: Update search distribution with elite samples:
$$\boxed{q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)}$$
7: **end for**

8: return best performing θ

$$q^i = \mathcal{N}\left(\mu^i, \Sigma^i\right)$$

$$\mu^i = \begin{pmatrix} \mu_1^i \\ \vdots \\ \mu_d^i \end{pmatrix}$$

$$\Sigma^i = \begin{bmatrix} \sigma_1^i & & \\ & \ddots & \\ & & \sigma_d^i \end{bmatrix}$$

Gaussian Maximum Likelihood

$$\arg\max_{q} \frac{1}{m}\sum_{j=1}^{m} \log q(\hat{\theta}_j)$$ where $q = \mathcal{N}(\mu, \Sigma)$

$$\nabla_q \frac{1}{m}\sum_{j=1}^{m} \log q(\hat{\theta}_j) = 0$$

$$\nabla_q \frac{1}{m}\sum_{j=1}^{m} -\frac{1}{2}\left(\hat{\theta}_j - \mu\right)^T \Sigma^{-1} \left(\hat{\theta}_j - \mu\right) - \frac{1}{2}\log \det(\Sigma) + C \;= 0$$

Gaussian Maximum Likelihood

$$\nabla_\mu \frac{1}{m} \sum_{j=1}^{m} -\frac{1}{2} \left(\hat{\theta}_j - \mu\right)^T \Sigma^{-1} \left(\hat{\theta}_j - \mu\right) - \frac{1}{2} \log \det \left(\Sigma\right) + C \; = 0$$

$$= \frac{1}{m} \sum_{j=1}^{m} \Sigma^{-1} \left(\hat{\theta}_j - \mu\right) = 0$$

$$= \frac{1}{m} \sum_{j=1}^{m} \underline{\Sigma} \Sigma^{-1} \left(\hat{\theta}_j - \mu\right) = \underline{\Sigma} 0$$

Gaussian Maximum Likelihood

$$\nabla_\mu \frac{1}{m}\sum_{j=1}^{m} -\frac{1}{2}\left(\hat{\theta}_j - \mu\right)^T \Sigma^{-1}\left(\hat{\theta}_j - \mu\right) - \frac{1}{2}\log\det\left(\Sigma\right) + C \;= 0$$

$$= \frac{1}{m}\sum_{j=1}^{m} \Sigma^{-1}\left(\hat{\theta}_j - \mu\right) = 0$$

$$= \frac{1}{m}\sum_{j=1}^{m} \Sigma\Sigma^{-1}\left(\hat{\theta}_j - \mu\right) = \Sigma 0$$

$$= \frac{1}{m}\sum_{j=1}^{m} \left(\hat{\theta}_j - \mu\right) = 0$$

$$= \left(\frac{1}{m}\sum_{j=1}^{m}\left(\hat{\theta}_j\right)\right) - \mu = 0 \quad \Rightarrow \quad \boxed{\mu^* = \frac{1}{m}\sum_{j=1}^{m}\hat{\theta}_j}$$

Gaussian Maximum Likelihood

$$\nabla_{\Sigma} \frac{1}{m} \sum_{j=1}^{m} -\frac{1}{2} \left(\hat{\theta}_j - \mu\right)^T \Sigma^{-1} \left(\hat{\theta}_j - \mu\right) - \frac{1}{2} \underline{\log \det (\Sigma)} + C = 0$$

$$= \sum_{i=1}^{d} \log \sigma_i$$

$$\Sigma = \begin{bmatrix} \sigma_i & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix}$$

$$\Downarrow$$

$$\nabla_{\sigma_i} \frac{1}{m} \sum_{j=1}^{m} -\frac{1}{2\sigma_i} \left(\hat{\theta}_{j,i} - \underline{\mu_i^*}\right)^2 - \frac{1}{2} \log \sigma_i = 0$$

Gaussian Maximum Likelihood

$$\nabla_{\sigma_i} \frac{1}{m} \sum_{j=1}^{m} -\frac{1}{2\sigma_i} \left(\hat{\theta}_{j,i} - \mu_i^* \right)^2 - \frac{1}{2} \log \sigma_i = 0$$

$$= \frac{1}{2\sigma_i^2} \frac{1}{m} \sum_{j=1}^{m} \left(\hat{\theta}_{j,i} - \mu_i^* \right)^2 - \frac{1}{2\sigma_i} = 0$$

$$\frac{1}{2\sigma_i^2} \frac{1}{m} \sum_{j=1}^{m} \left(\hat{\theta}_{j,i} - \mu_i^* \right)^2 = \frac{1}{2\sigma_i}$$

$$\boxed{\sigma_i^* = \frac{1}{m} \sum_{j=1}^{m} \left(\hat{\theta}_{j,i} - \mu_i^* \right)^2}$$

Cross-Entropy Method

ALGORITHM 4: CEM

1: $q^0 \leftarrow$ initialize search distribution

2: **for** iteration $i = 0, \ldots, k-1$ **do**
3: Sample parameters $\theta_1, \ldots, \theta_n \sim q^i(\theta)$
4: Evaluate performance of samples $J(\theta_1), \ldots, J(\theta_n)$
5: Select elite samples with highest performance $\hat{\theta}_1, \ldots, \hat{\theta}_m$

6: Update search distribution with elite samples:
$\boxed{q^{i+1} = \arg\max_q \frac{1}{m} \sum_{j=1}^{m} \log q(\hat{\theta}_j)}$
7: **end for**

