Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

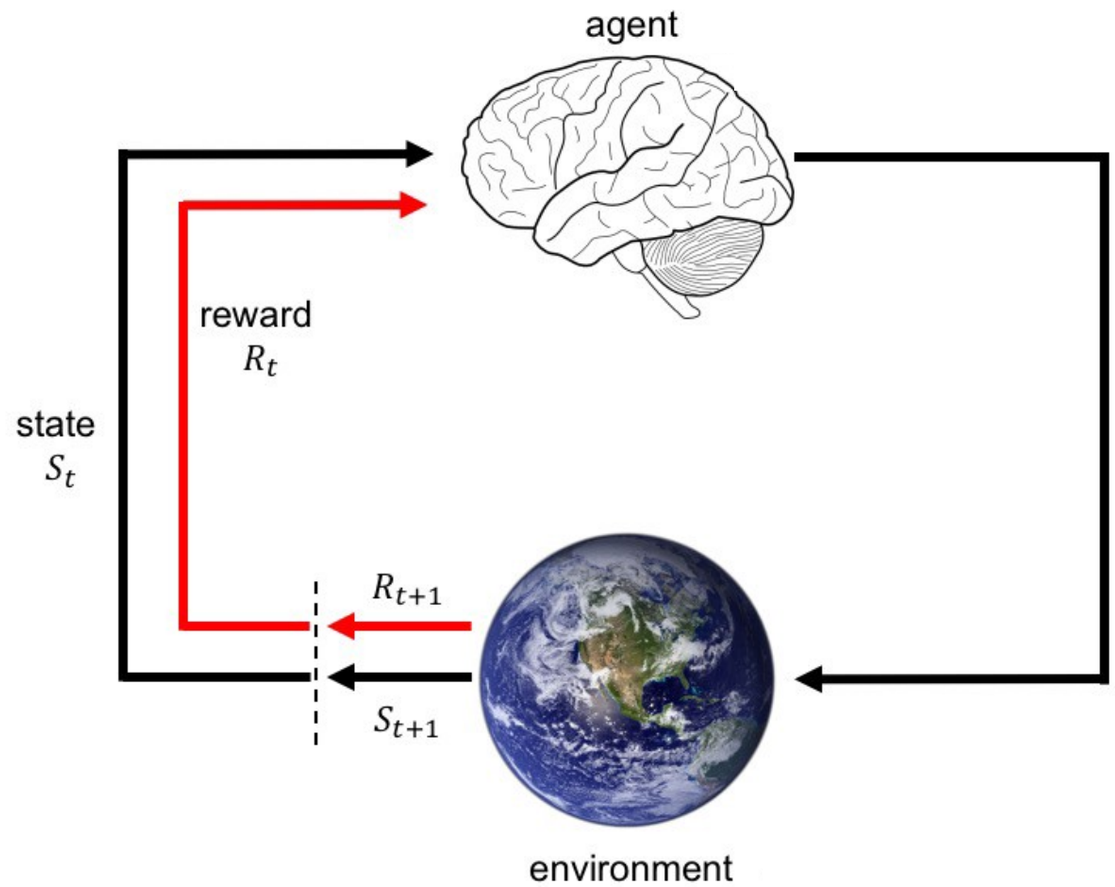
Evolutionary Methods

Spring 2020, CMU 10-403

Katerina Fragkiadaki

Reinforcement Learning

Learning behaviours from rewards while interacting with the environment



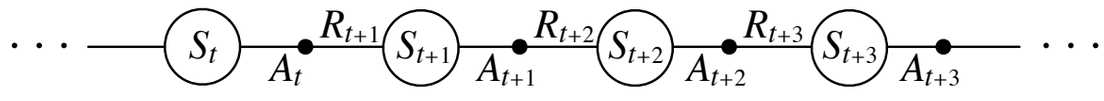
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

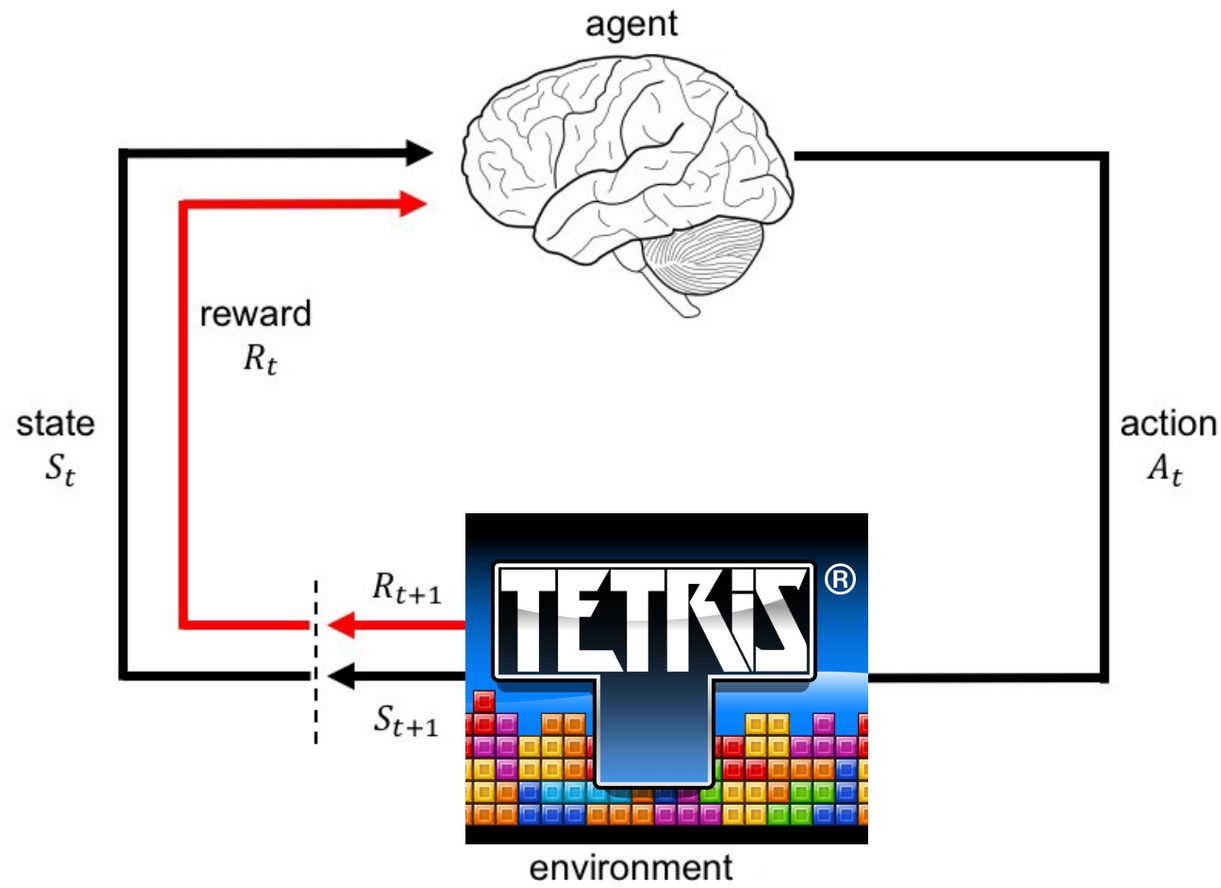
gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$



