Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

Asynchronous RL

CMU 10703

Katerina Fragkiadaki

Non-stationary data problem for Deep RL

- Stability of training neural networks requires the **gradient updates to be de-correlated**
- This is not the case if data arrives **sequentially**
- Gradient updates computed from some part of the space can cause the value (Q) function approximator to **oscillate**
- Our solution so far has been: **Experience buffers** where experience tuples are mixed and sampled from. Resulting sampled batches are more stationary that the ones encountered online (without buffer)
- This limits deep RL to **off-policy** methods, since data from an older policy are used to update the weights of the value approximator.

Asynchronous Deep RL

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

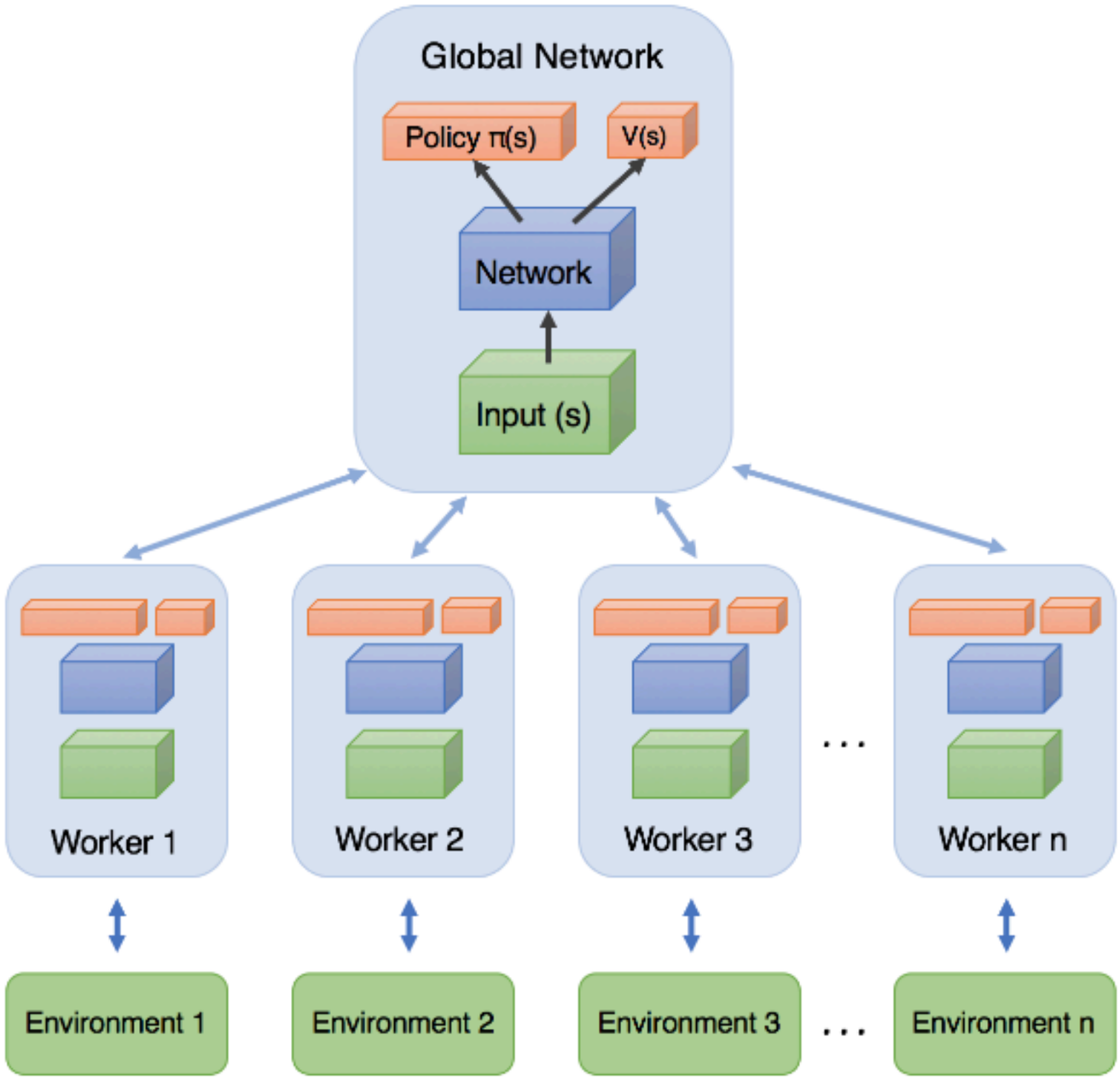
- Alternative: parallelize the collection of experience and stabilize training **without experience buffers**!

- **Multiple threads of experience**, one per agent, each exploring in different part of the environment contributing experience tuples

- **Different exploration strategies** (e.g., various \epsilon values) in different threads increase diversity

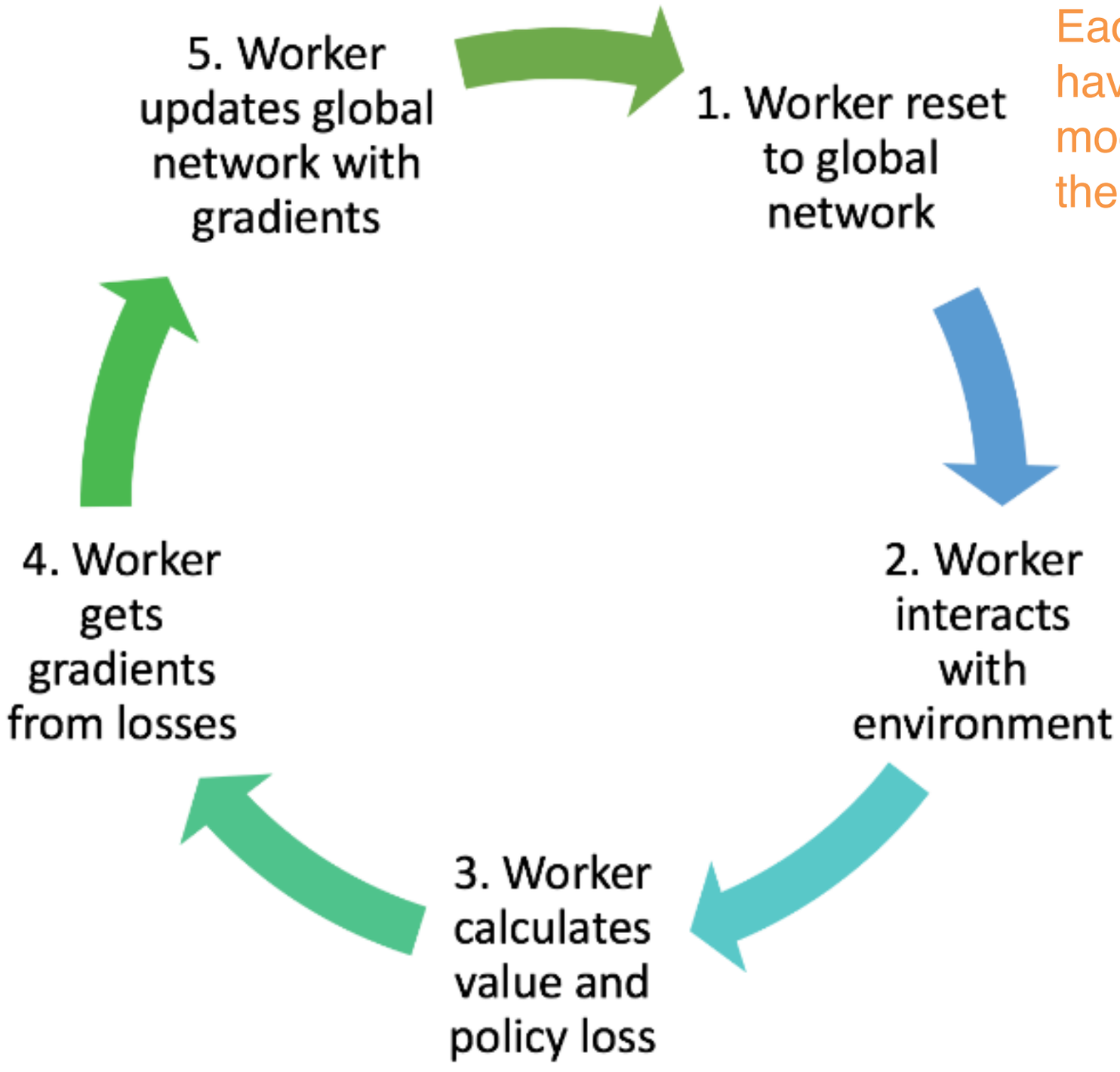
- It can be applied to **both on policy and off policy** methods, applied it to SARSA, DQN, and advantage actor-critic

Distributed RL



Distributed Asynchronous RL

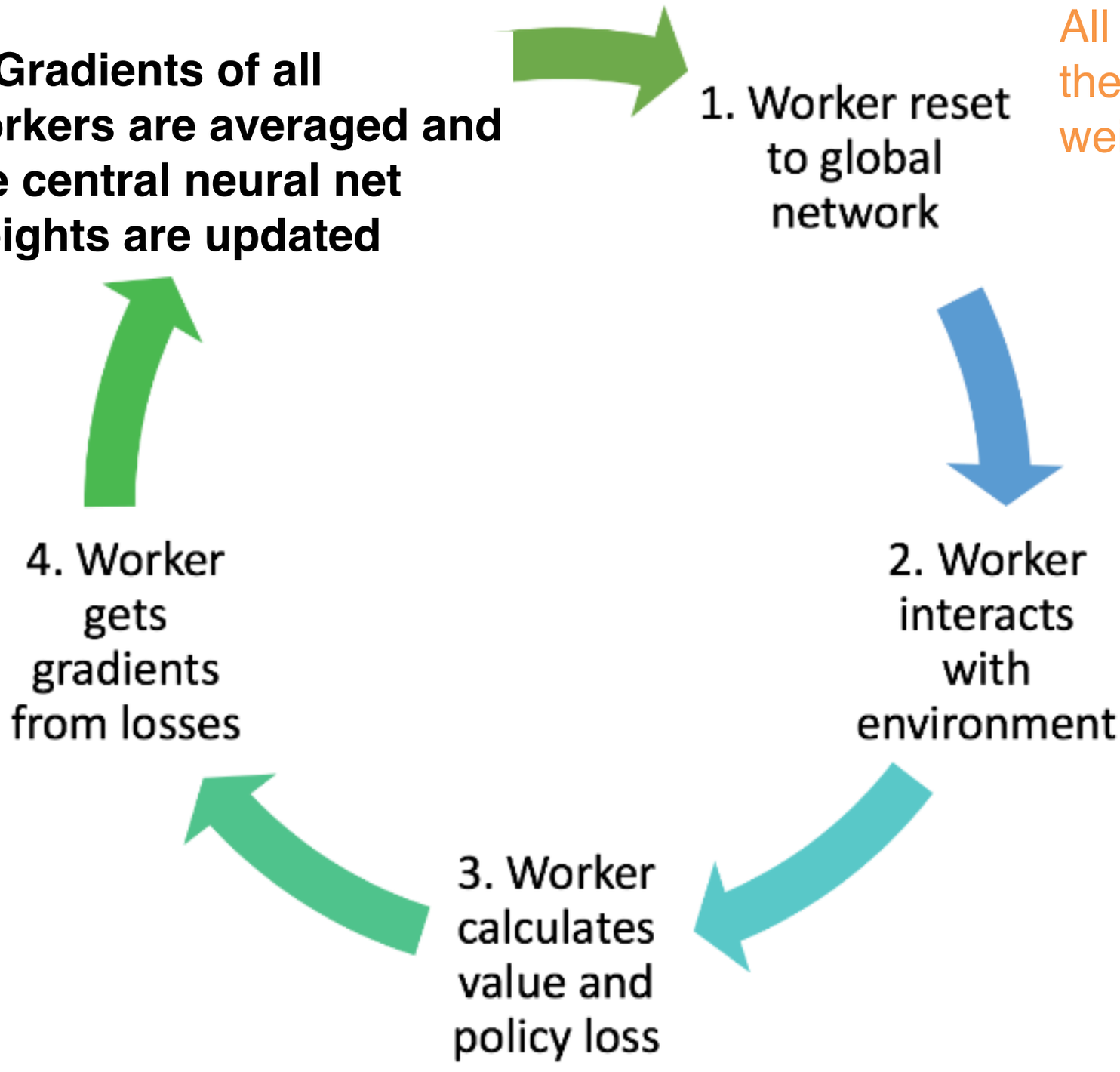
No locking



Each worker may have slightly modified version of the policy/critic

The actor critic trained in such asynchronous way is knows as A3C

Distributed Synchronous RL



All worker may have the same actor/critic weights

The actor critic trained in such synchronous way is knows as A2C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A3C

s_1, s_2, s_3, s_4

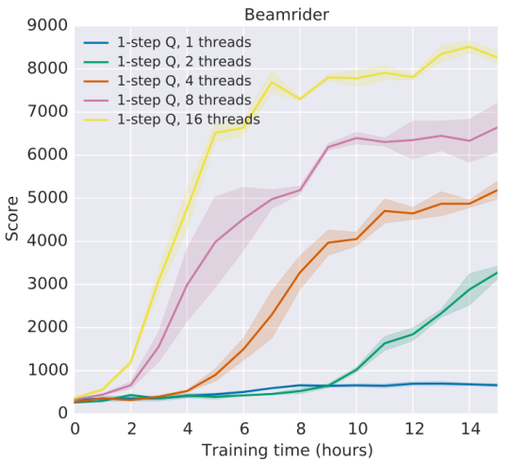
r_1, r_2, r_3

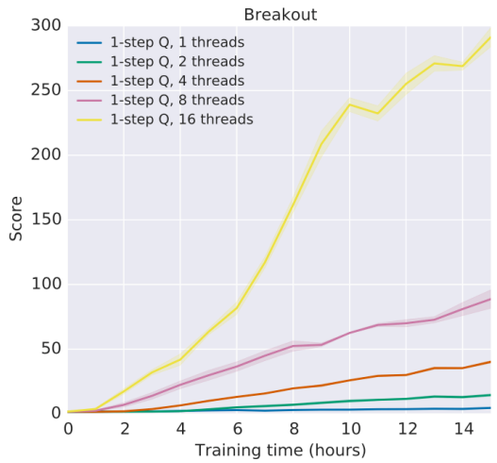
What is the approximation used for the advantage?