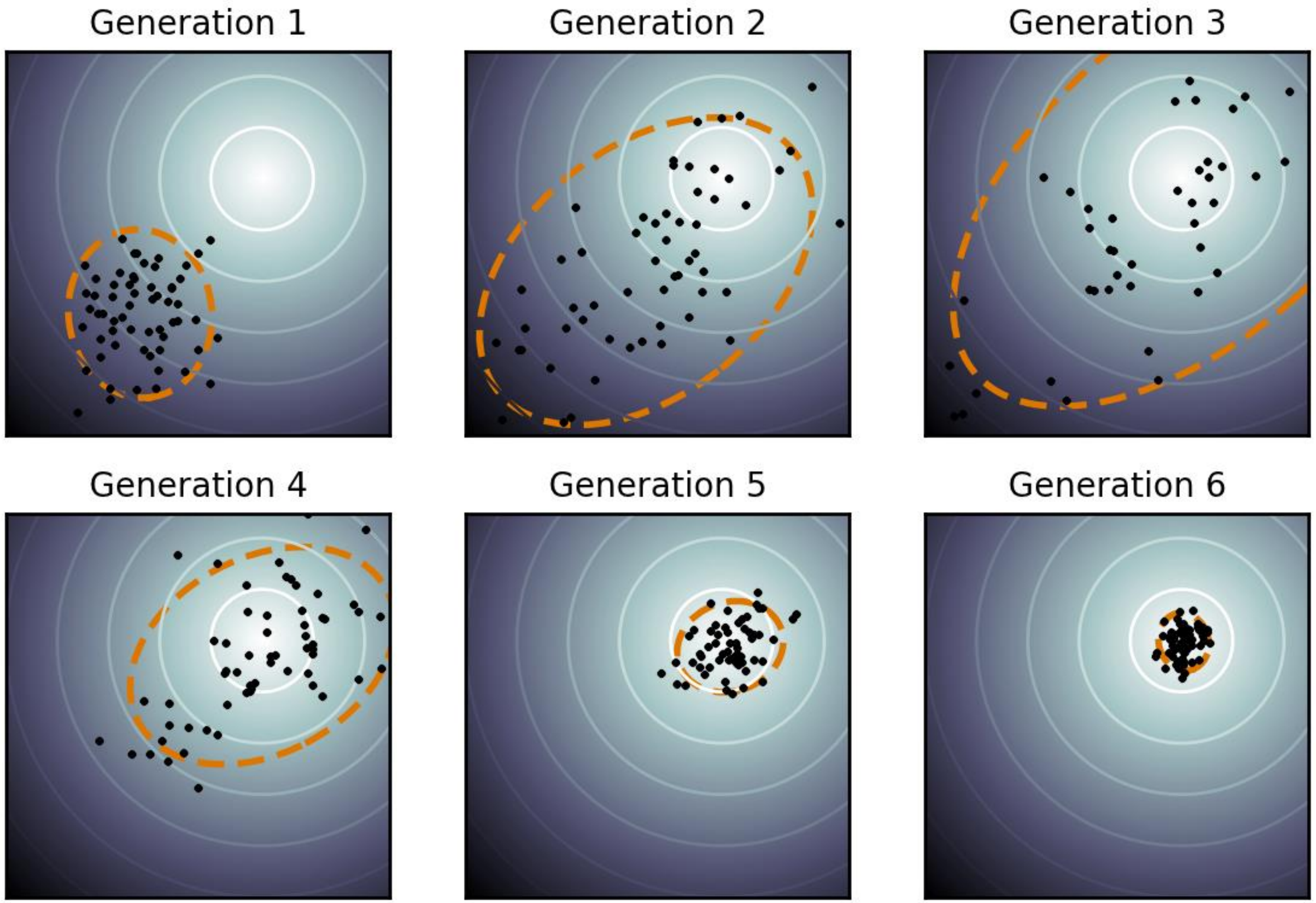
8: return best performing θ

$$q^i = \mathcal{N}\left(\mu^i, \Sigma^i\right)$$

$$\mu^* = \frac{1}{m} \sum_{j=1}^{m} \hat{\theta}_j$$

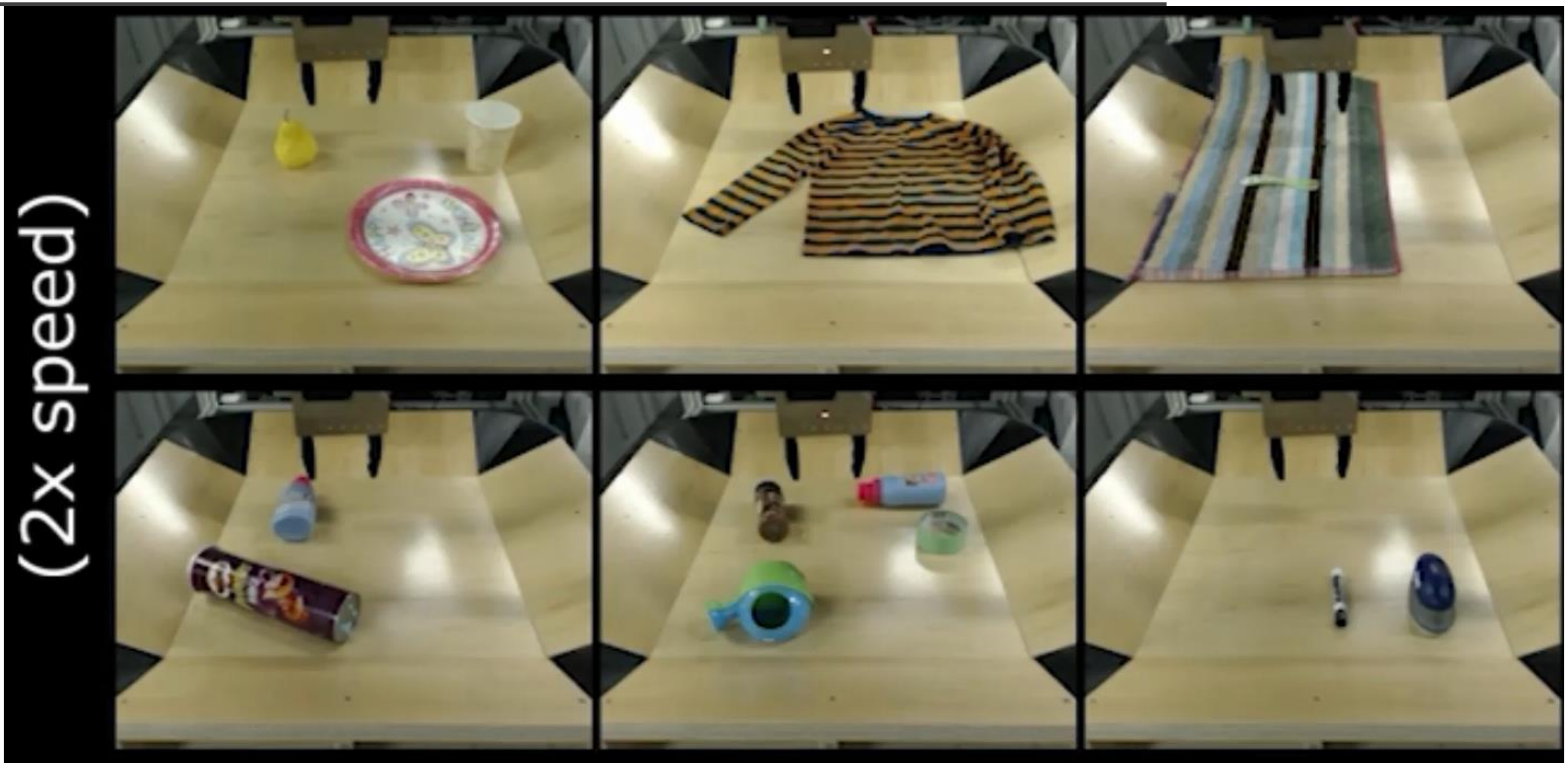
$$\sigma_i^* = \frac{1}{m} \sum_{j=1}^{m} \left(\hat{\theta}_{j,i} - \mu_i^*\right)^2$$