A concrete example: Playing Tetris

Learning behaviours from rewards while interacting with the virtual environment



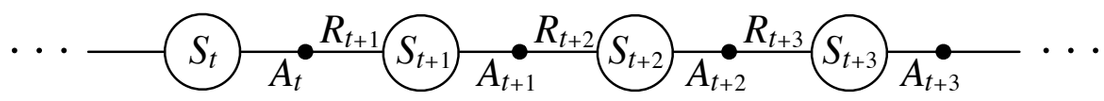
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

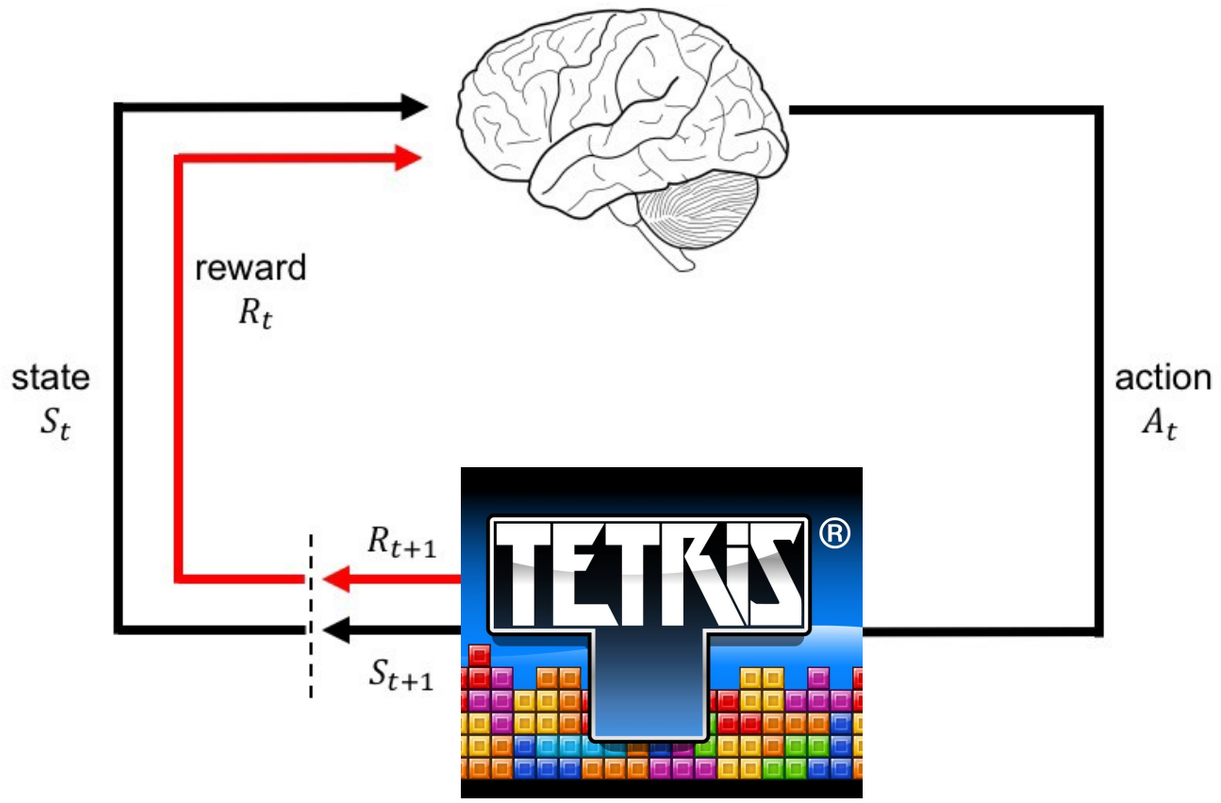
gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$



Policy search for Playing Tetris

- states: the board configuration and the falling piece (lots of states ~ 2^200)
- actions: translations and rotations of the piece
- rewards: scores we collect by cancelling rows, big negative reward when we loose.
- Q: can we think of a different reward structure?

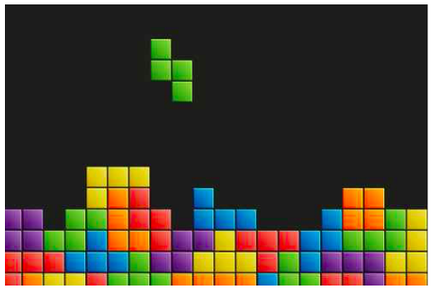


Our goal is to learn a policy that maximizes the score of the game in expectation.

- Q: what does ``in expectation" mean?
- A: both the agent's policy and the environment can be stochastic. We thus need to consider our average performance across environments and actions selected.