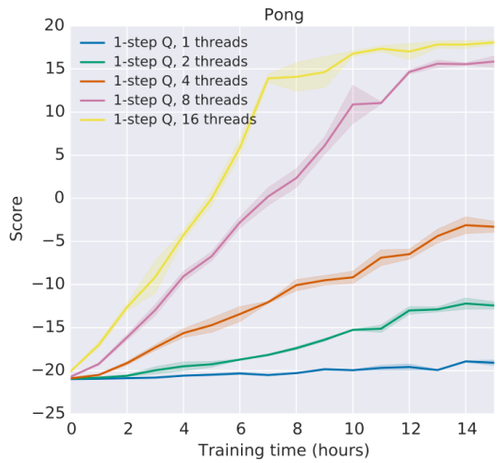
$R_3 = r_3 + \gamma V(s_4, \theta'_v)$ $\qquad$ $A_3 = R_3 - V(s_3; \theta'_v)$

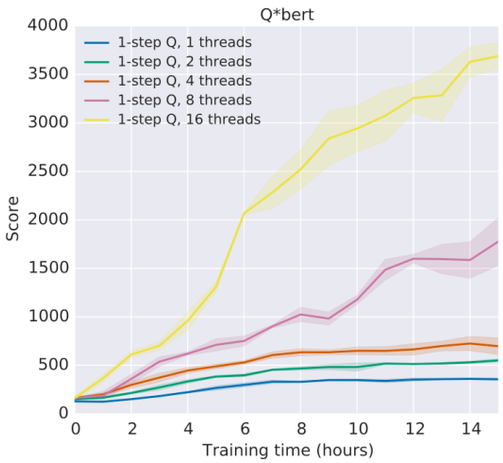
$R_2 = r_2 + \gamma r_3 + \gamma^2 V(s_4, \theta'_v)$ $\qquad$ $A_2 = R_2 - V(s_2; \theta'_v)$

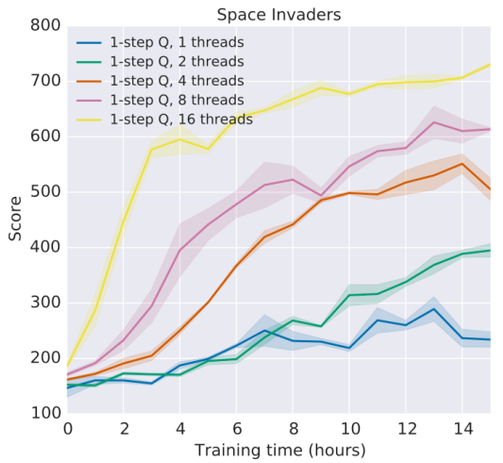
Advantages of Asynchronous (multi-threaded) RL