Covariance Matrix Adaptation (CMA)



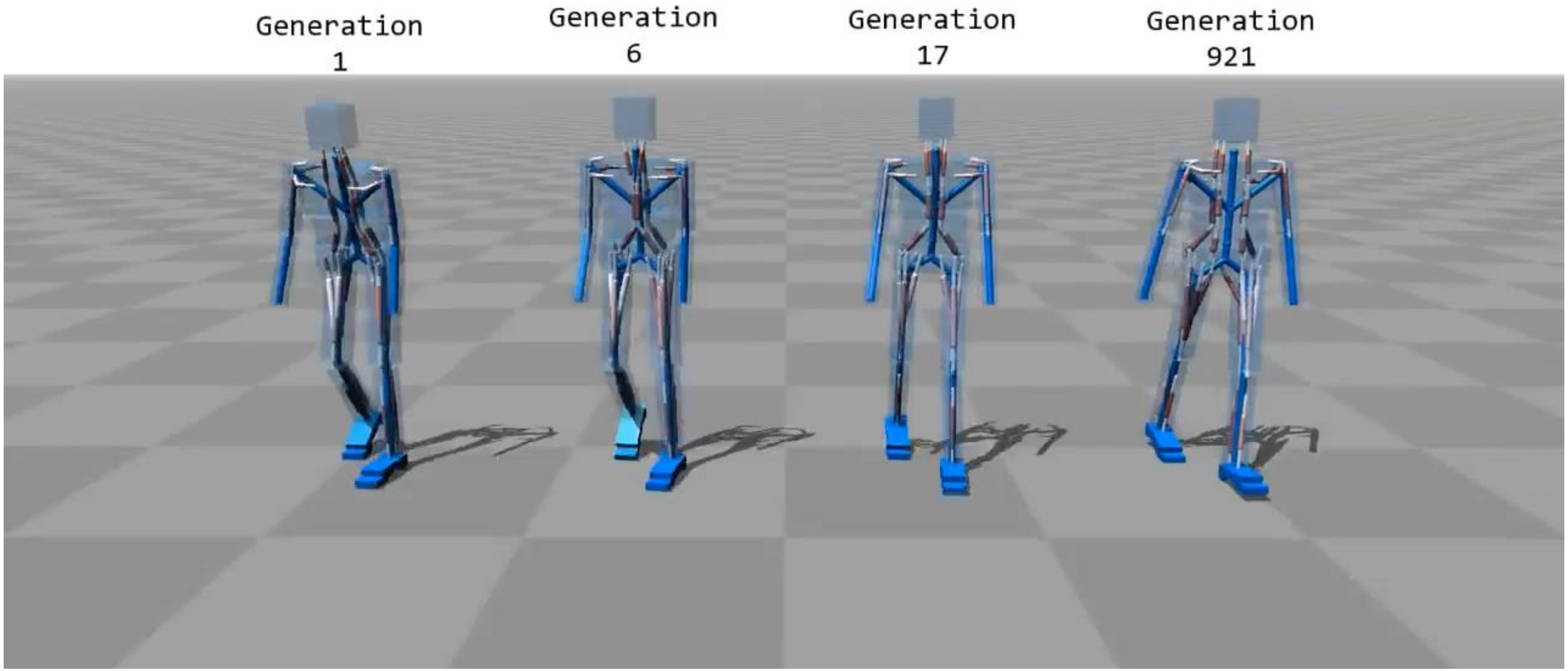
The CMA Evolution Strategy: A Comparing Review
[Hansen 2006]

Evolution Strategy Applications



Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control
[Ebert et al. 2015]

Evolution Strategy Applications



Flexible Muscle-Based Locomotion for Bipedal Creatures
[Geijtenbeek et al. 2013]

Evolution Strategies

✓ Highly parallelizable

✓ Can work well for < 100 parameters

✗ Slow convergence

✗ Difficult to scale to large numbers of parameters

Nondifferentiable Objective

$$\theta^* = \arg\min_{\theta} \boxed{J(\pi_\theta)}$$

nondifferentiable

$$\nabla_\theta J(\pi_\theta)$$

Can we approximate?

Finite-Differences

- Start with initial guess θ^0
- Approximate partial derivatives using finite-differences

$$\frac{\partial J}{\partial \theta_j} \approx \frac{J(\pi_{\theta+\epsilon_j}) - J(\pi_{\theta-\epsilon_j})}{2\epsilon}$$

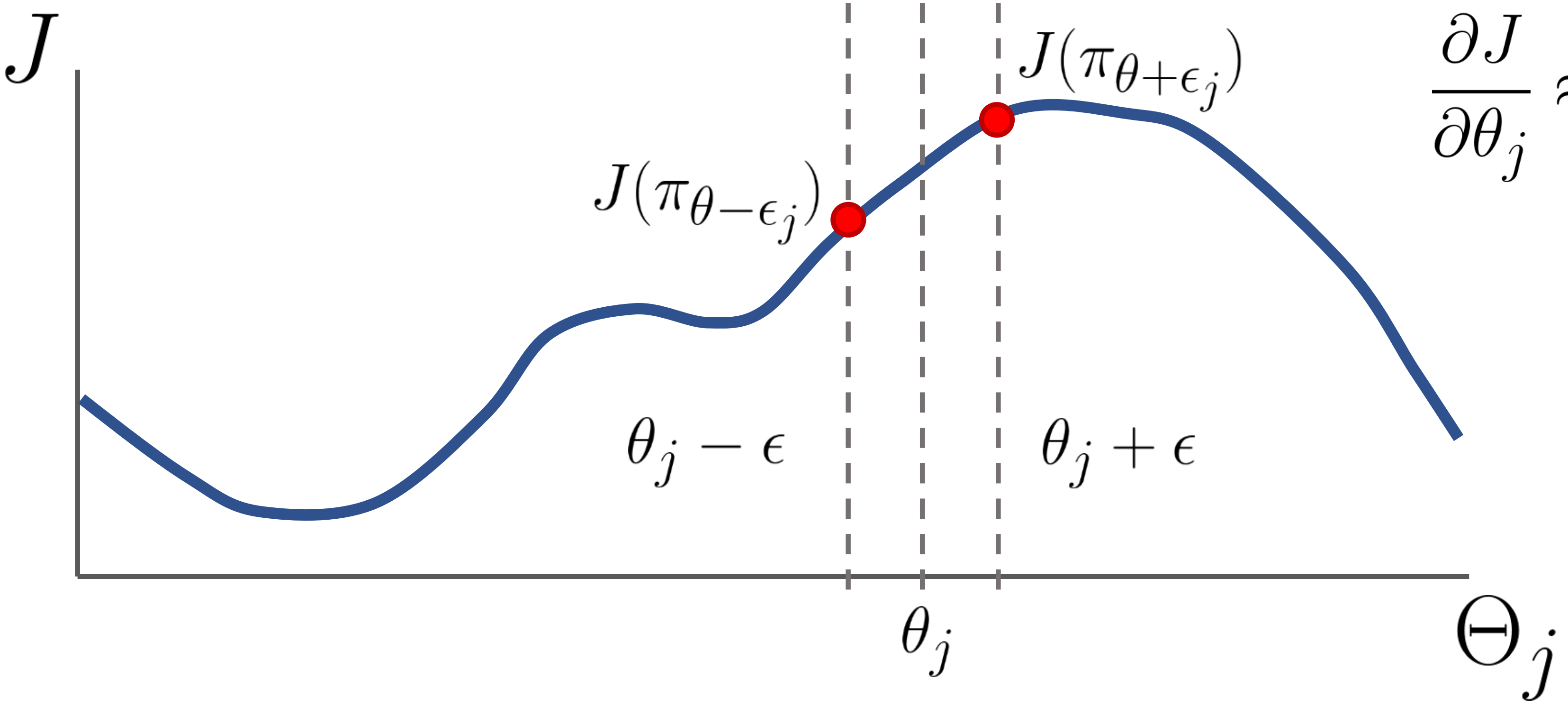
$$\epsilon_j = \begin{pmatrix} 0 \\ \vdots \\ \boxed{\epsilon} \\ \vdots \\ 0 \end{pmatrix}$$