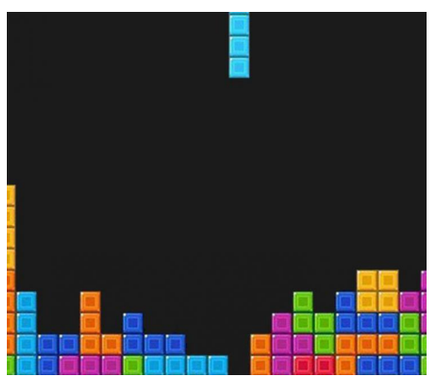
Policy search for Playing Tetris

Policy: a mapping from states to actions



Go right

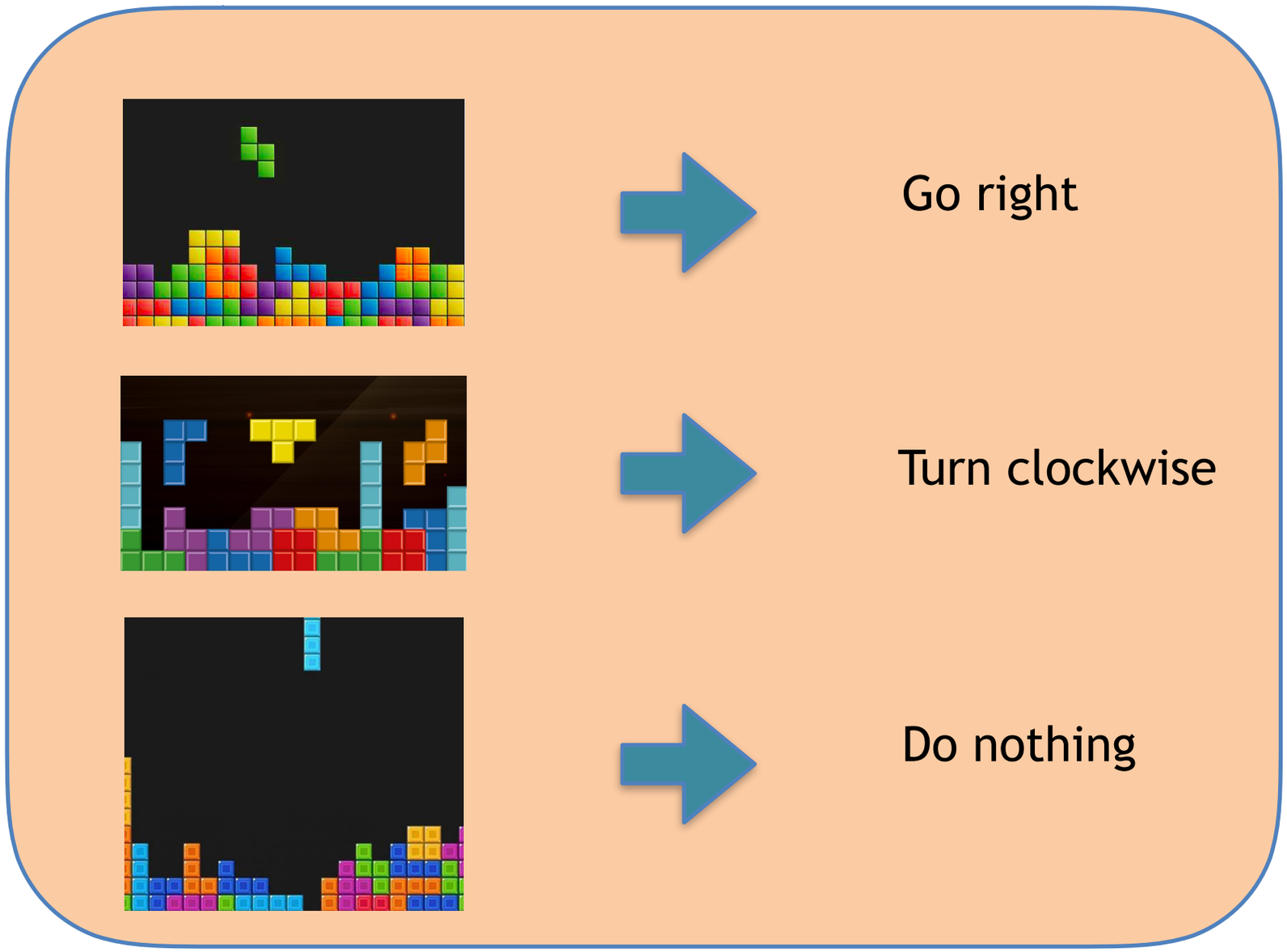
Turn clockwise



Do nothing

Policy search for Playing Tetris: tabular policy

Policy: a tabular mapping from states to actions



If the number of state space was small, we could have an exhaustive enumeration of states paired with the optimal action(s) to take provided by a Master player.

Policy search for Playing Tetris

Imagine we simply ask an expert (Master) player what to do at each state we encounter, and learn using supervised learning. We collect data for one full mon of him playing 24/7.

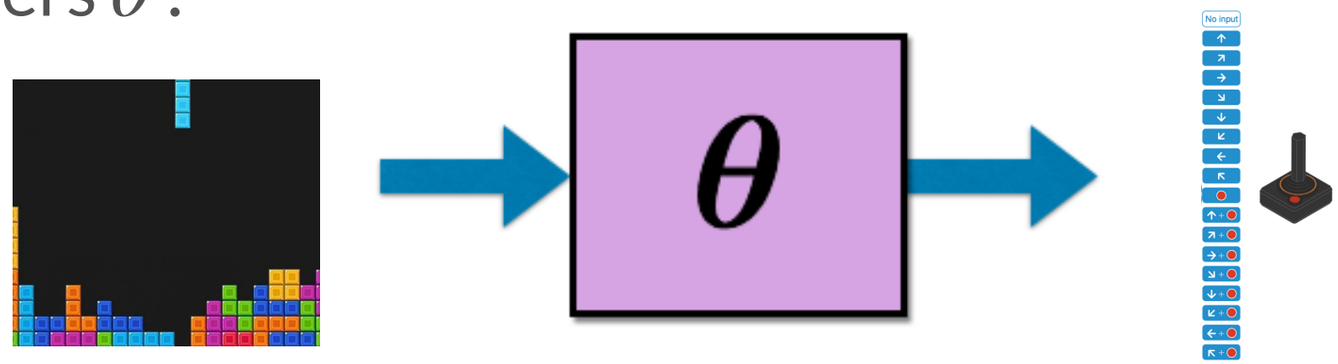
Q1: During the training period, could the agent see all states in Tertis, to figure out what is the corresponding best action and create the table?

A1: No, there will always be states at test time that we have not visited at training time, i.e., and we will not know what to do.

Q2: any solutions?

Policy search for Playing Tetris: function approximation

Policy: a functional mapping from states to actions, parametrized by parameters θ .



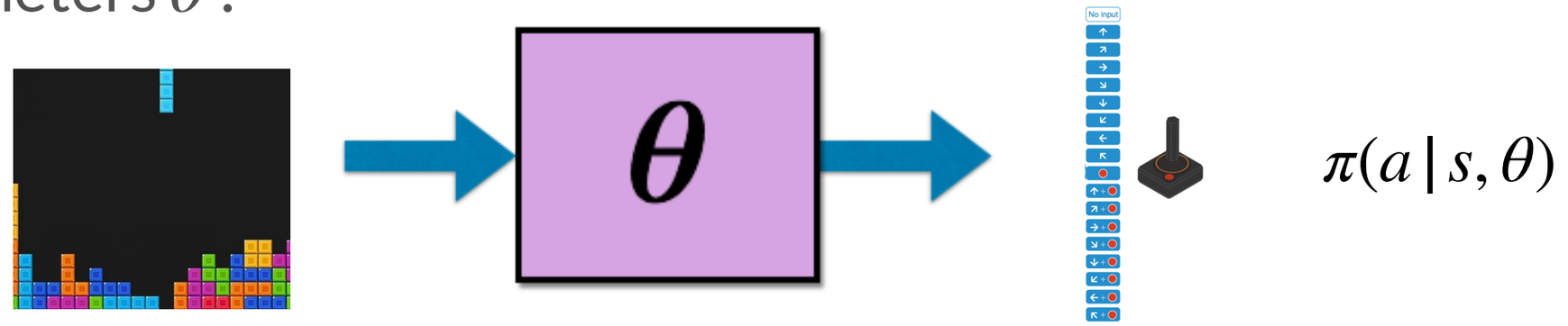
$$\pi(a \mid s, \theta)$$

In principal, we can represent actions to take for all states.

- Q1: Who is larger: the number of parameters or the number of states?
- Q2: Do we know how to act now on states that we didn't see during training?
- Q3: For states-action pairs that we have seen during training, will we get them right?
- Q4: What are the properties that our function should have to generalize well from seen to unseen states?

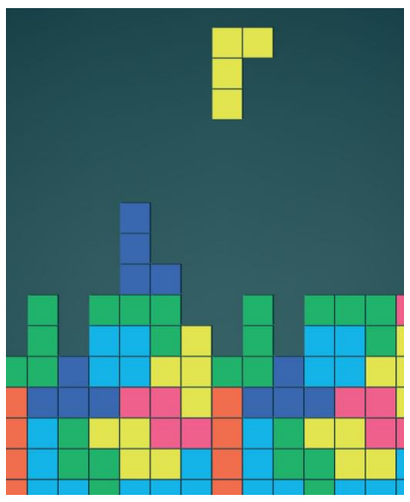
Policy search for Playing Tetris: function approximation

- Policy: a functional mapping from states to actions, parametrized by parameters θ .



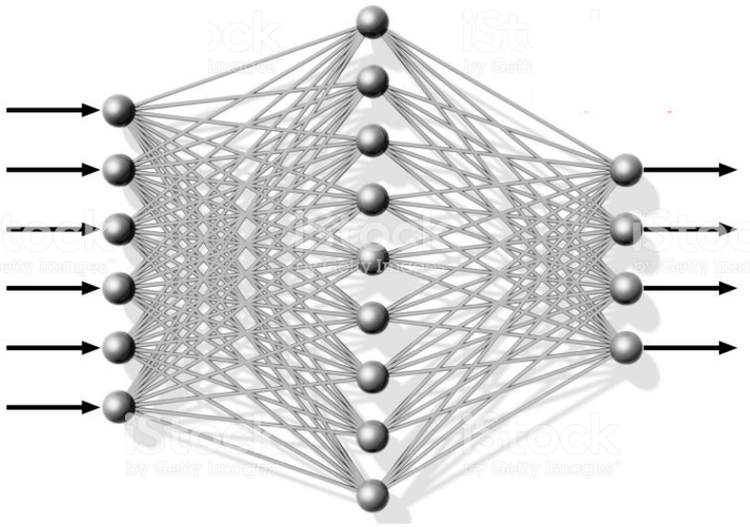
$$\pi(a \mid s, \theta)$$

- Our function should learn features that make two distinct states (in pixel space) to be close in feature space when they share the same optimal action. E.g., in our case, the color of the blocks is irrelevant, as well as whether a configuration takes place to the right or to the left of the screen.