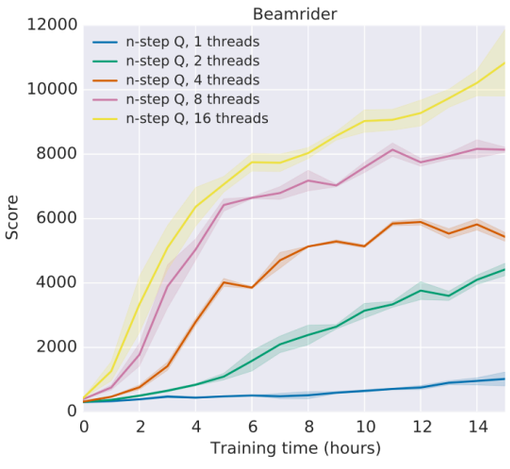


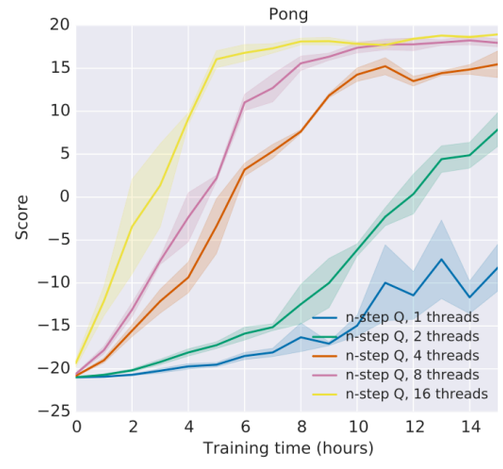


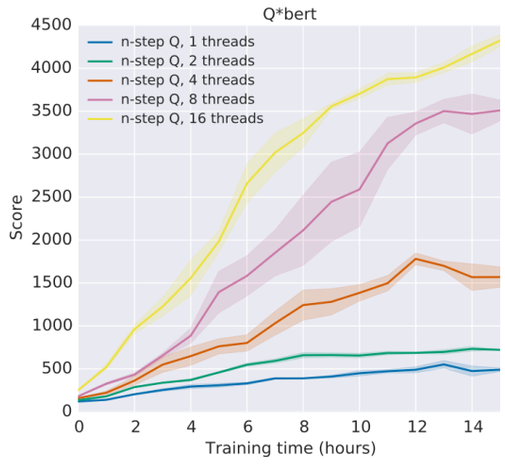


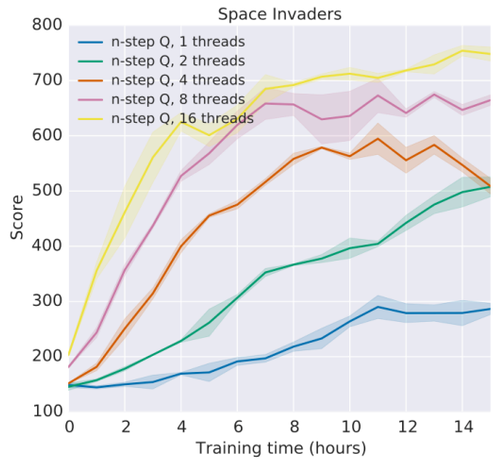


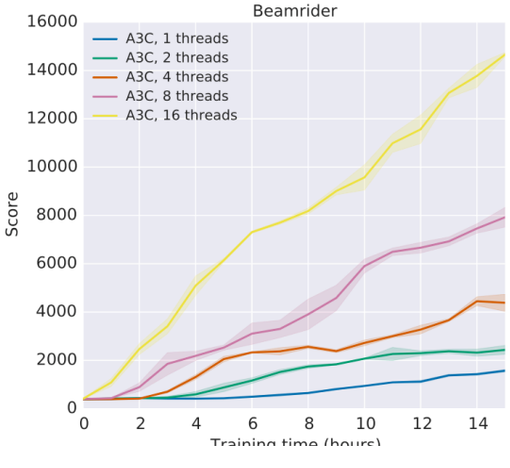


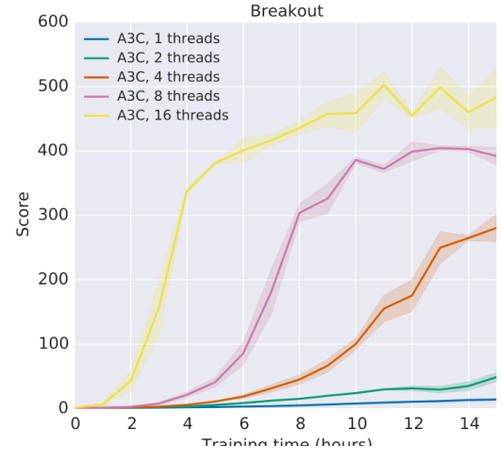


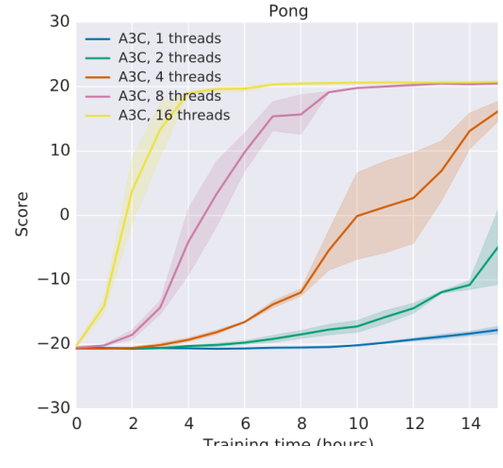


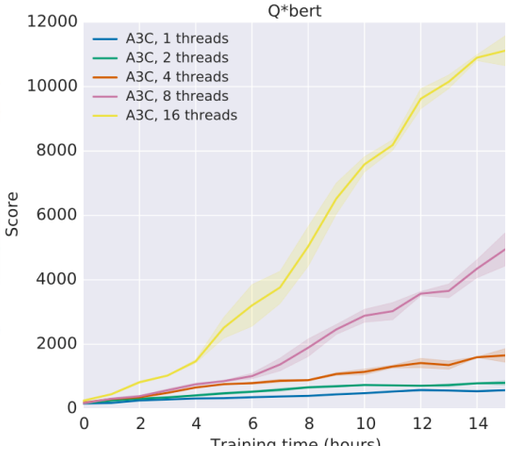


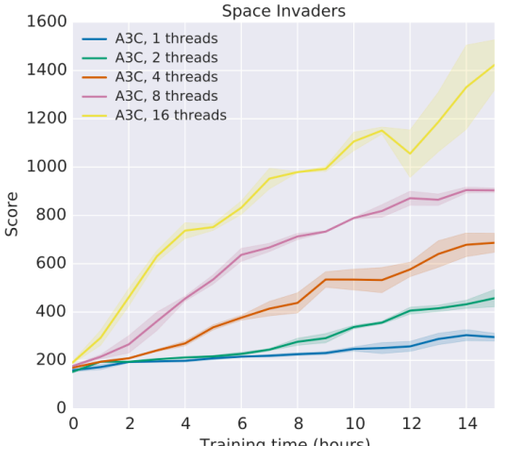












Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

Evolutionary Methods

CMU 10703

Katerina Fragkiadaki

Part of the slides borrowed by Xi Chen, Pieter Abbeel, John Schulman

Policy Optimization

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$

τ :a trajectory

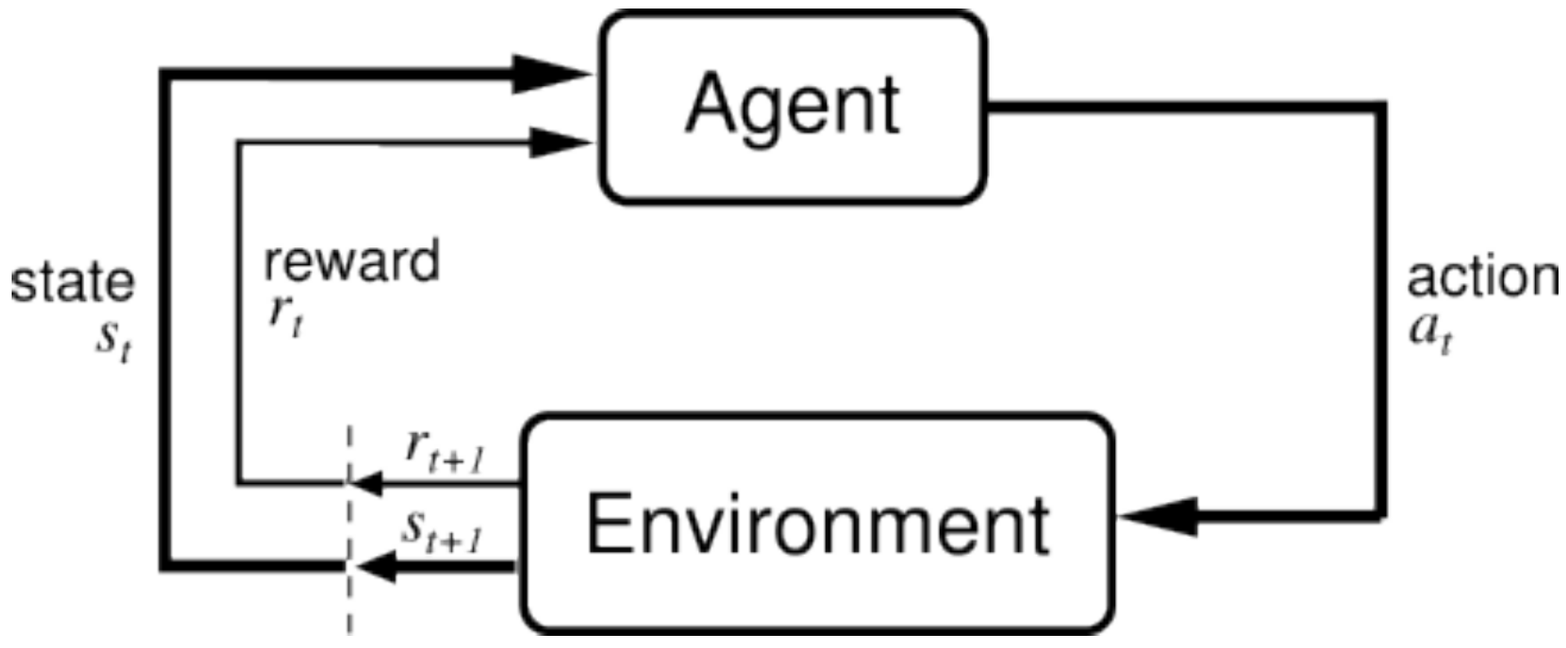
Policy Optimization

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$

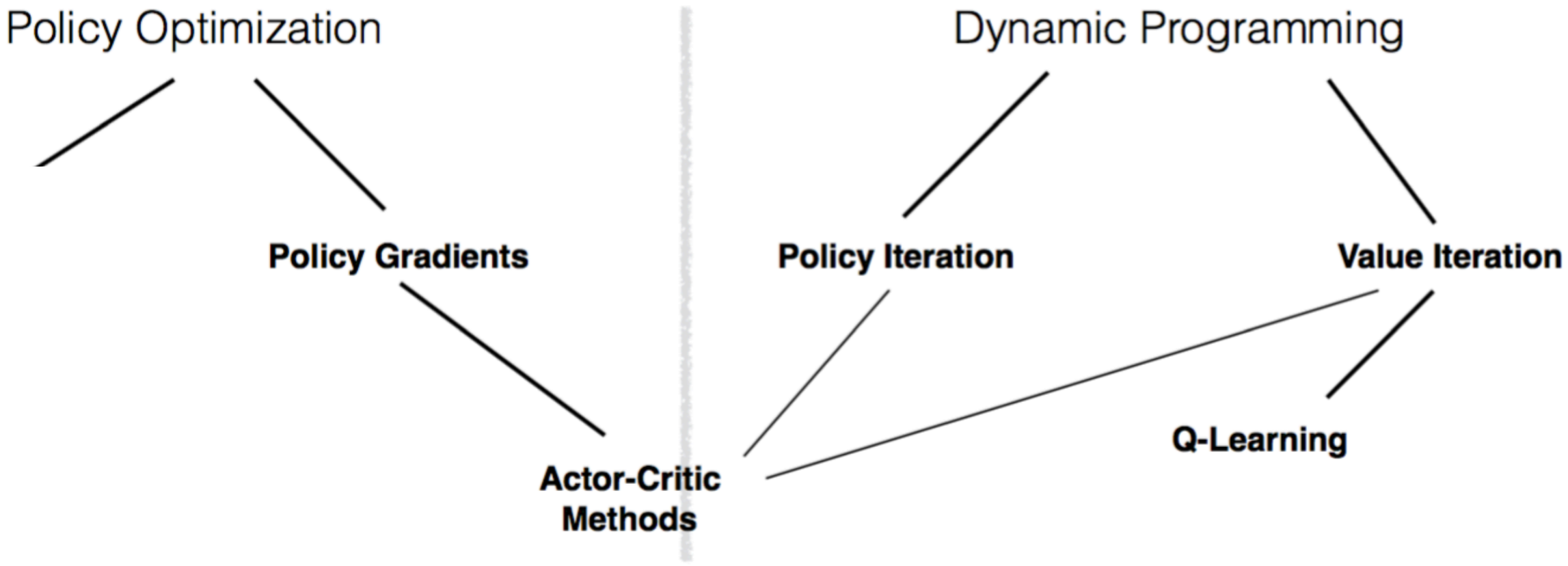
τ :a trajectory

Policy Optimization and RL

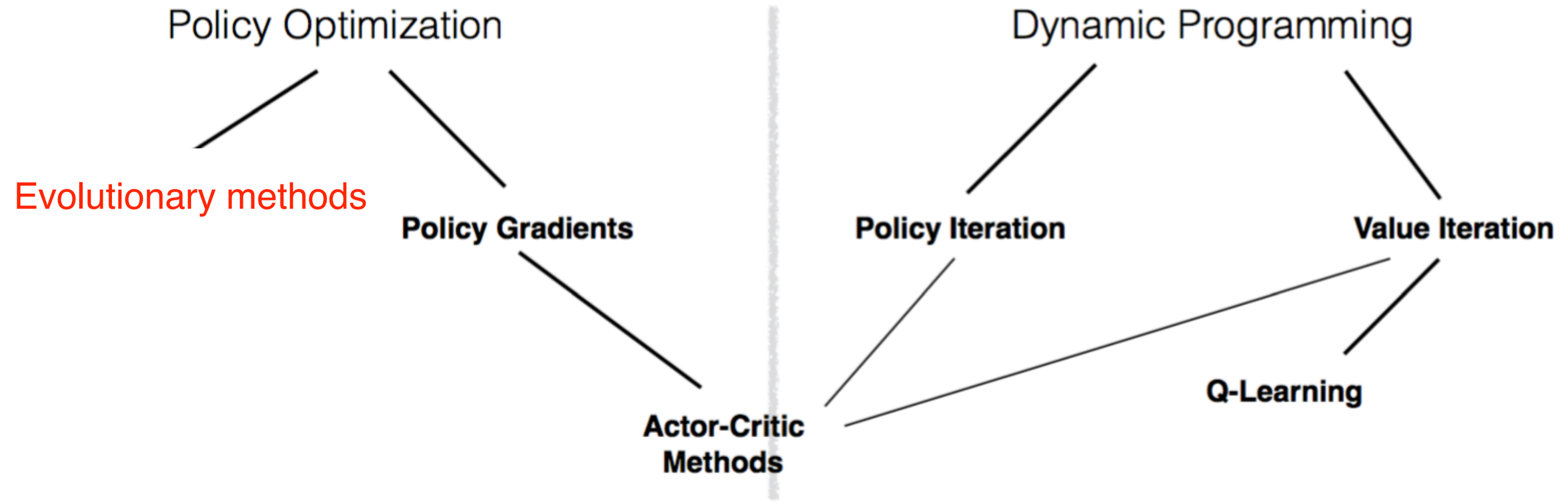
$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right] = \max_{\theta} \mathbb{E}\left[\sum_{t=0}^{T} R(s_t) \,|\, \pi_\theta, \mu_0(s)\right]$$



$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right] = \max_{\theta} \mathbb{E}\left[\sum_{t=0}^{T} R(s_t) \,|\, \pi_\theta, \mu_0(s)\right]$$



$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$



Black-box Policy Optimization

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$

No information regarding the structure of the reward

Evolutionary methods

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}\left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0)\right]$$

General algorithm:

Initialize a population of parameter vectors (*genotypes*)

1. Make random perturbations (*mutations*) to each parameter vector
2. Evaluate the perturbed parameter vector (*fitness*)
3. Keep the perturbed vector if the result improves (*selection*)
4. GOTO 1

Biologically plausible…

Cross-entropy method

Let's consider our parameters to be sampled from a multivariate isotropic Gaussian
We will evolve this Gaussian towards sampled that have highest fitness

CEM:

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

for iteration = 1, 2, …

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

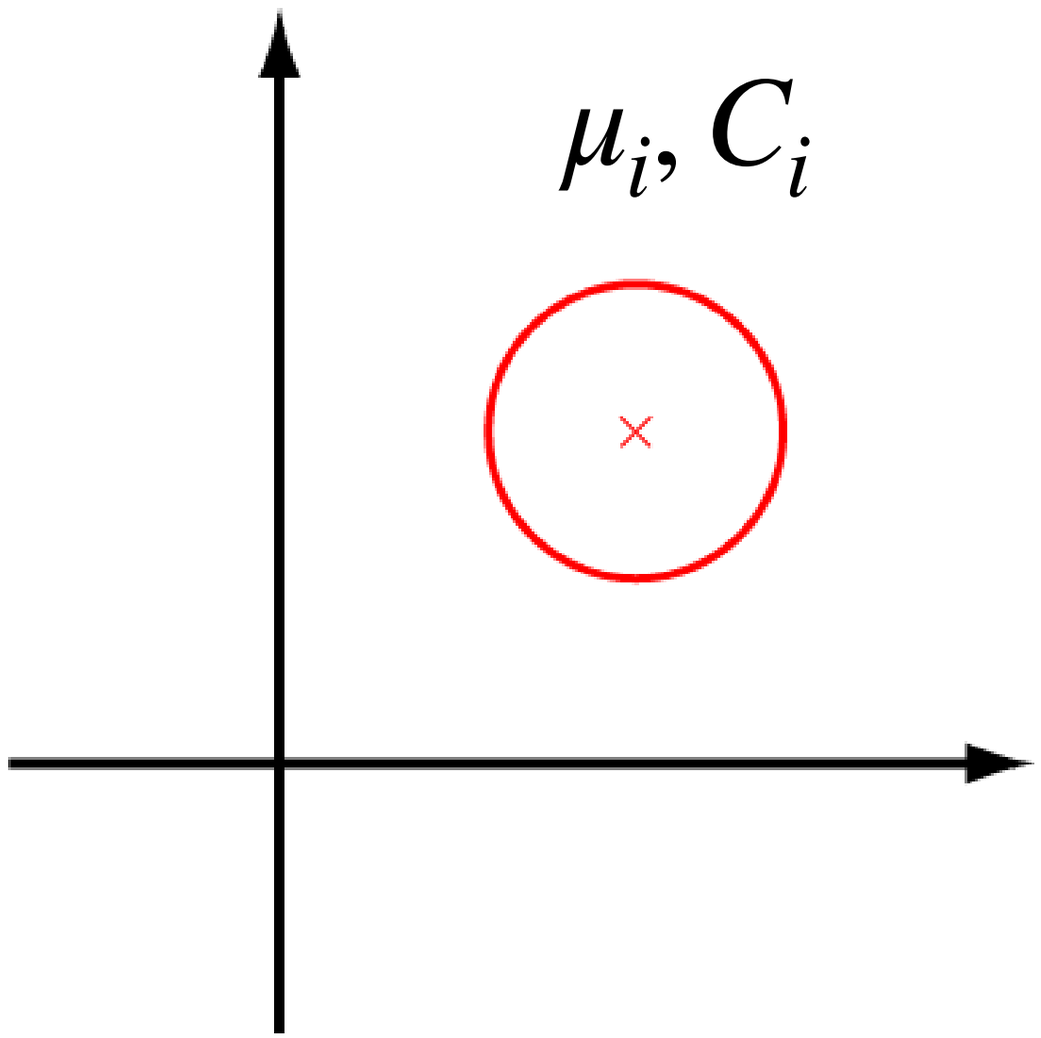
For each θ_i, perform one rollout to get return $R(\tau_i)$