jth component

very small

Finite-Differences

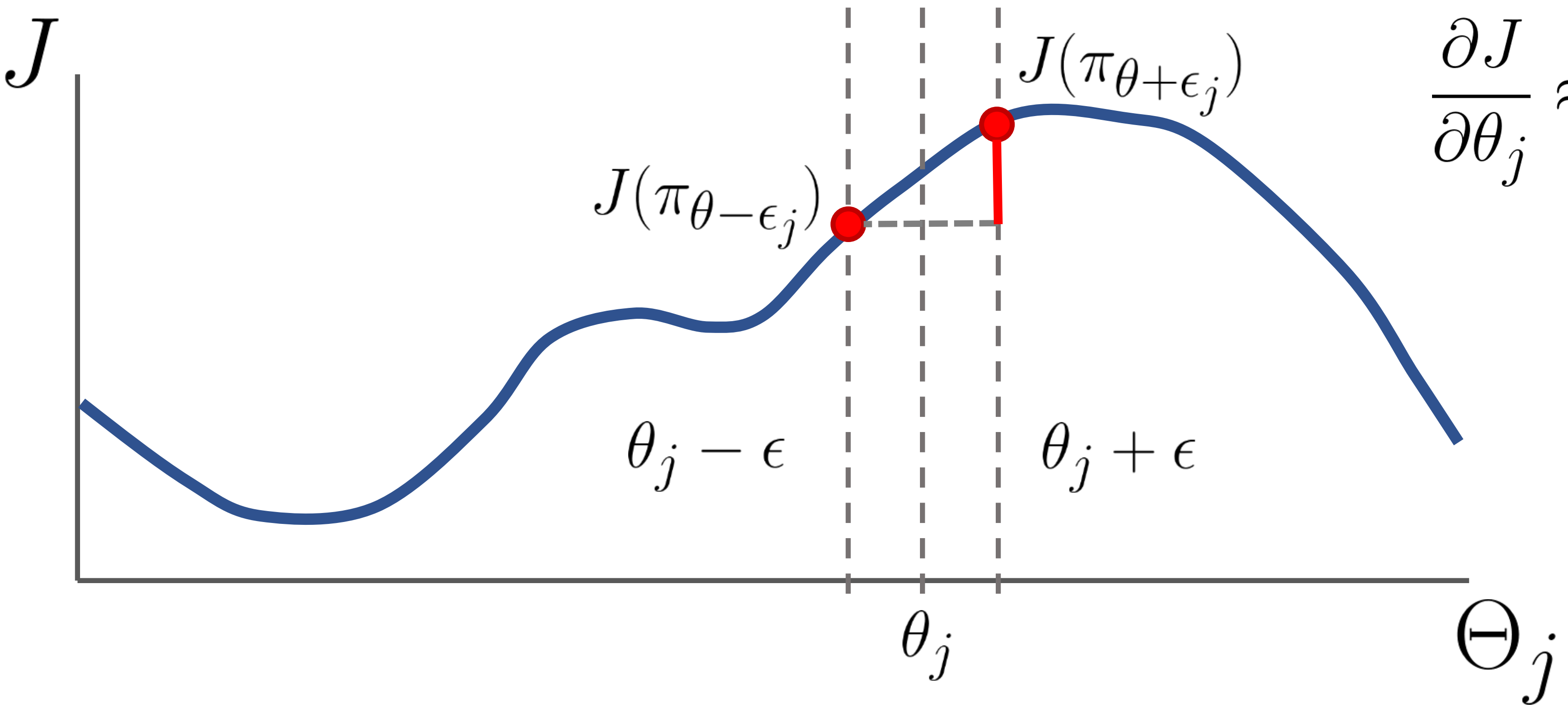
- Start with initial guess θ^0
- Approximate partial derivatives using finite-differences



$$\frac{\partial J}{\partial \theta_j} \approx \frac{J(\pi_{\theta+\epsilon_j}) - J(\pi_{\theta-\epsilon_j})}{2\epsilon}$$