Who will provide the features?

Human engineered features

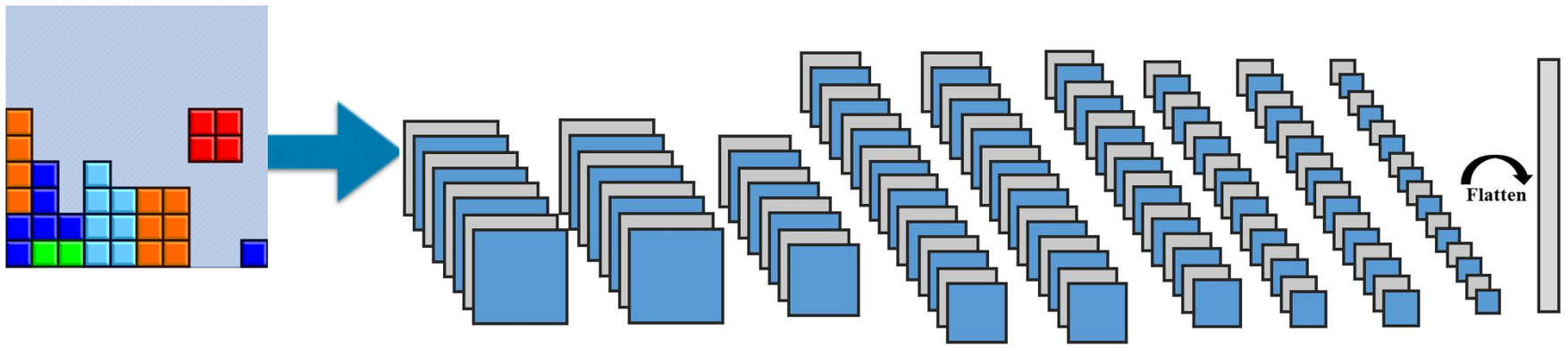


$$\pi(a \mid s, \theta)$$

Two choices:

1. The engineer will manually define a set of features to capture the state (board configuration). Then the model will learn to map those hand-designed features to a distribution over actions, e.g., using a linear model or shallow network as its functional form, and imitation or reinforcement learning as its learning objective.

Who will provide the features?

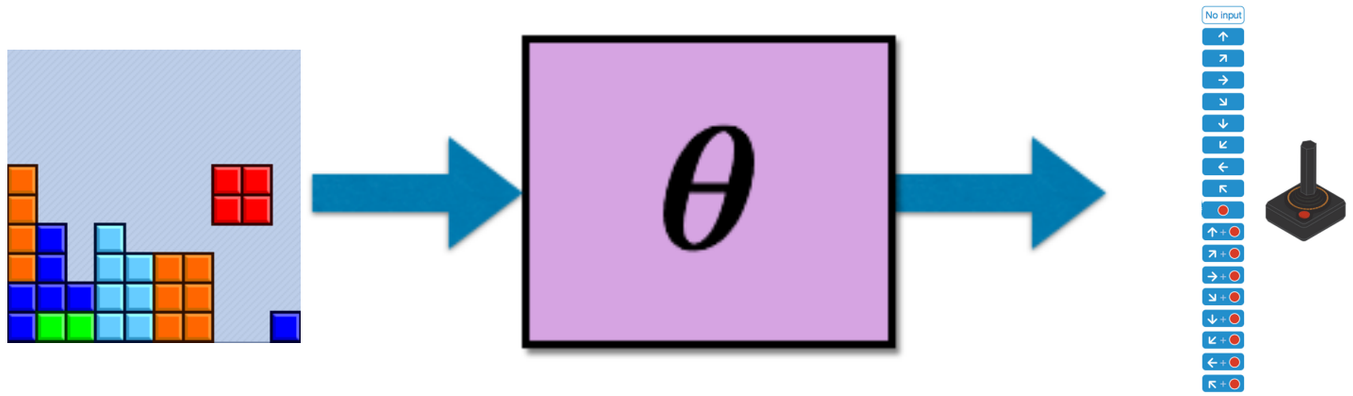


$$\pi(a \mid s, \theta)$$

Two choices:

2. The model will learn the features to capture the state (board configuration) as the weight kernels of the different layers of the deep neural network by mapping feature activations to a distribution over actions and optimizing imitation or reinforcement learning objectives. Feature discovery and classifier learning are not separated.

Learning the parameters for our policy function



$$\pi(a \mid s, \theta)$$

$$\max_{\theta} . \; U(\theta) = \max_{\theta} . \; \mathbb{E}\left[R(\tau) \mid \pi_\theta, \mu_0(s_0)\right]$$

τ : trajectory, a sequence of state and actions, a game fragment or a full game

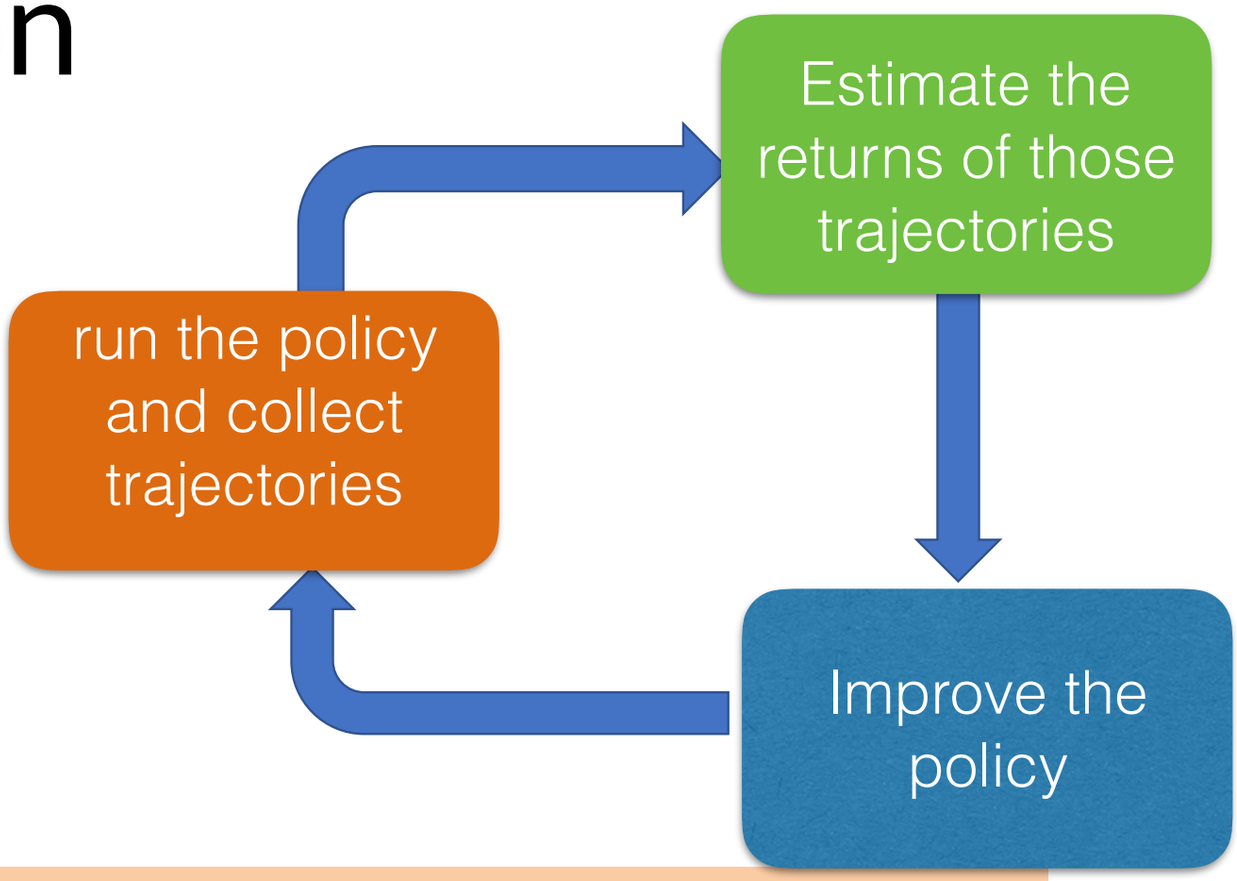
$R(\tau)$:reward of a trajectory: (discounted) sum of the rewards of the indivisual state/actions