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

endfor

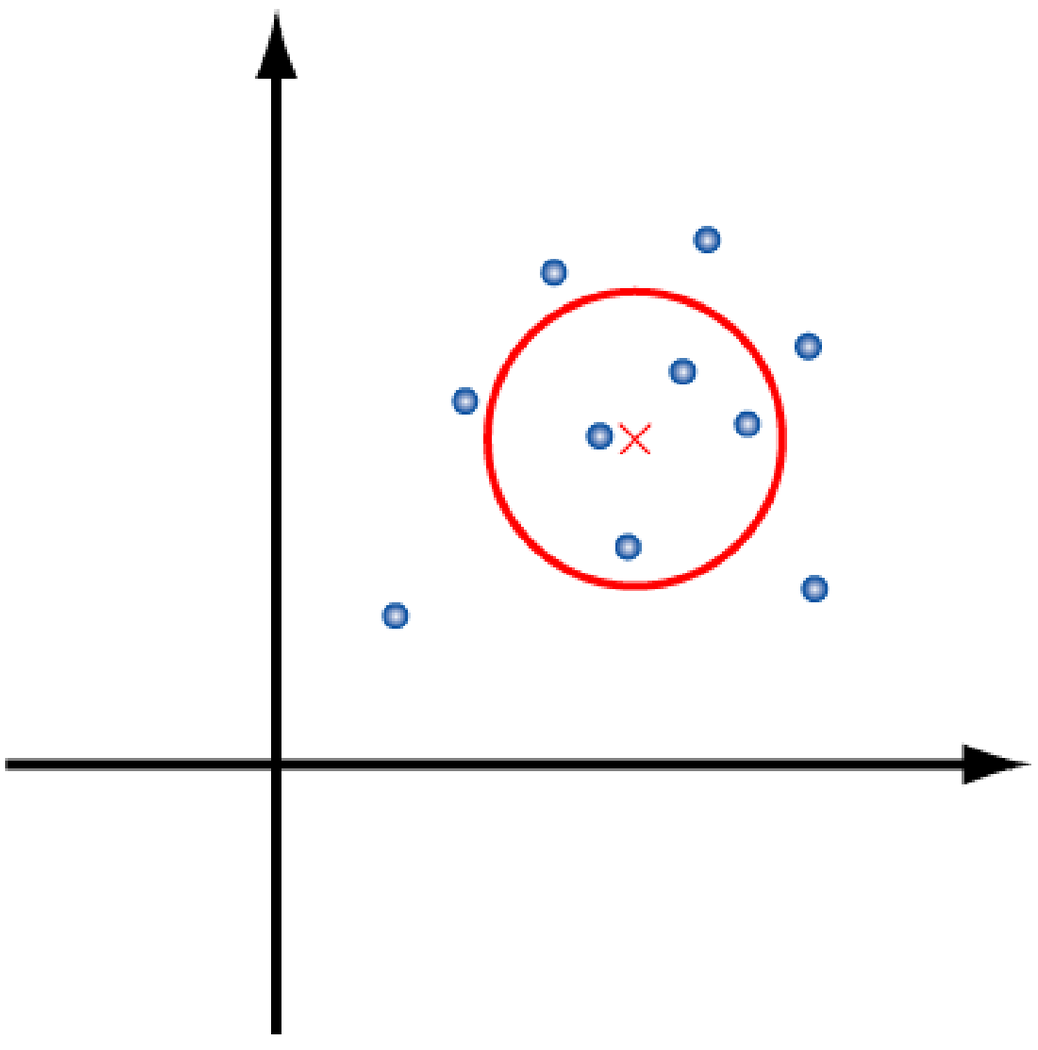
Covariance Matrix Adaptation

Let's consider our parameters to be sampled from a multivariate Gaussian
We will evolve this Gaussian towards sampled that have highest fitness



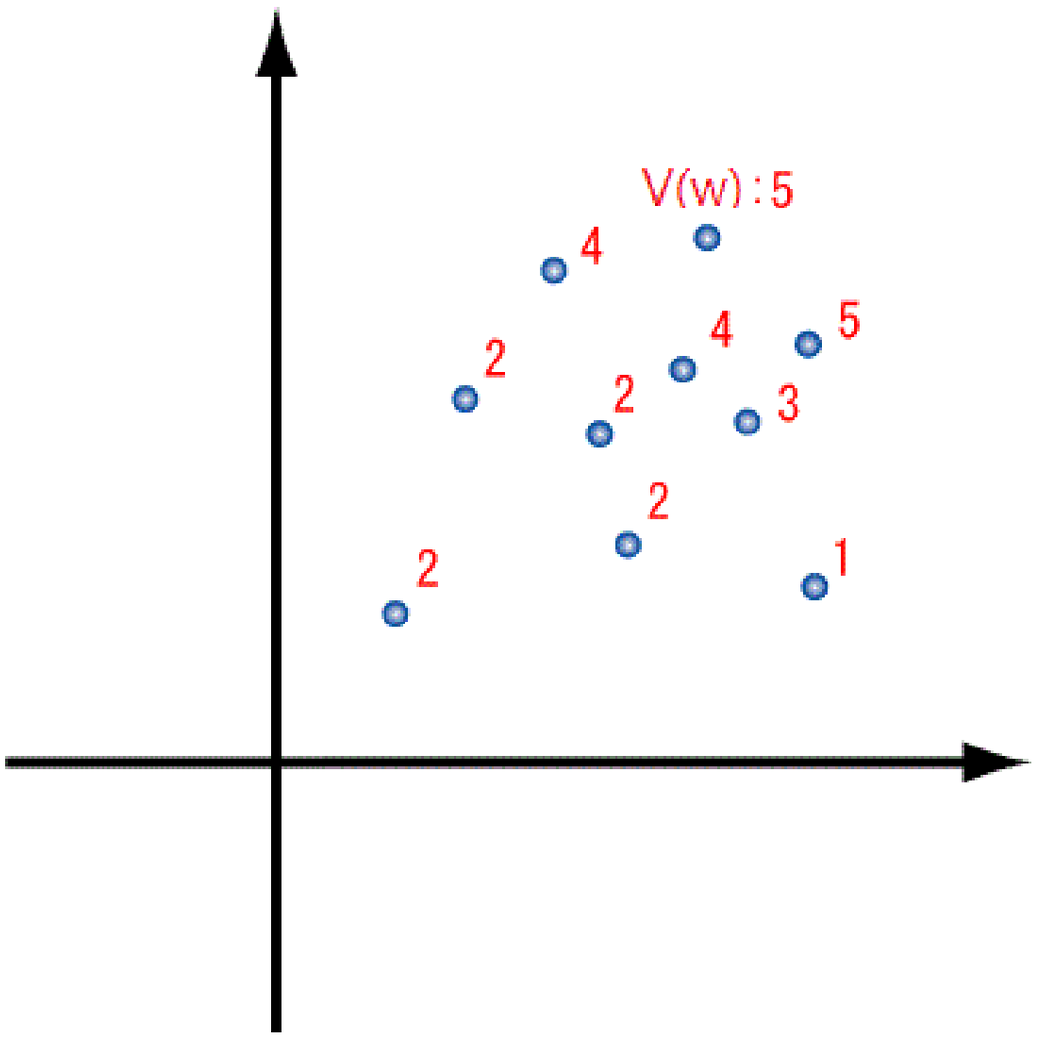
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



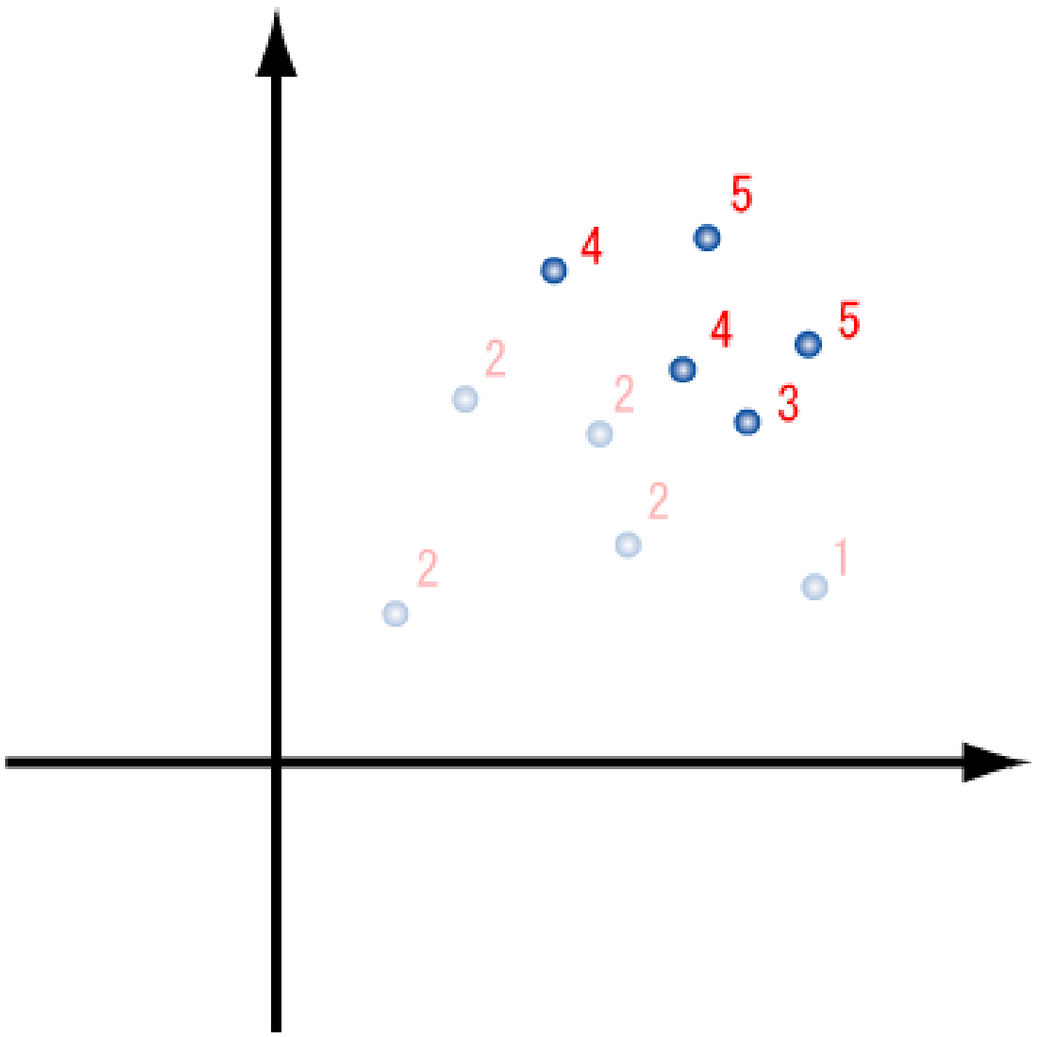
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



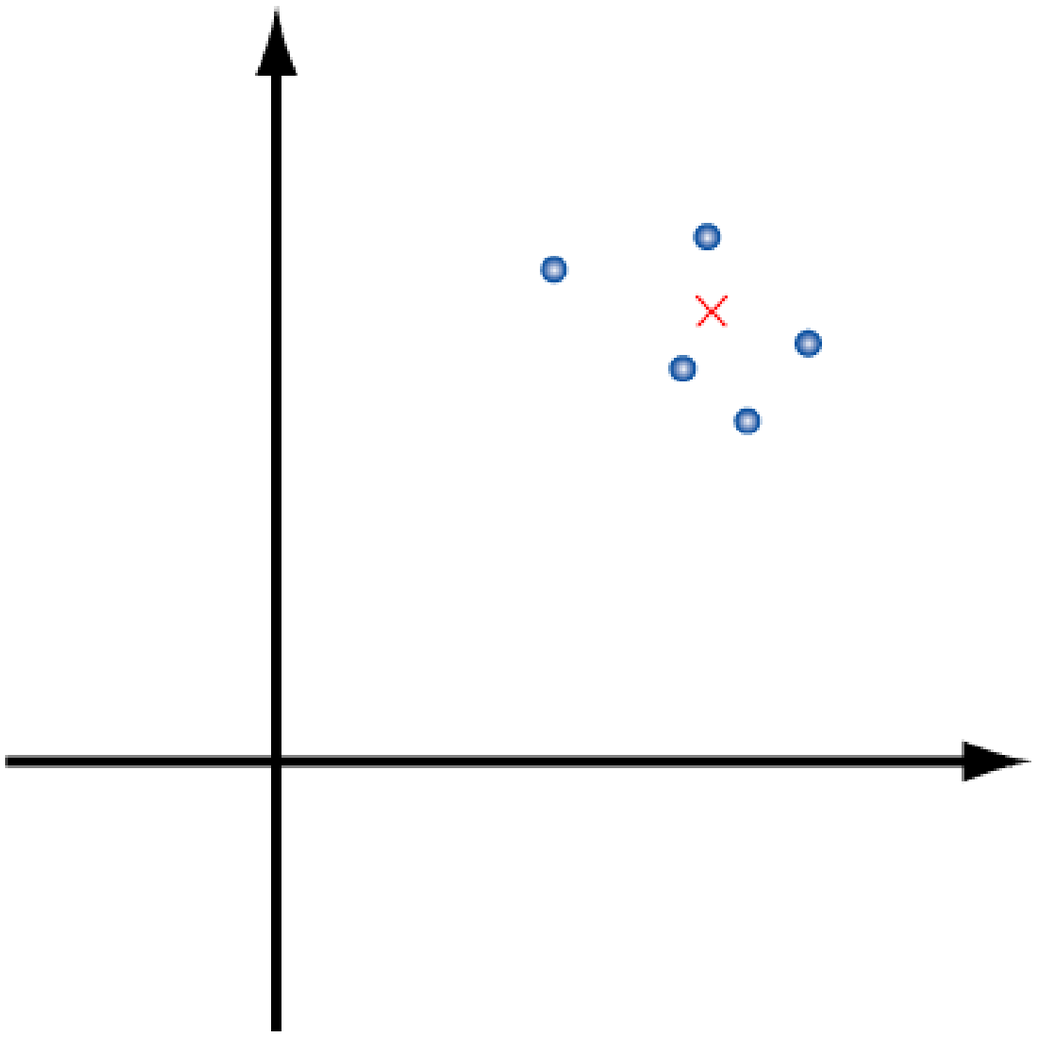
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