Finite-Differences

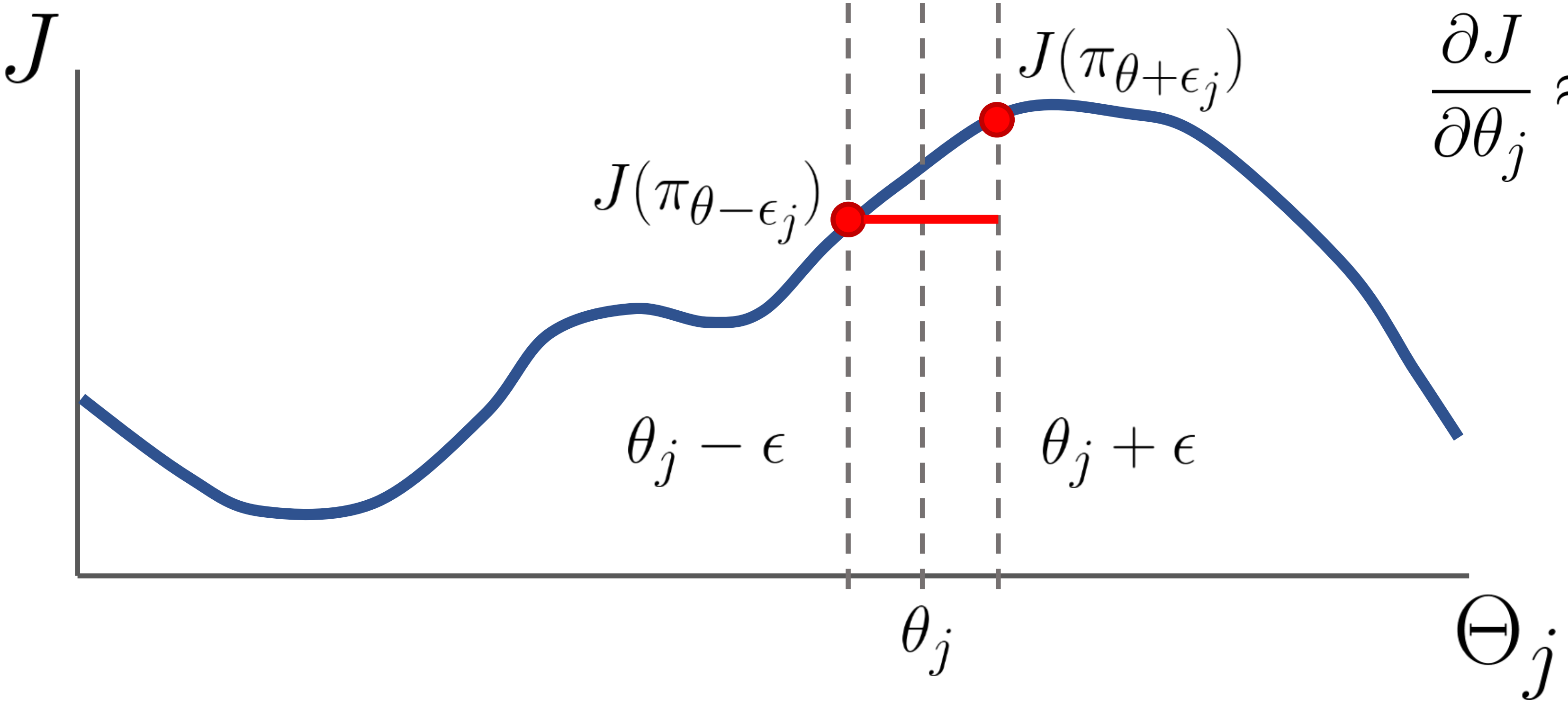
- Start with initial guess θ^0
- Approximate partial derivatives using finite-differences



$$\frac{\partial J}{\partial \theta_j} \approx \frac{\boxed{J(\pi_{\theta+\epsilon_j}) - J(\pi_{\theta-\epsilon_j})}}{2\epsilon}$$

Finite-Differences

- Start with initial guess θ^0
- Approximate partial derivatives using finite-differences



$$\frac{\partial J}{\partial \theta_j} \approx \frac{J(\pi_{\theta+\epsilon_j}) - J(\pi_{\theta-\epsilon_j})}{\boxed{2\epsilon}}$$

Finite-Differences

- Start with initial guess θ^0
- Approximate partial derivatives using finite-differences

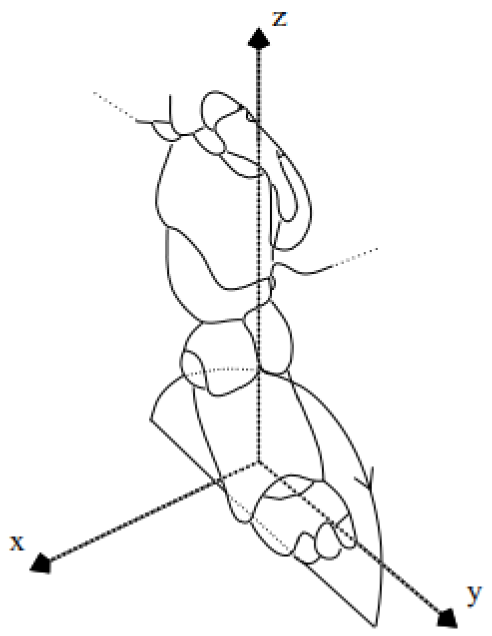
$$\triangle_j = \frac{J(\pi_{\theta+\epsilon_j}) - J(\pi_{\theta-\epsilon_j})}{2\epsilon}$$

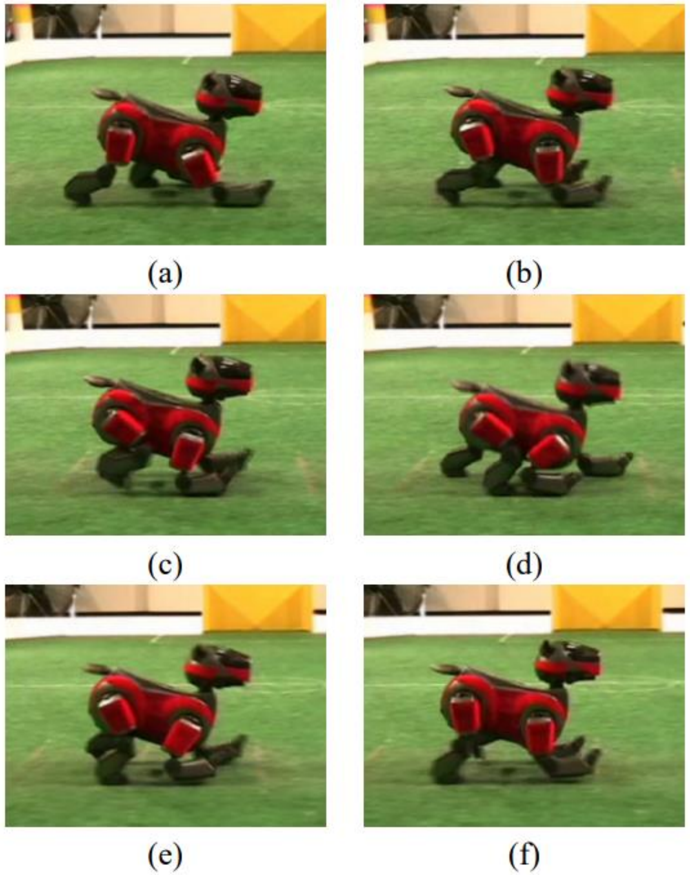
for every j

- Update: $\theta \leftarrow \theta + \alpha \triangle$

$$\triangle = \begin{pmatrix} \triangle_1 \\ \vdots \\ \triangle_n \end{pmatrix}$$