$\mu_0(s_0)$: the initial state distribution

Q: Any ideas on how to solve this optimization?

Black-box optimization

- No gradient information, no information regarding the structure of the reward, that it is additive over states, that states are interconnected in a particular way, and so on.



Initialize the parameters θ randomly.

1.Perturb policy parameters,

2.collect trajectories and evaluate their rewards,

3.Promote the policy parameters that resulted in trajectories that gave the largest improvement,

4.GOTO 1.

Q:Could we use Gaussian processes to capture a posterior over parameter vectors?

Evolutionary methods

$$\max_{\theta} . \quad U(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[R(\tau) \,|\, \pi_\theta, \mu_0(s_0) \right]$$

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*

Simple and biologically plausible…

Cross-entropy method

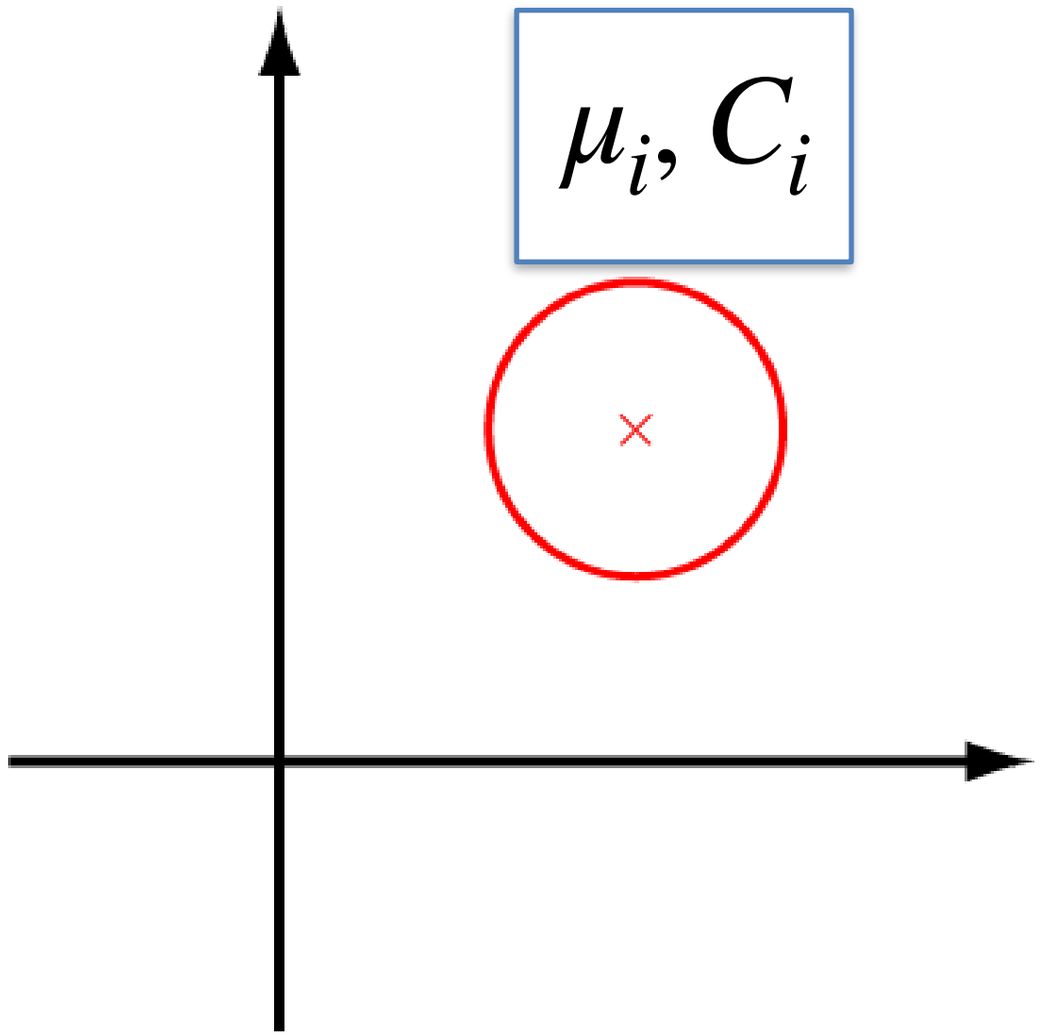
Parameters to be sampled from a multivariate Gaussian with a diagonal covariance matrix. We will evolve this Gaussian distribution (update the mean and variances pf the parameter elements towards parameter samples that have highest fitness scores.

Input: parameter space Θ, number of parameter vectors n, proportion $\rho \leq 1$, noise η
Initialize: Set the parameter $\mu = \bar{0}$ and $\sigma^2 = 100I$ (I is the identity matrix)
for $k = 1, 2, \ldots$ **do**
Generate a random sample of n parameter vectors $\{\theta_i\}_{i=1}^{n} \sim \mathcal{N}(\mu, \sigma^2 I)$
For each θ_i, play L games and calculate the average number of rows removed (score) by the controller
Select $\lfloor \rho n \rfloor$ parameters with the highest score $\theta'_1, \ldots, \theta'_{\lfloor \rho n \rfloor}$
Update μ and σ: $\mu(j) = \frac{1}{\lfloor \rho n \rfloor} \sum_{i=1}^{\lfloor \rho n \rfloor} \theta'_i(j)$ and $\sigma^2(j) = \frac{1}{\lfloor \rho n \rfloor} \sum_{i=1}^{\lfloor \rho n \rfloor} [\theta'_i(j) - \mu(j)]^2 + \eta$

Works embarrassingly well in low-dimensions, e.g., to search for a linear policy over the 22 Bertsekas features for Tetris.