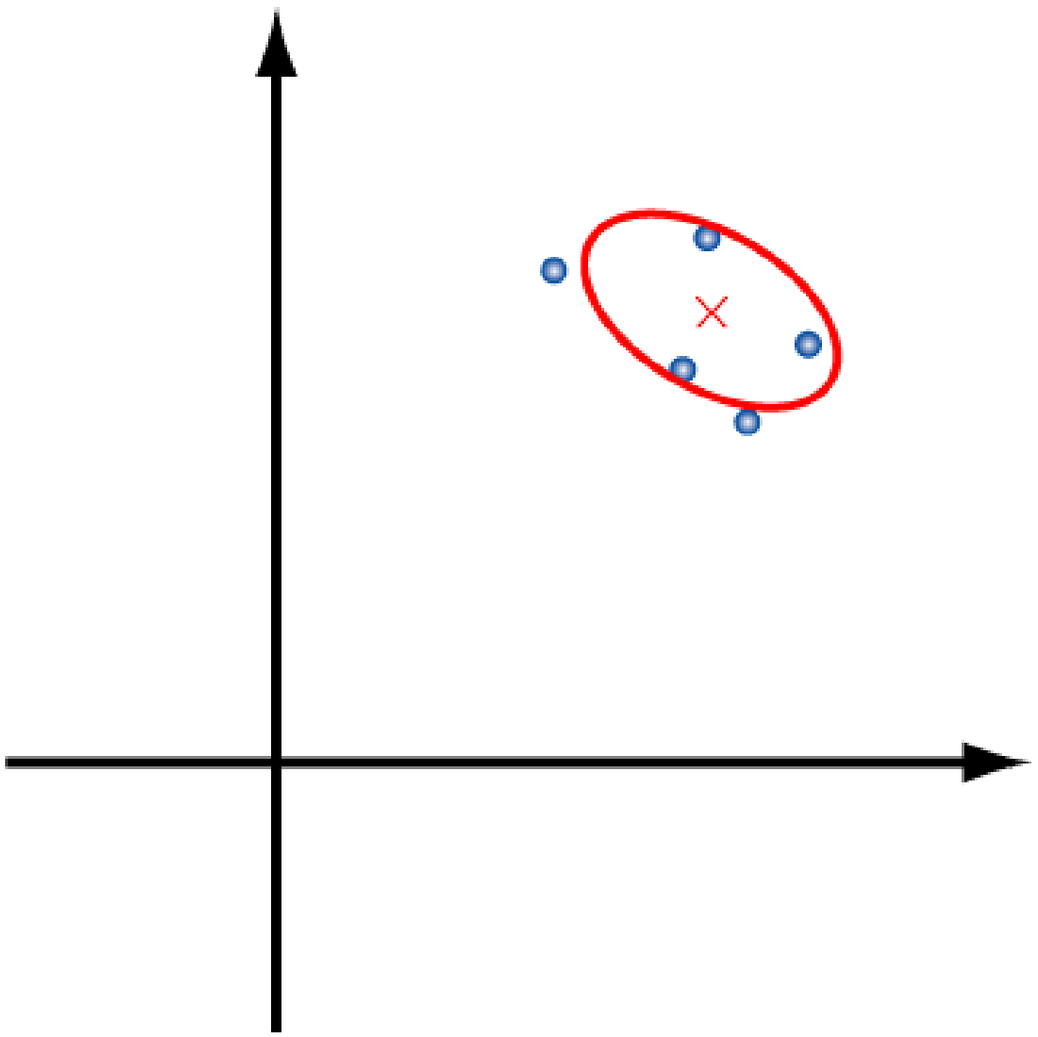
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



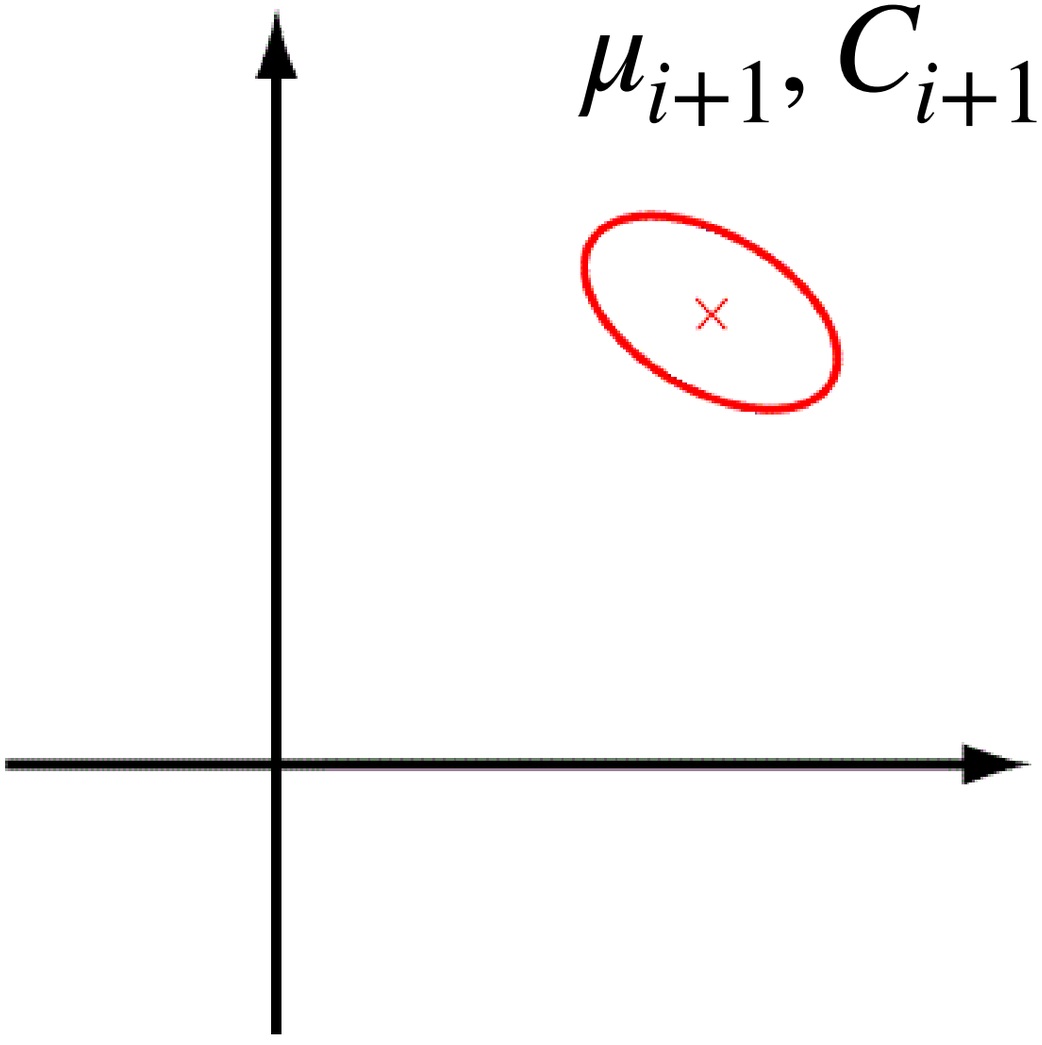
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



- Sample
- Select elites
- Update mean
- Update covariance
- iterate

CMA-ES, CEM

Work embarrassingly well in low-dimensions

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

[NIPS 2013]

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

Question

- Evolutionary methods work well on relatively low-dim problems
- Can they be used to optimize deep network policies?

PG VS ES

We are sampling in both cases…

Policy Gradients Review

$$\max_{\theta} . \; J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\mu P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau) \right]
\end{aligned}$$

Sample estimate:

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

ES

Considers distribution over policy parameters

$$\max_{\mu} . \; U(\mu) = \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right]$$

$$\begin{aligned}
\nabla_\mu U(\mu) &= \nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right] \\
&= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) F(\theta) \right]
\end{aligned}$$

ES

Considers distribution over policy parameters

$$\max_{\mu} . \; U(\mu) = \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right]$$

$$\begin{aligned}
\nabla_\mu U(\mu) &= \nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right] \\
&= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) F(\theta) \right]
\end{aligned}$$

Sample estimate:

$$\nabla_\mu U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta^{(i)}) F(\theta^{(i)})$$

PG

Considers distribution over actions

$$\max_{\theta} . \; J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\mu P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau) \right]
\end{aligned}$$

Sample estimate:

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

ES

Considers distribution over policy parameters

$$\max_{\mu} . \; U(\mu) = \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right]$$

$$\begin{aligned}
\nabla_\mu U(\mu) &= \nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right] \\
&= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) F(\theta) \right]
\end{aligned}$$

Sample estimate:

$$\nabla_\mu U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta^{(i)}) F(\theta^{(i)})$$

From trajectories to actions

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log P_\theta(\tau^{(i)}) R(\tau^{(i)}) \quad \Rightarrow \quad \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) R(s_t^{(i)}, a_t^{(i)})$$

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[\prod_{t=0}^{T} \underbrace{P(s_{t+1}^{(i)} \mid s_t^{(i)}, a_t^{(i)})}_{\text{dynamics}} \cdot \underbrace{\pi_\theta(a_t^{(i)} \mid s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{T} \underbrace{\log P(s_{t+1}^{(i)} \mid s_t^{(i)}, a_t^{(i)})}_{\text{dynamics}} + \underbrace{\log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{T} \underbrace{\log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})}_{\text{policy}} \right]$$

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})$$

PG

Considers distribution over actions

$$\max_{\theta} . \; J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\mu P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau) \right]
\end{aligned}$$

Sample estimate:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^{(i)} \,|\, s_t^{(i)}) R(s_t^{(i)}, a_t^{(i)})$$

ES

Considers distribution over policy parameters

$$\max_{\mu} . \; U(\mu) = \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right]$$

$$\begin{aligned}
\nabla_\mu U(\mu) &= \nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[F(\theta) \right] \\
&= \nabla_\mu \int P_\mu(\theta) F(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) F(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} F(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) F(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) F(\theta) \right]
\end{aligned}$$

Sample estimate:

$$\nabla_\mu U(\mu) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\mu \log P_\mu(\theta^{(i)}) F(\theta^{(i)})$$

A concrete example

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

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

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

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:

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

Sampling parameter vectors

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

Imagine we have access to random vectors $\epsilon \sim \mathcal{N}(0,I)$

$$\theta_1 = \mu + \sigma * \epsilon_1,\ \epsilon_1 \sim \mathcal{N}(0,I)$$

$$\theta_2 = \mu + \sigma * \epsilon_2,\ \epsilon_2 \sim \mathcal{N}(0,I)$$

The theta samples have the desired mean and variance

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:

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

$$\approx \frac{1}{2\sigma}\left[R(\tau_1)\epsilon_1 + R(\tau_2)\epsilon_2\right]$$

$\theta_1 = \mu + \sigma * \epsilon_1,\ \epsilon_1 \sim \mathcal{N}(0, I)$

$\theta_2 = \mu + \sigma * \epsilon_2,\ \epsilon_2 \sim \mathcal{N}(0, I)$

Natural Evolutionary Strategies

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
    Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Connection to Finite Differences

- Antithetic sampling
  - Sample a pair of policies with mirror noise $(\theta_+ = \mu + \sigma\epsilon, \theta_- = \mu - \sigma\epsilon)$

Connection to Finite Differences

- 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_-)

Connection to Finite Differences

- 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

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

Connection to Finite Differences

- 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

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

vs

Finite Difference

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

Finite Differences

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:

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

Evolution methods VS Policy Gradients

Sample estimate:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\alpha_t^{(i)} \mid s_t^{(i)}) R(s_t^{(i)}, a_t^{(i)})$$

Sample estimate:

$$\nabla_\mu U(\mu) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\mu \log P_\mu(\theta^{(i)}) F(\theta^{(i)})$$

- Open Question: Policy Gradient at action level or parameter level?

We add noise \epsilon in our **actions** (\epsilon-greedy)!

$$\nabla_\theta F_{PG}(\theta) = \mathbb{E}_\epsilon \left\{ R(\mathbf{a}(\epsilon, \theta)) \nabla_\theta \log p(\mathbf{a}(\epsilon, \theta); \theta) \right\}$$

We add noise \ksi in our policy (neural) **parameters**!

$$\nabla_\theta F_{ES}(\theta) = \mathbb{E}_\xi \left\{ R(\mathbf{a}(\xi, \theta)) \nabla_\theta \log p(\tilde{\theta}(\xi, \theta); \theta) \right\}$$

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

Neural net architectures that work well with (stochastic) gradient descent optimization do not work will with ES. Contributions:

- Virtual batch norm
- Discretization of continuous actions - better exploration during mutation!
- Parallelization with a need for tiny only cross-worker communication!!

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

Neural net architectures that work well with (stochastic) gradient descent optimization do not work will with ES. Contributions:

- Virtual batch norm
- Discretization of continuous actions - **better exploration during mutation!**
- Parallelization with a need for tiny only cross-worker communication!!