Finite-Differences





Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion
[Kohl and Stone 2004]

Finite-Differences

- Start with initial guess θ^0
- Approximate partial derivatives using finite-differences

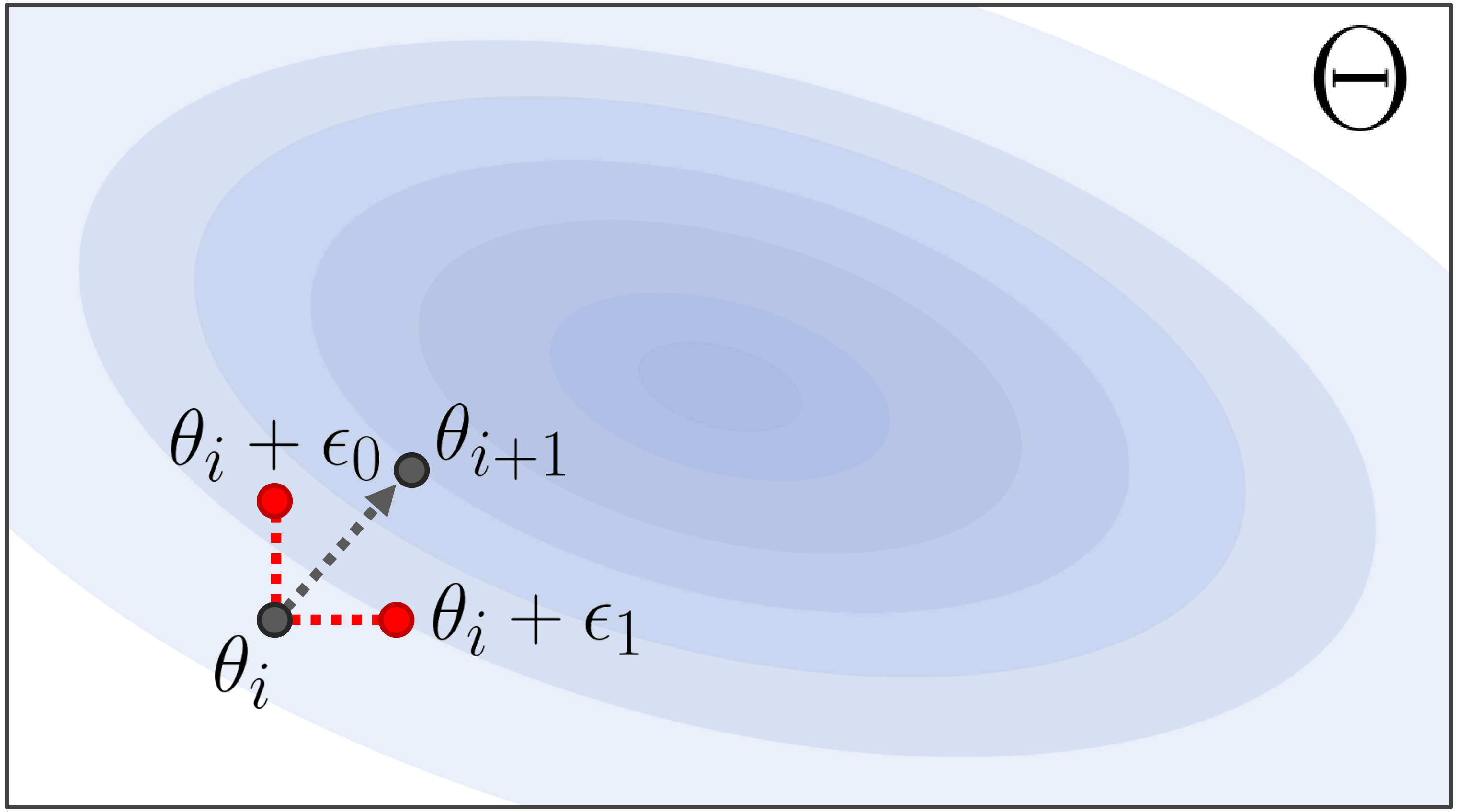
$$\triangle_j = \frac{J(\pi_{\theta+\epsilon_j}) - J(\pi_{\theta-\epsilon_j})}{2\epsilon}$$