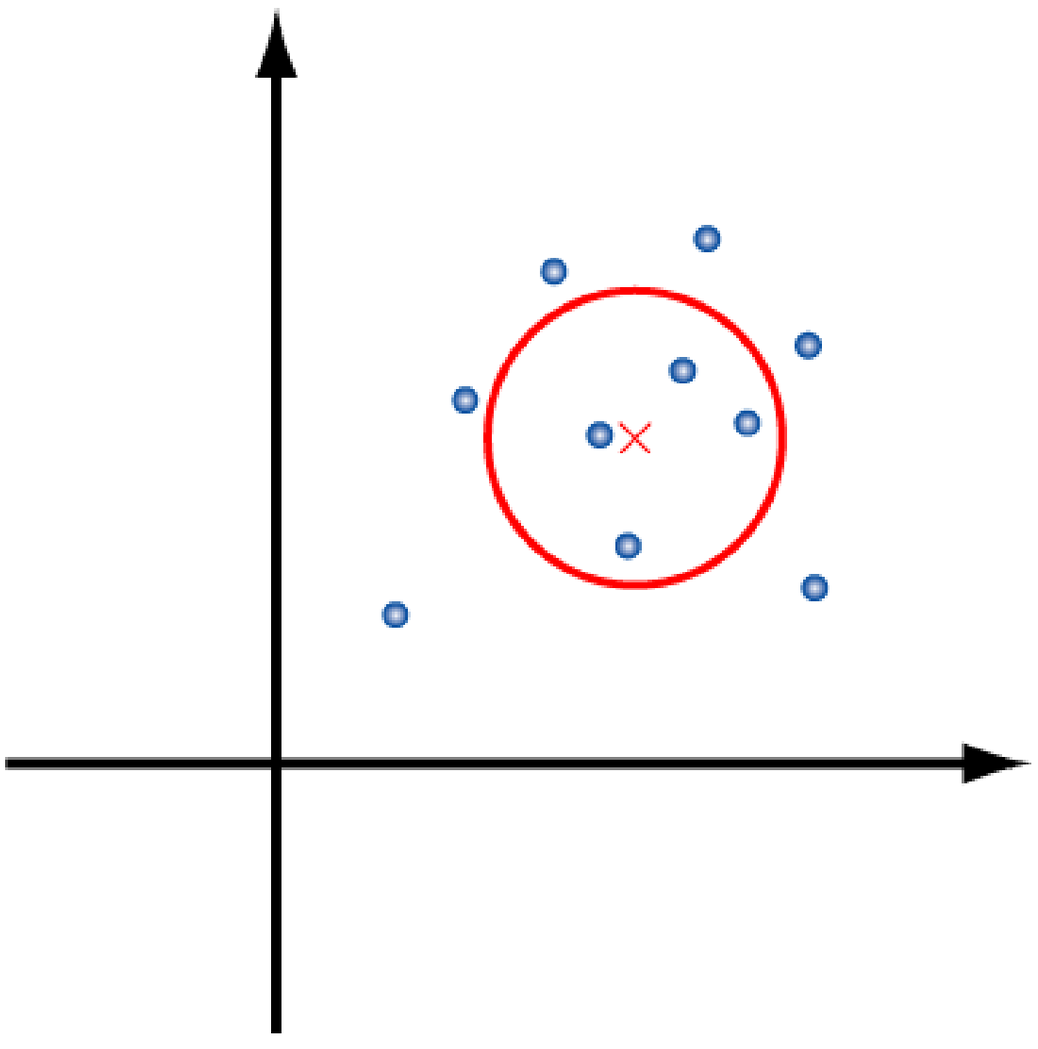
Covariance Matrix Adaptation

We can also consider a full covariance matrix



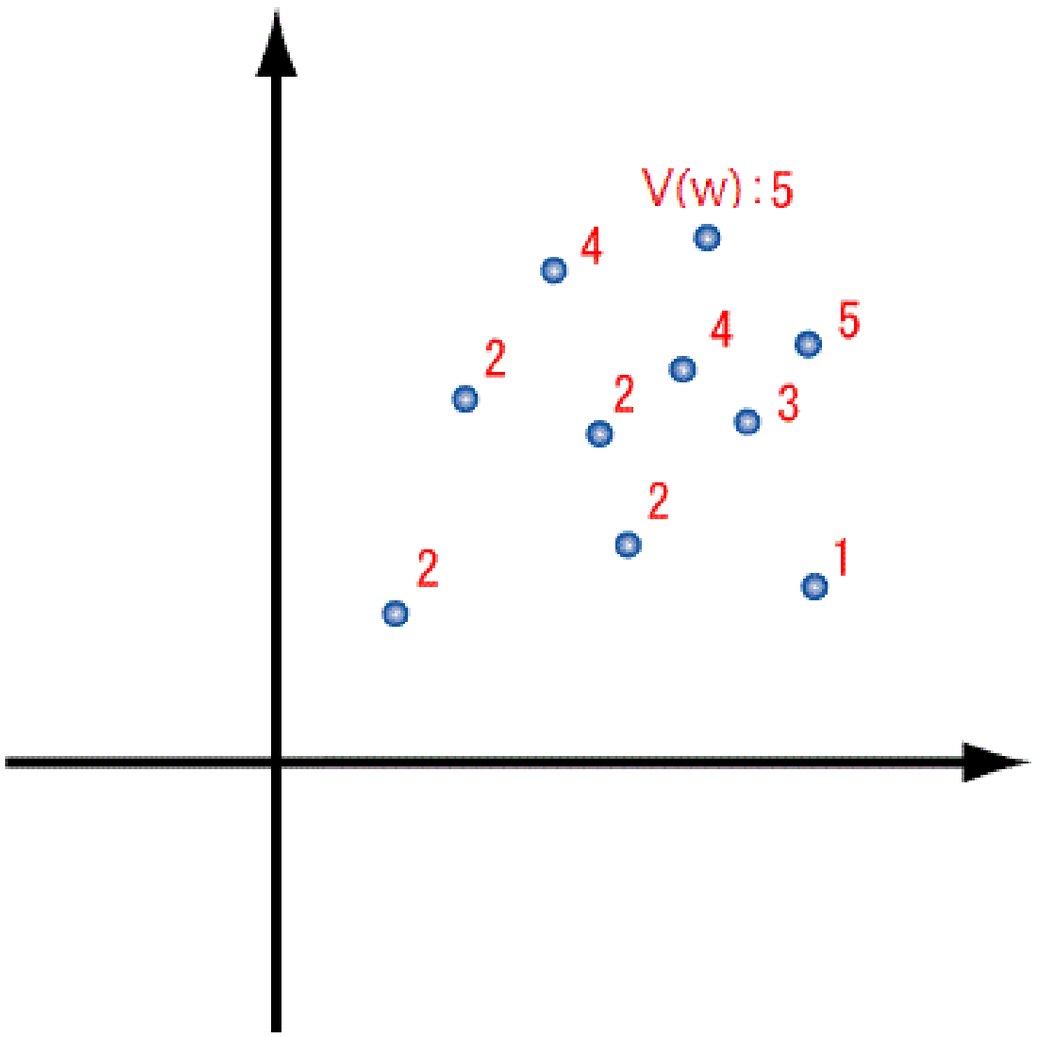
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