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

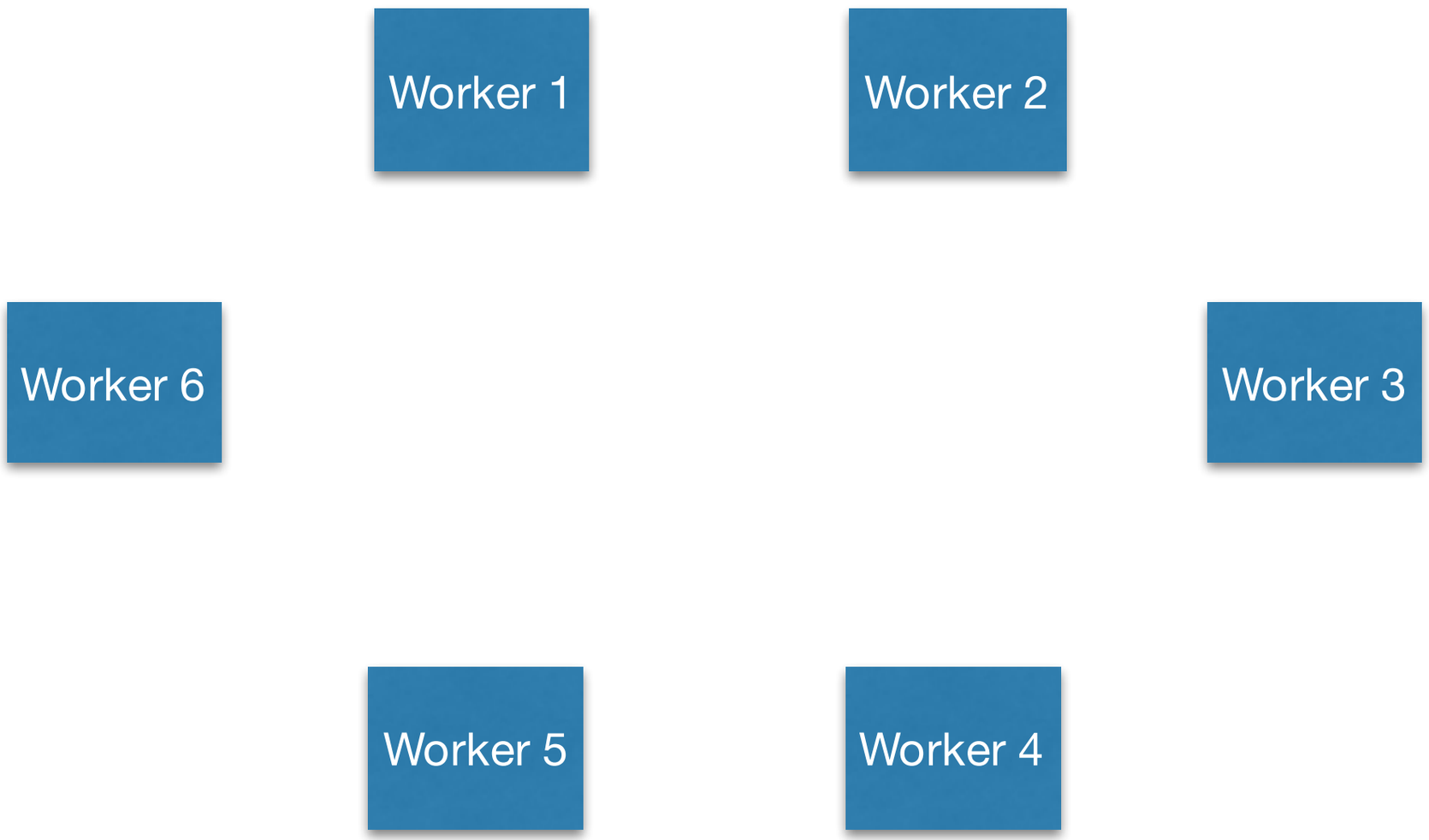
Neural net architectures that work well with (stochastic) gradient descent optimization do not work will with ES. Contributions:

- Virtual batch norm
- Discretization of continuous actions - better exploration during mutation!
- Parallelization with a need for tiny only cross-worker communication!!

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

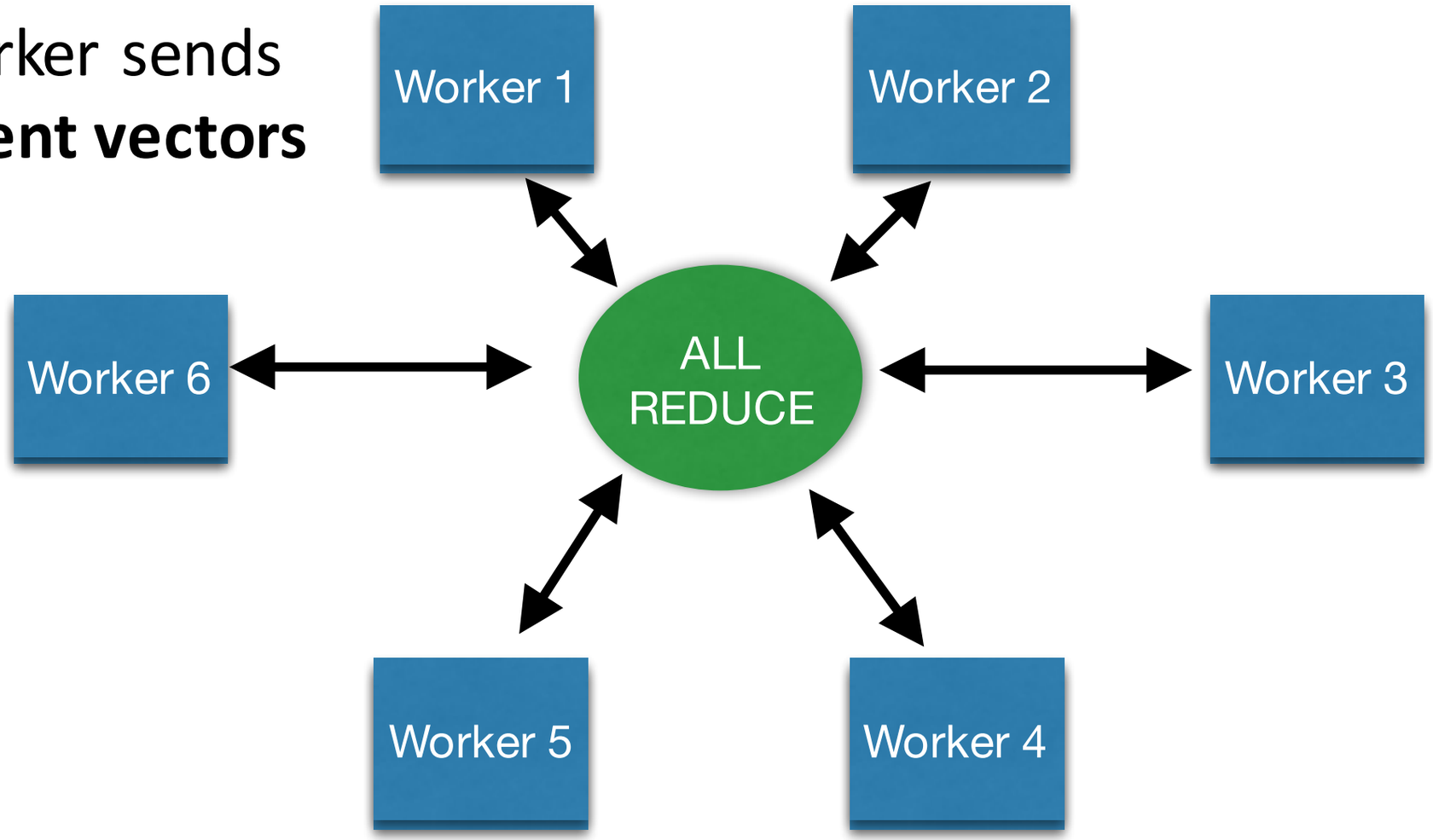
Distributed SGD



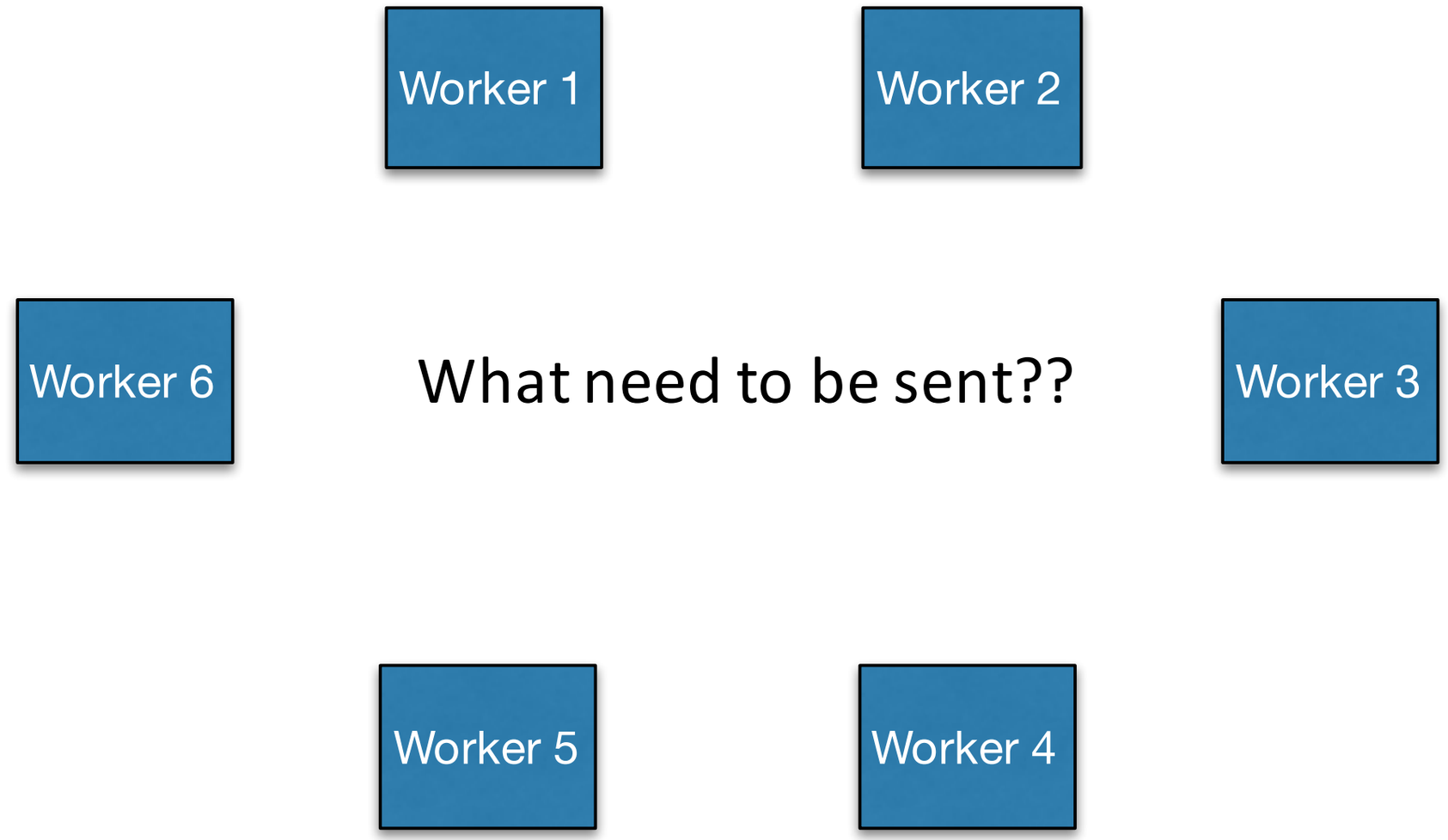
Used in Asynchronous RL!

Distributed SGD

Each worker sends
big gradient vectors



Distributed Evolution



Distributed Evolution

Algorithm 1 Evolution Strategies

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
6: **end for**

Worker 5

Worker 4

Distributed Evolution

Worker 1

Worker 2

Worker 6

θ and R(τ)?

Worker 3

θ is big!

but $\theta = \mu + \sigma\epsilon$

Same for all workers

Only need seed of random number generator!