- Update: $\theta \leftarrow \theta + \alpha\triangle$

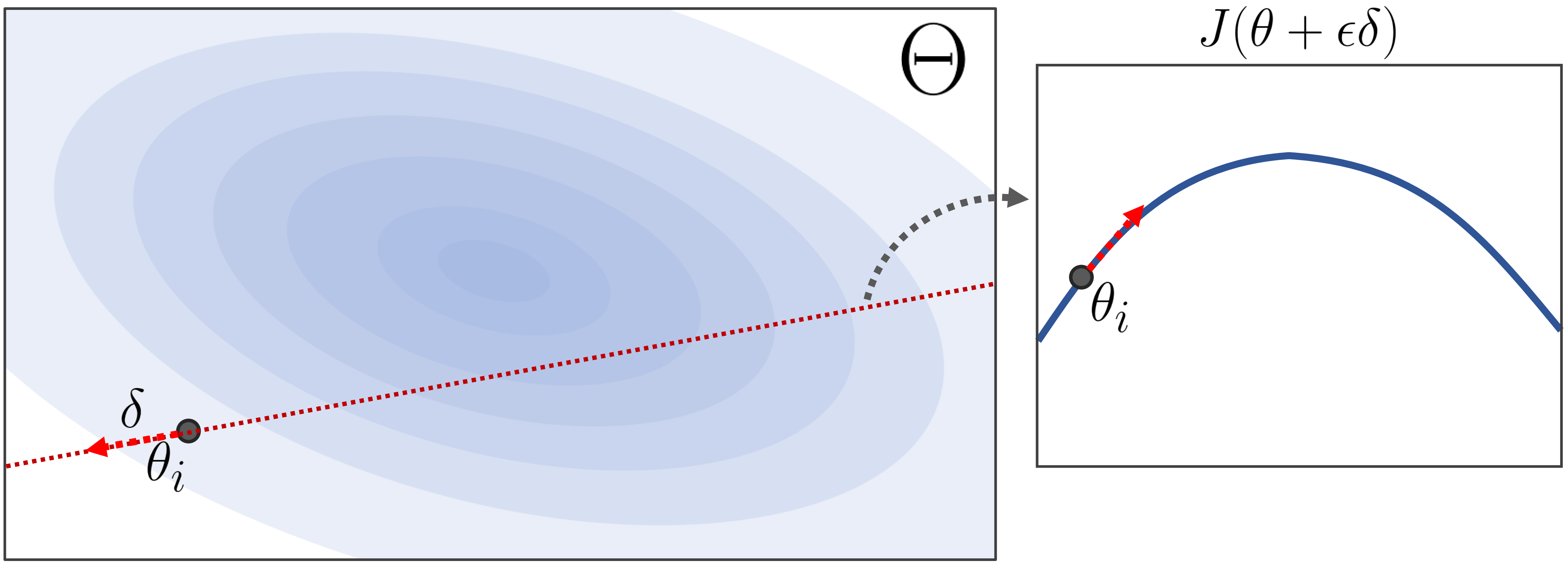
for every j

$2n$ evaluations per iterations

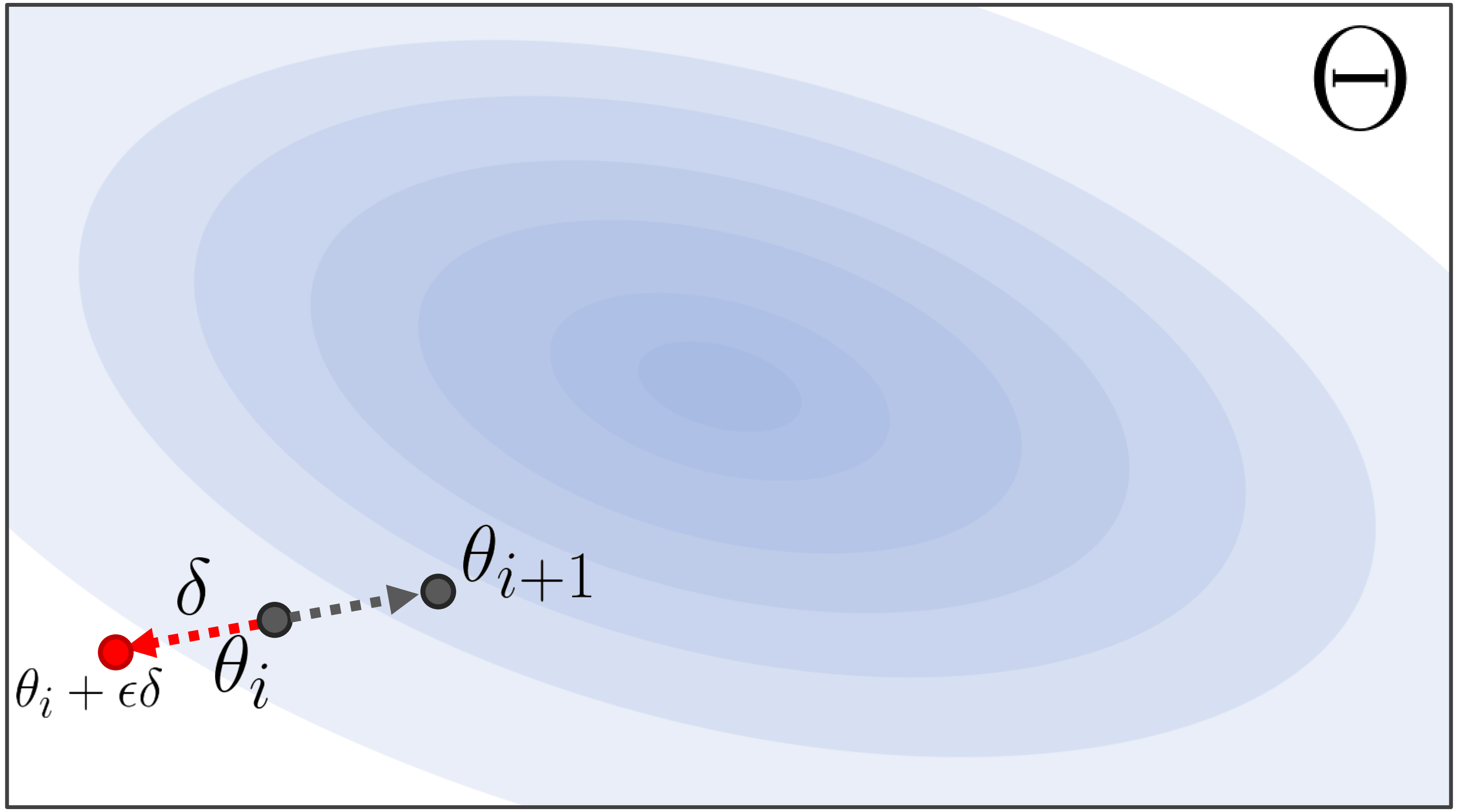
Finite-Differences



Directional Derivative



Directional Derivative



Finite-Differences (Directional Derivative)

- Start with initial guess θ^0
- Sample direction vector δ
- Approximate directional derivative

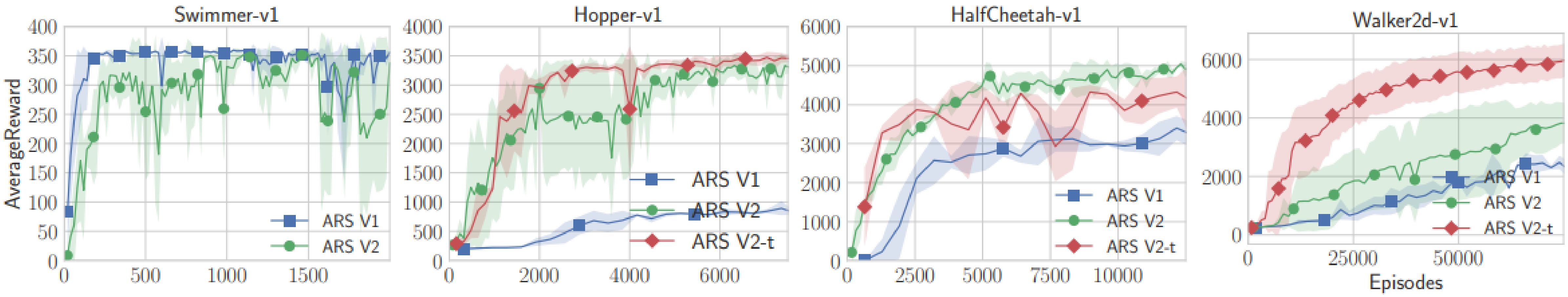
$$\triangle = \frac{J(\pi_{\theta+\epsilon\delta}) - J(\pi_{\theta-\epsilon\delta})}{2\epsilon}\,\delta$$

fewer evaluations per iteration

- Update: $\theta \leftarrow \theta + \alpha\triangle$

"directional derivative"