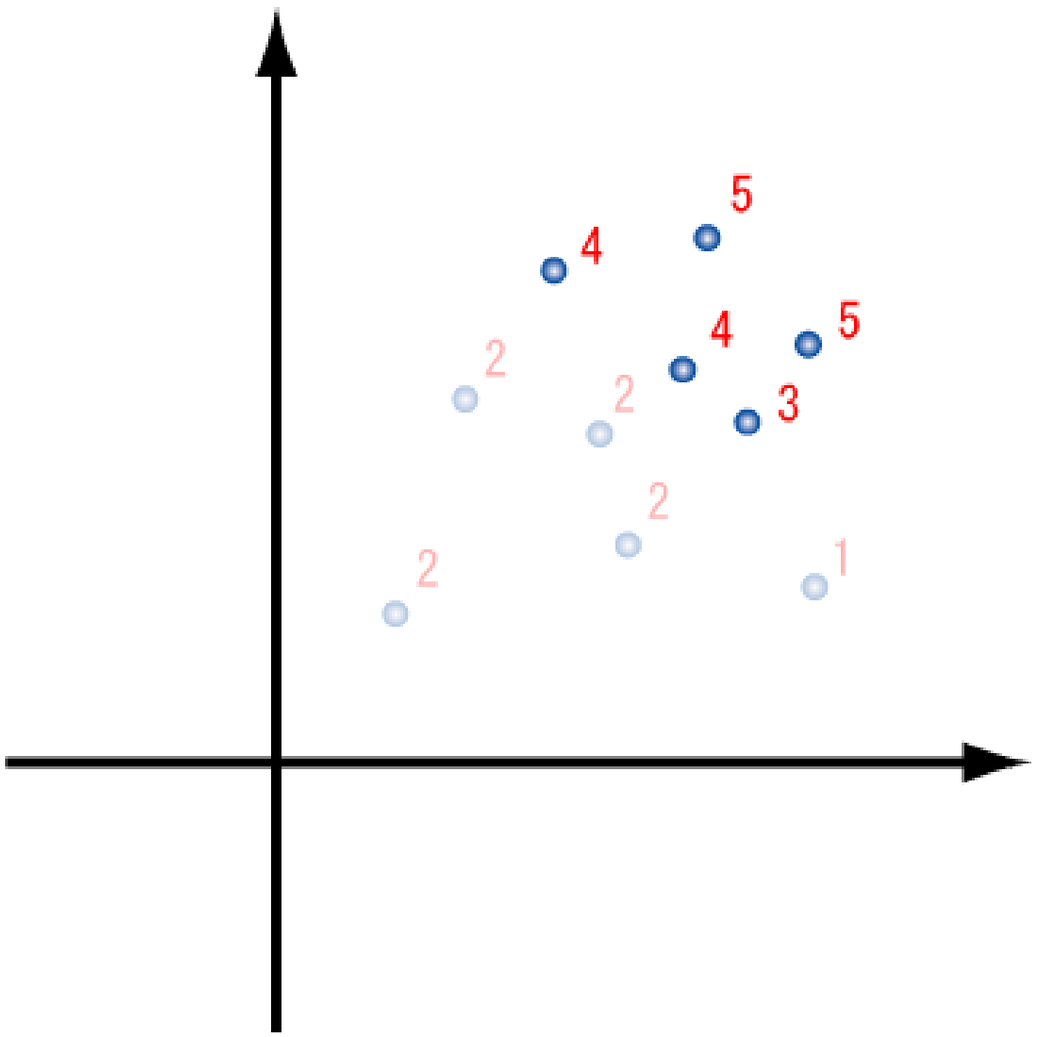
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



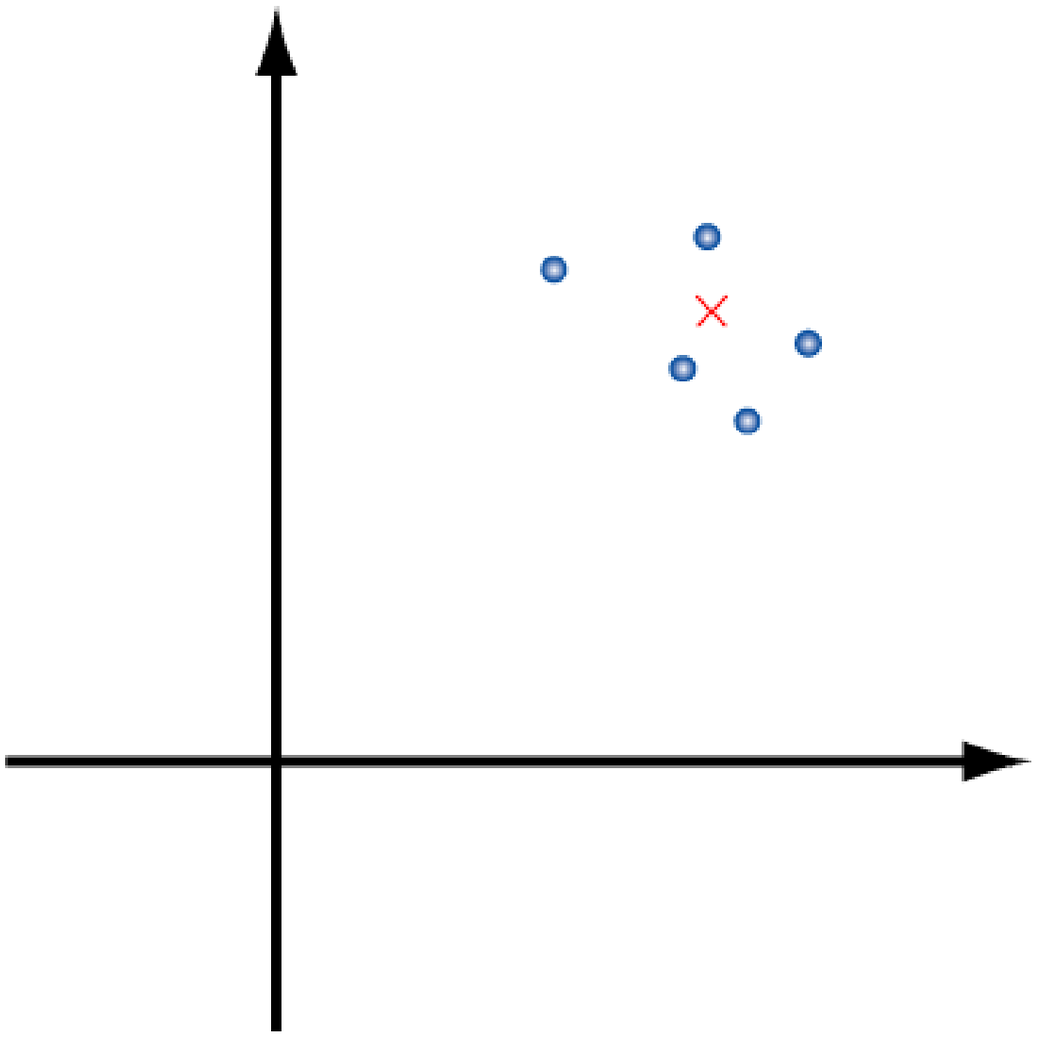
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