Distributed Evolution

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

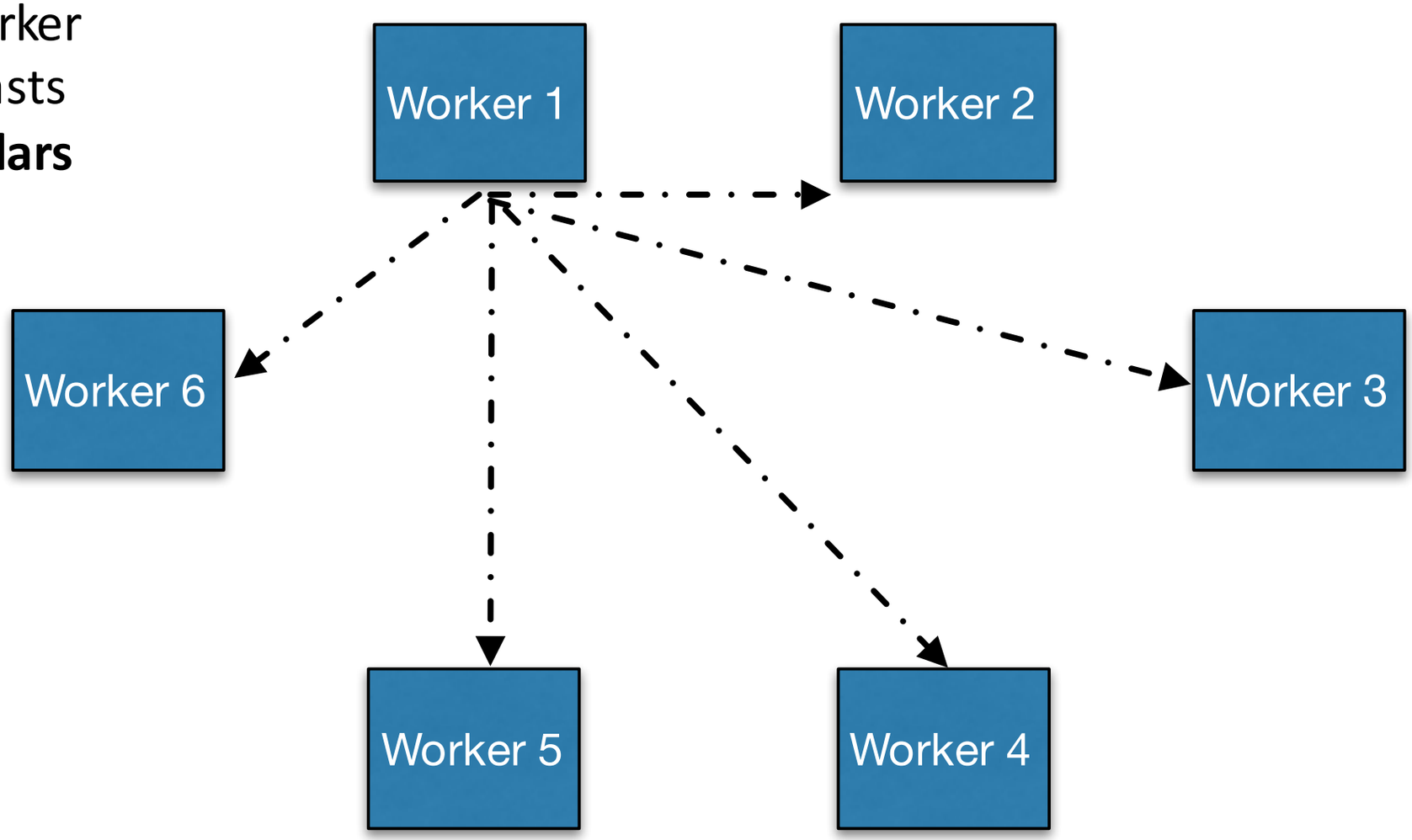
Algorithm 2 Parallelized Evolution Strategies

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]

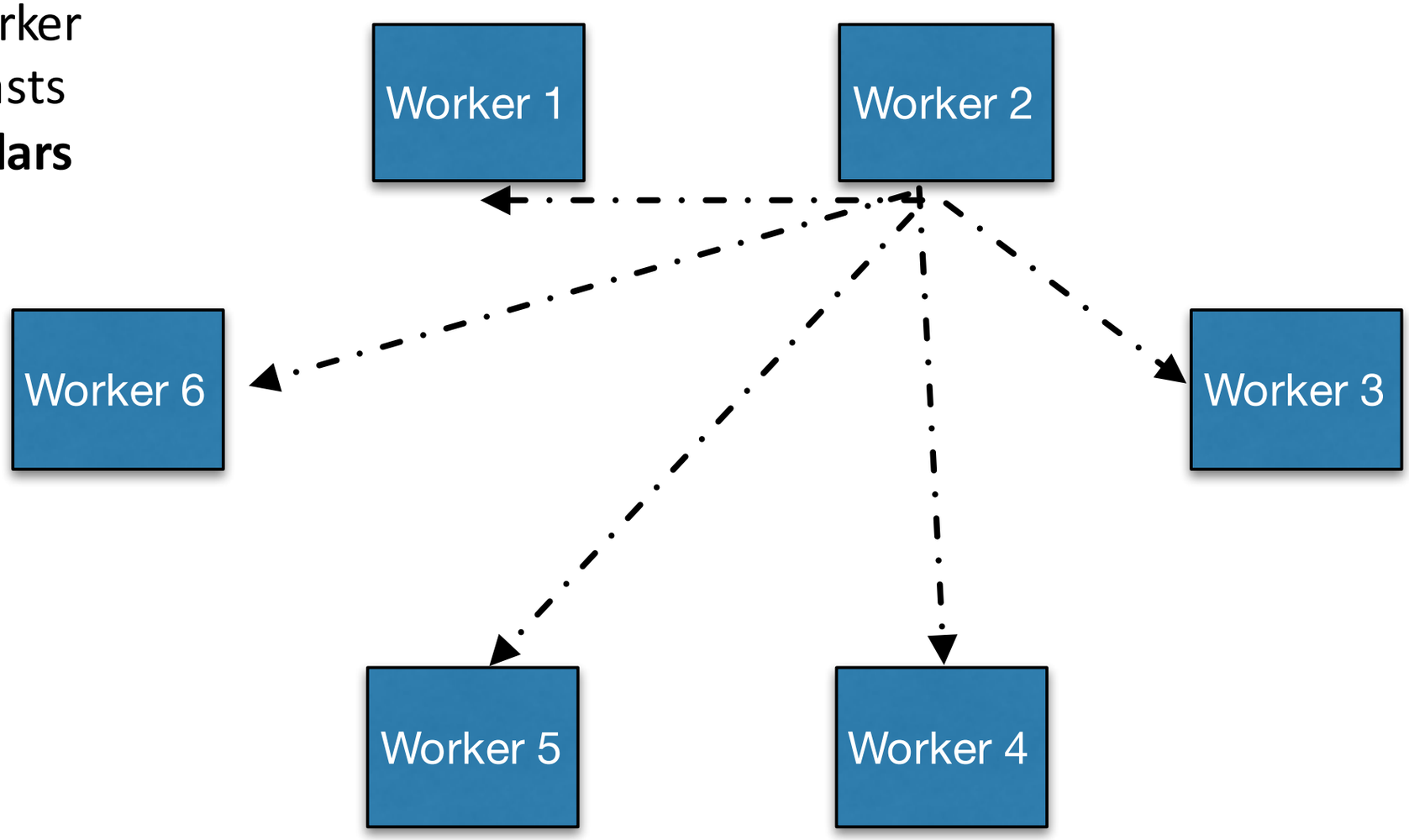
Distributed Evolution

Each worker broadcasts **tiny scalars**



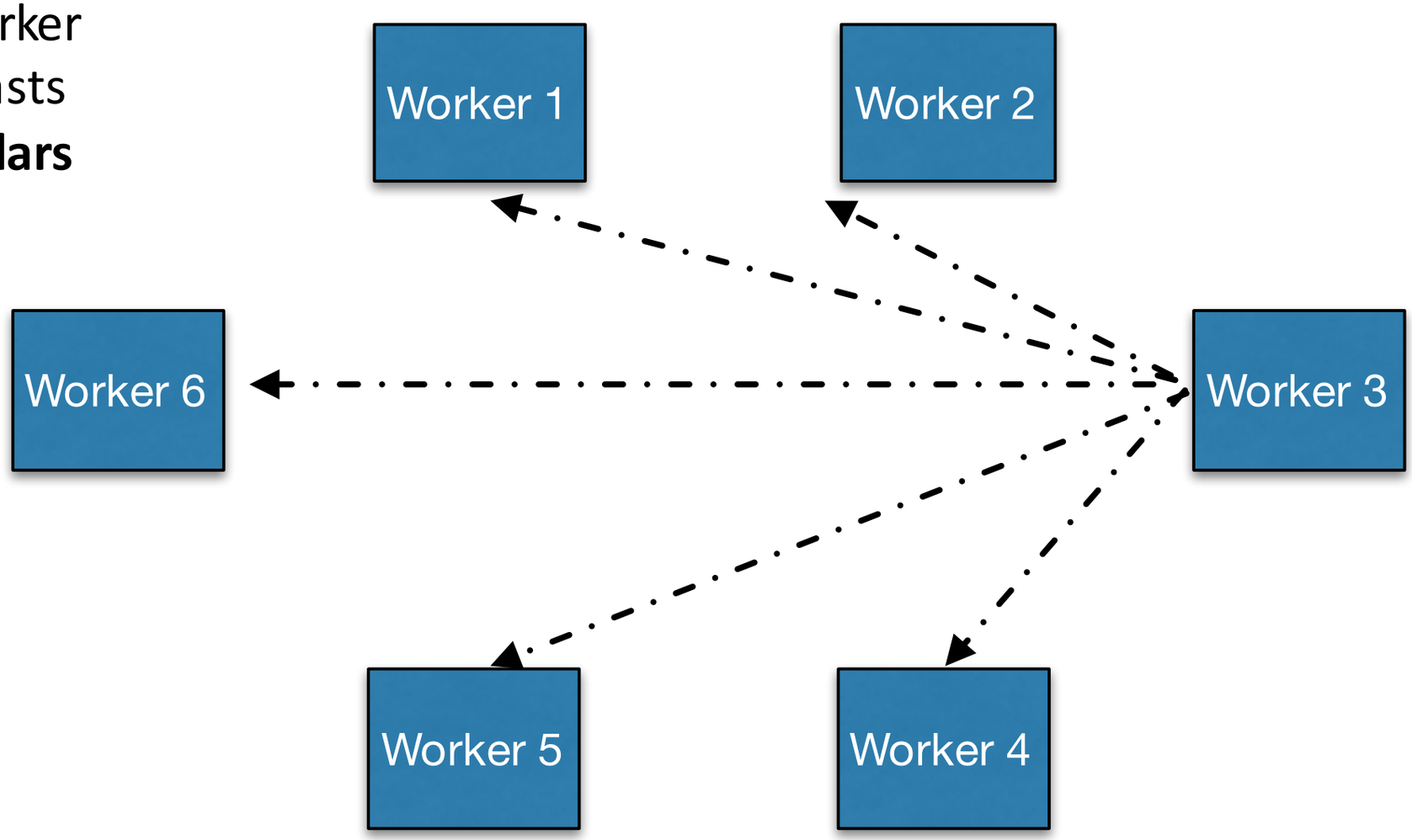
Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution Scales Very Well :-)

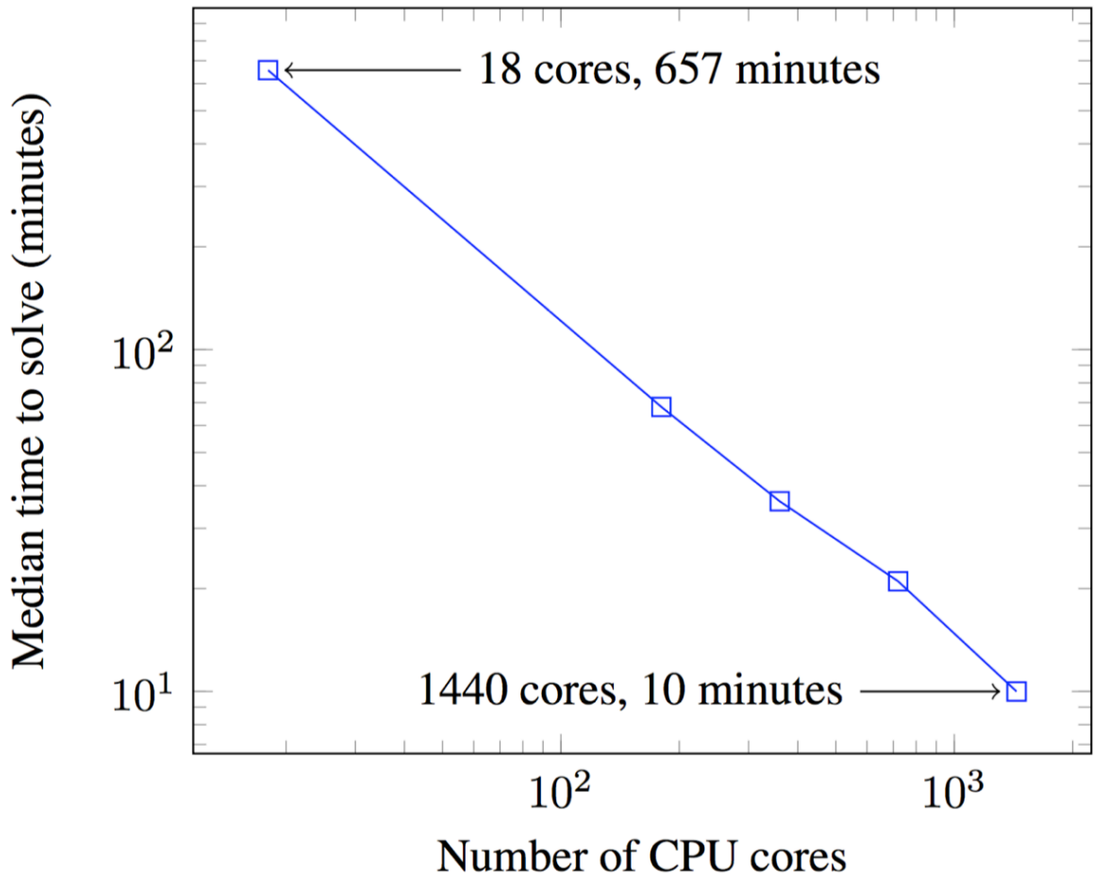
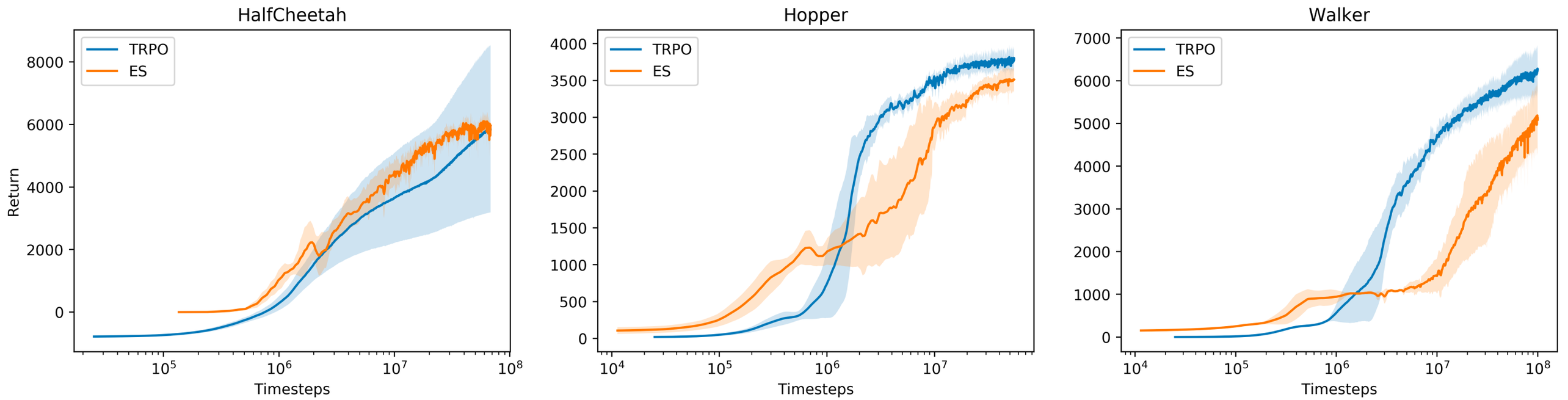