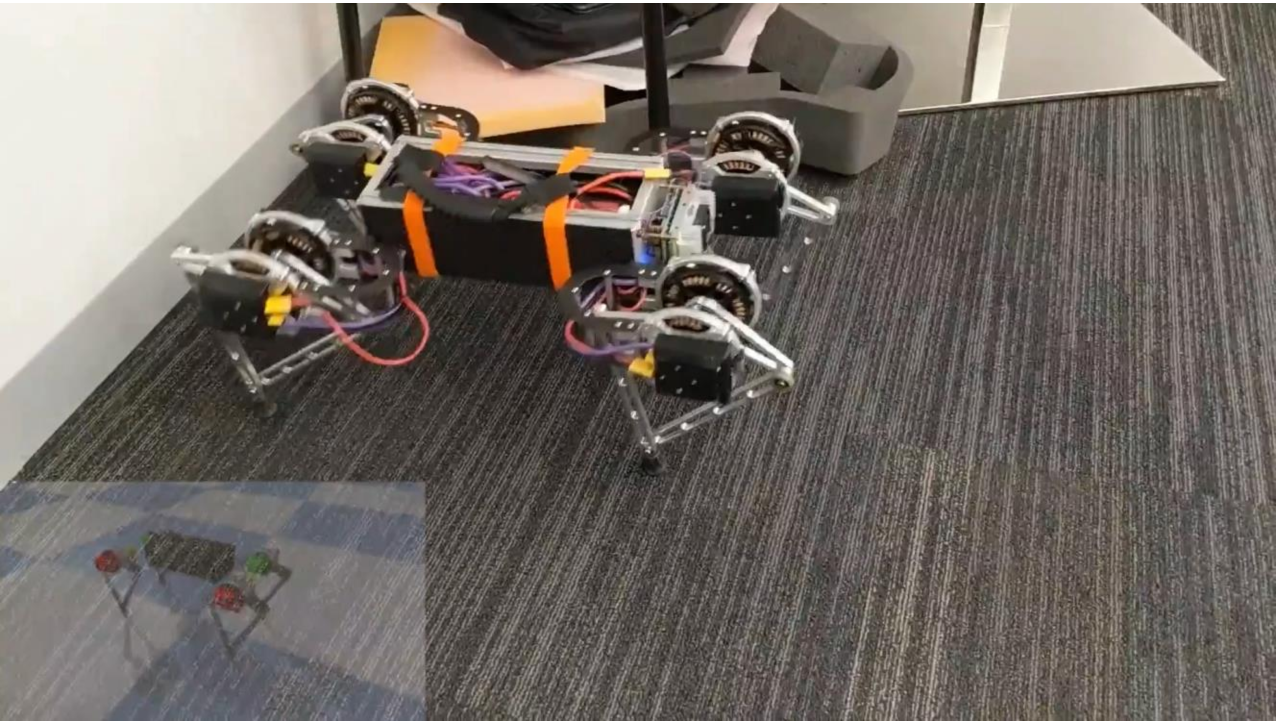
Augmented Random Search (ARS)

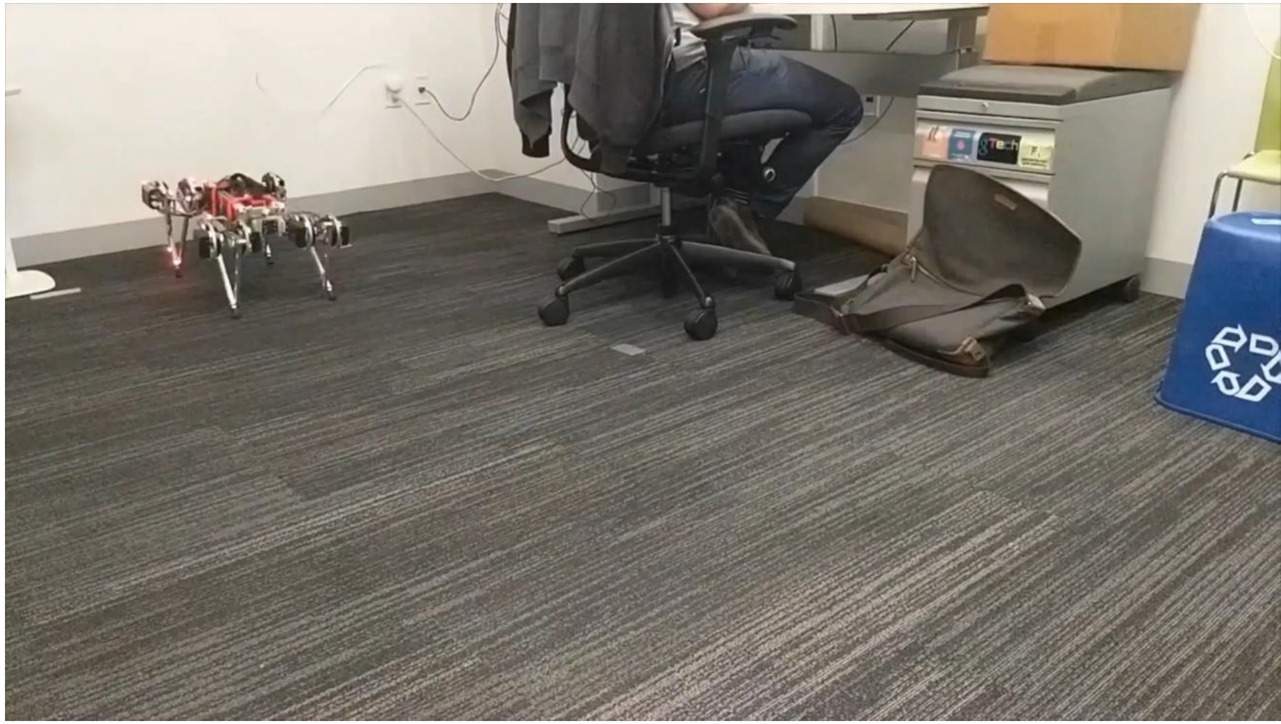


Task	# timesteps	Maximum average reward after # timesteps ARS	PPO	A2C	CEM	TRPO
Swimmer-v1	10^6	361	≈ 110	≈ 30	≈ 0	≈ 120
Hopper-v1	10^6	3047	≈ 2300	≈ 900	≈ 500	≈ 2000
HalfCheetah-v1	10^6	2345	≈ 1900	≈ 1000	≈ -400	≈ 0
Walker2d-v1	10^6	894	≈ 3500	≈ 900	≈ 800	≈ 1000

Simple Random Search Provides a Competitive Approach to Reinforcement Learning [Mania et al. 2018]

ARS Applications





Policies Modulating Trajectory Generators
[Iscen et al. 2018]

Summary

- Policy Optimization
- Black Box Optimization
- Evolutionary Methods
- Finite-Difference Methods