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

[Salimans, Ho, Chen, Sutskever, 2017]

Distributed Evolution Requires More Samples :-(



[Salimans, Ho, Chen, Sutskever, 2017]

Alternative derivations

Monte Carlo Policy Gradients (REINFORCE), gradient direction:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

How would we implement this in TF or Pytorch

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

Equivalently:

$$L^{IS}_{\theta_{old}}(\theta) = \mathbb{E}_{\tau \sim \theta_{old}} \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \hat{A}_t \right]$$

Derivation of likelihood ratio derivative from importance sampling

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\tau \sim P_\theta(\tau)}\left[R(\tau)\right] \\
&= \sum_{\tau} P(\tau \mid \theta) R(\tau) \\
&= \sum_{\tau} P(\tau \mid \theta_{old}) \frac{P(\tau \mid \theta)}{P(\tau \mid \theta_{old})} R(\tau) \\
&= \sum_{\tau \sim P(\tau \mid \theta_{old})} \frac{P(\tau \mid \theta)}{P(\tau \mid \theta_{old})} R(\tau) \\
&= \mathbb{E}_{\tau \sim \theta_{old}} \frac{P(\tau \mid \theta)}{P(\tau \mid \theta_{old})} R(\tau)
\end{aligned}
$$

$$
\mathbb{E}_{\tau \sim \theta_{old}} \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \hat{A}_t
$$

$$
\nabla J(\theta)\big|_{\theta_{old}} = \mathbb{E}_{\tau \sim \theta_{old}} \frac{\nabla_\theta P(\tau \mid \theta)\big|_{\theta_{old}}}{P(\tau \mid \theta_{old})} R(\tau)
$$

$$
= \mathbb{E}_{\tau \sim \theta_{old}} \nabla_\theta \log P(\tau \mid \theta)\big|_{\theta_{old}} R(\tau)
$$

Alternative derivations

Monte Carlo Policy Gradients (REINFORCE), gradient direction:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

How would we implement this in TF or Pytorch

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

Equivalently:

$$L^{IS}_{\theta_{old}}(\theta) = \mathbb{E}_{\tau \sim \theta_{old}} \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \hat{A}_t \right]$$

Next: Instead of leaving the stepwise to some schedule of the learning rate (heuristically chosen), make it part of the algorithm!

Hard to choose stepsizes

- Policy gradients

$$\hat{g} = \hat{\mathbb{E}}_t \Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \Big]$$

- Can differentiate the following loss

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \Big[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \Big].$$

- Input data is nonstationary due to changing policy: observation and reward distributions change
- Bad step is more damaging than in supervised learning, since it affects visitation distribution

- Step too big
  Bad policy->data collected under bad policy-> we cannot recover
  (in SL, data does not depend on neural network weights)
- Step too small
  Not efficient use of experience
  (in SL, data can be trivially re-used)

SGD:

$$\theta_{new} = \theta_{old} + \epsilon \cdot \hat{g}$$

Natural Gradient Descent

$$\frac{-\nabla_\theta \mathcal{L}(\theta)}{\|\nabla_\theta \mathcal{L}(\theta)\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \underset{d \text{ s.t. } \|d\| \leq \epsilon}{\arg\min} \mathcal{L}(\theta + d)$$

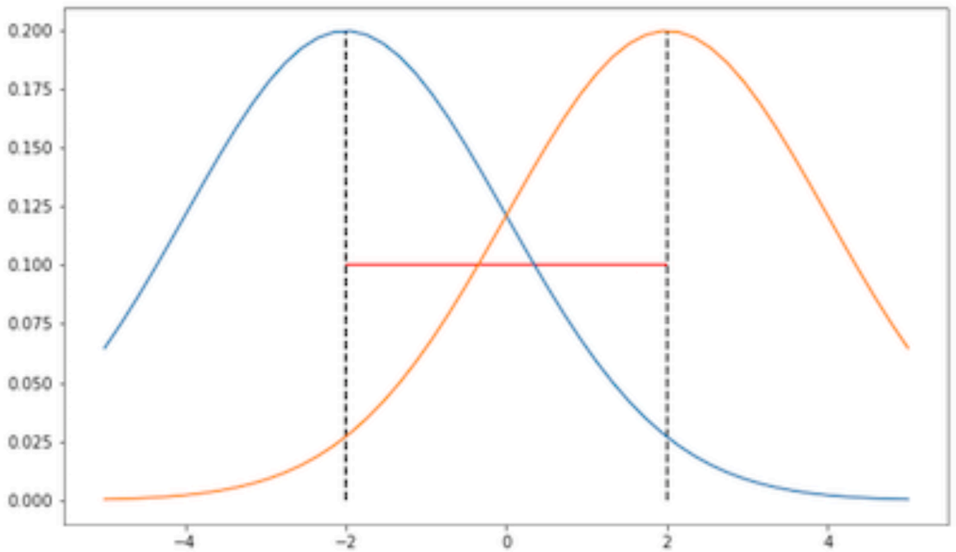
Take a step in direction d where d stays within a a ball of radius epsilon.

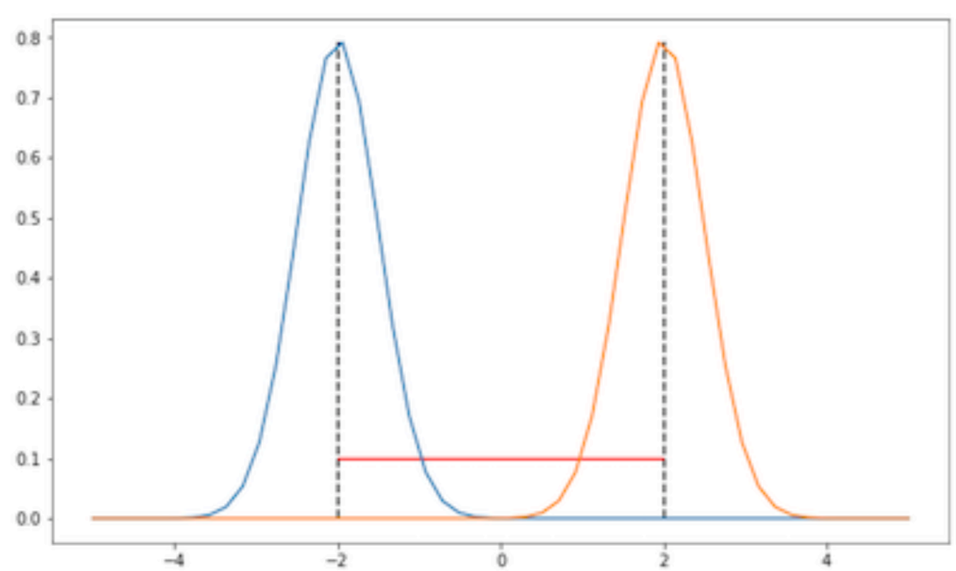
Gradient Descent in Parameter Space

Take a step in direction d where d stays within a a ball of radius epsilon.

$$\frac{-\nabla_\theta \mathcal{L}(\theta)}{\|\nabla_\theta \mathcal{L}(\theta)\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg\min_{d \text{ s.t. } \|d\| \leq \epsilon} \mathcal{L}(\theta + d)$$

Euclidean distance in parameter space





Imagine we are trying to minimize -loglik, and want to update the parameters of the distribution.

Same distance in parameter space results in very different distances in distribution space

Ideally, **we want to take steps based on distance in distribution space. Way more robust, less tweaking of the step.**

What metric shall we use?

Gradient Descent in Distribution Space

We will use the Kullback-Leibler divergence (KL) to measure distances between distributions before and after the update.

$$\frac{-\nabla_\theta \mathcal{L}(\theta)}{\|\nabla_\theta \mathcal{L}(\theta)\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg\min_{d \text{ s.t. } \|d\| \leq \epsilon} \mathcal{L}(\theta + d)$$

Instead:

$$d^* = \arg\min_{d \text{ s.t. } \mathrm{KL}[p_\theta \| p_{\theta+d}] = c} \mathcal{L}(\theta + d)$$

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right)$$

$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Gradient Descent in Distribution Space

We will use the Kullback-Leibler divergence (KL) to measure distances between distributions before and after the update.

$$\frac{-\nabla_\theta \mathcal{L}(\theta)}{\|\nabla_\theta \mathcal{L}(\theta)\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \underset{d \text{ s.t. } \|d\| \leq \epsilon}{\arg\min} \mathcal{L}(\theta + d)$$

Instead:

$$d^* = \underset{d \text{ s.t. } \mathrm{KL}[p_\theta \| p_{\theta+d}] = c}{\arg\min} \mathcal{L}(\theta + d)$$

Unconstrained penalized objective:

$$d^* = \underset{d}{\arg\min}\, \mathcal{L}(\theta + d) + \lambda \left(\mathrm{KL}[p_\theta \| p_{\theta+d}] - c\right)$$

First order Taylor expansion for the loss and second order for the KL:

$$\approx \underset{d}{\arg\min}\, \mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}} d + \frac{1}{2} \lambda\, d^{\mathrm{T}} \mathrm{F} d - \lambda c$$

Gradient Descent in Distribution Space

We will use the Kullback-Leibler divergence (KL) to measure distances between distributions before and after the update.

$$\frac{-\nabla_\theta \mathcal{L}(\theta)}{\|\nabla_\theta \mathcal{L}(\theta)\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \underset{d \text{ s.t. } \|d\| \leq \epsilon}{\arg\min} \mathcal{L}(\theta + d)$$

Instead:

$$d^* = \underset{d \text{ s.t. } \mathrm{KL}[p_\theta \| p_{\theta+d}] = c}{\arg\min} \mathcal{L}(\theta + d)$$

Unconstrained penalized objective:

$$d^* = \underset{d}{\arg\min}\, \mathcal{L}(\theta + d) + \lambda \left(\mathrm{KL}[p_\theta \| p_{\theta+d}] - c\right)$$

First order Taylor expansion for the loss and second order for the KL:

$$\approx \underset{d}{\arg\min}\, \mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}} d + \frac{1}{2} \lambda\, d^{\mathrm{T}} \mathrm{F} d - \lambda c$$

Natural Gradient Descent

Newton's method

$$
\begin{aligned}
0 &= \frac{\partial}{\partial d}\mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^{\mathrm{T}} d + \frac{1}{2}\lambda\, d^{\mathrm{T}}\mathrm{F} d - \lambda c \\
&= \nabla_\theta \mathcal{L}(\theta) + \lambda\, \mathrm{F} d \\
\lambda\, \mathrm{F} d &= -\nabla_\theta \mathcal{L}(\theta) \\
d &= -\frac{1}{\lambda}\mathrm{F}^{-1}\nabla_\theta \mathcal{L}(\theta)
\end{aligned}
$$

The natural gradient:

$$
\tilde{\nabla}_\theta \mathcal{L}(\theta) = \mathrm{F}^{-1}\nabla_\theta \mathcal{L}(\theta)
$$