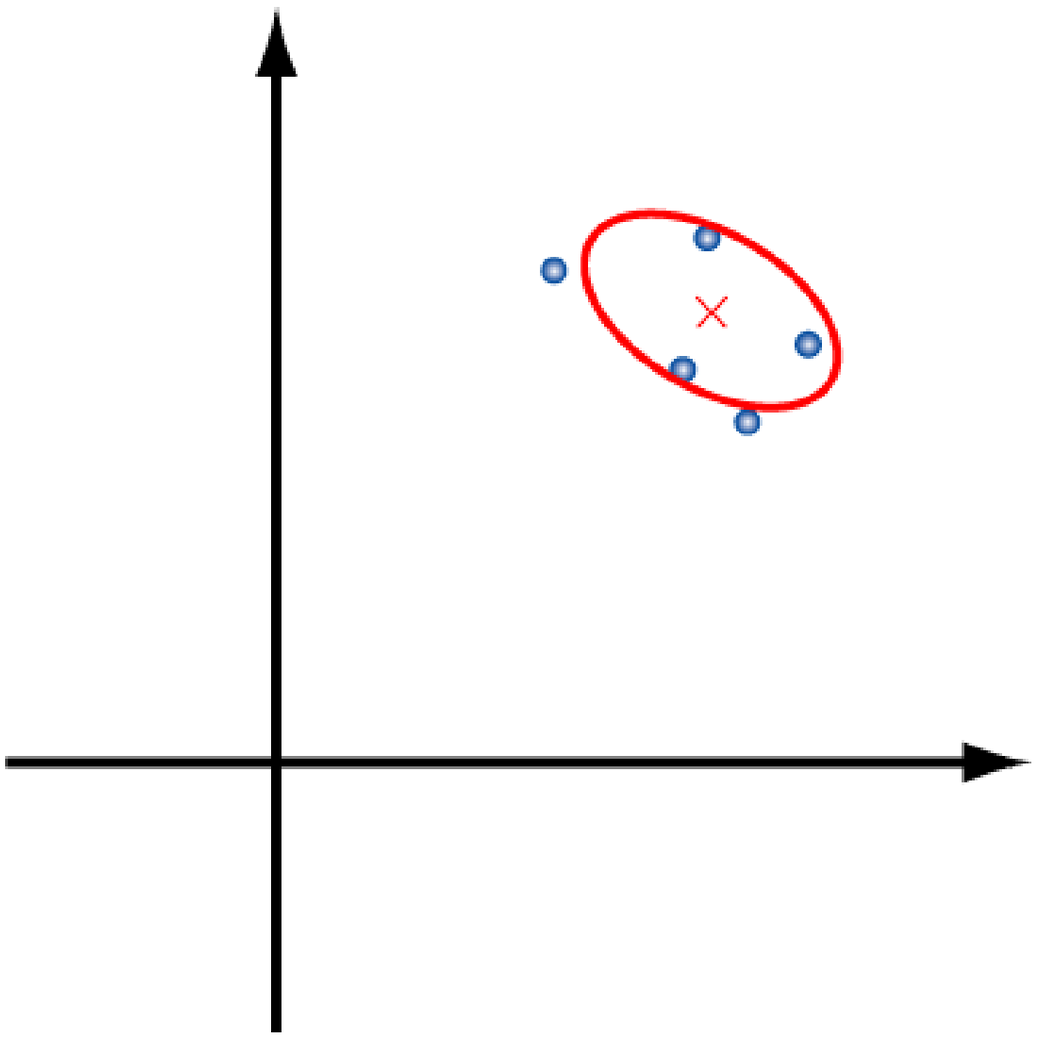
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



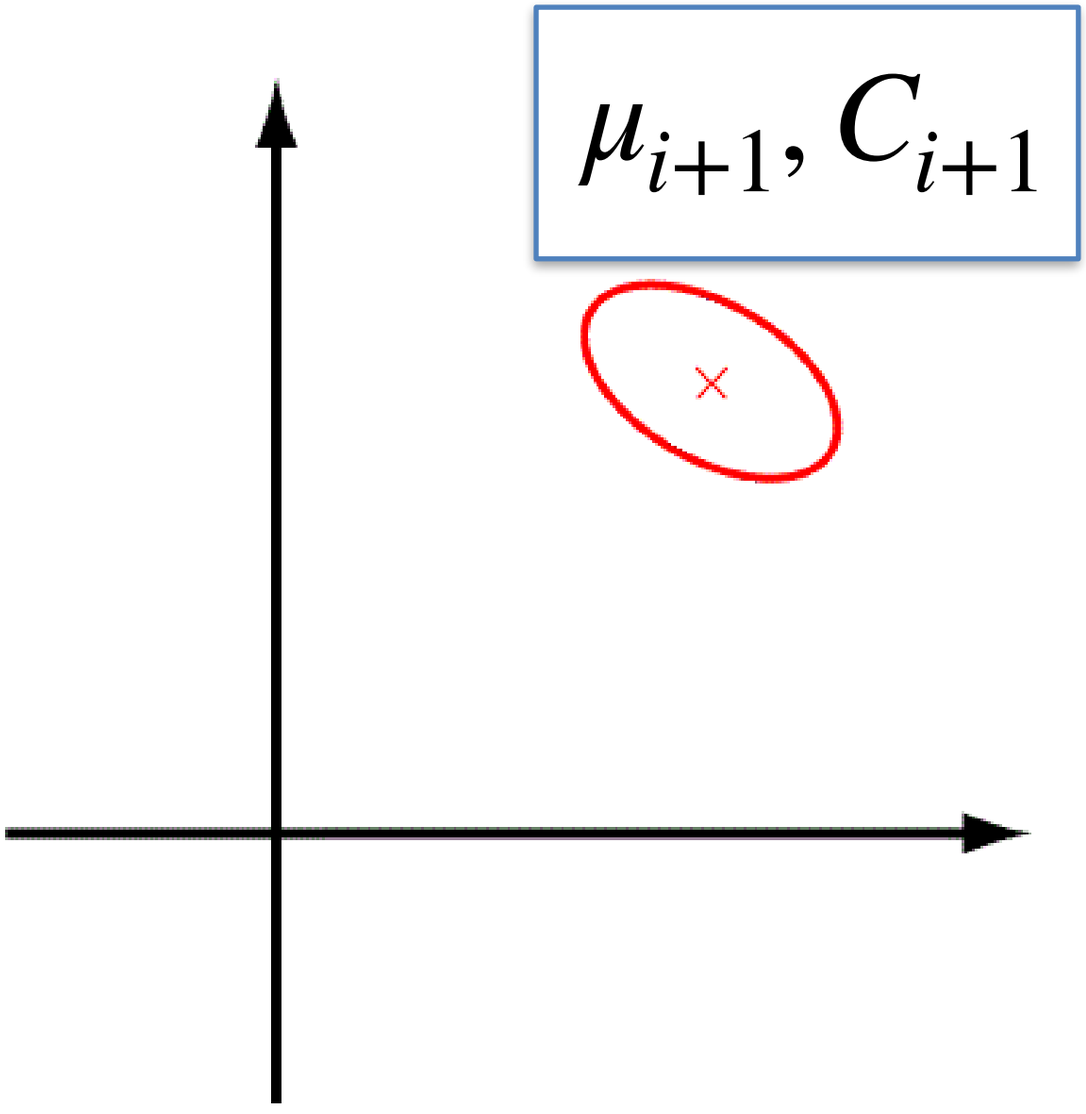
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



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

Covariance Matrix Adaptation



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

Natural Evolutionary Strategies (NES)

- So far, we have been evolving Gaussian parameter distributions, and we have been selecting the best parameter offsprings
- NES considers every offspring.
- NES has memory of the mean parameter so far and does not completely overwrite it at every iteration, rather it moves it a small step

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 $F_i = F(\mu_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\mu_{t+1} \leftarrow \mu_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Natural Evolutionary Strategies

- Consider a fixed variance Gaussian distribution of policy parameters: $\theta \sim P_\mu(\theta)$
- Goal: maximize $\mathbb{E}_{\theta \sim P_\mu} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$
- The variable we are optimizing is the mean $\mu \in \mathbb{R}^d$

Natural Evolutionary Strategies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
- Goal: maximize $\mathbb{E}_{\theta \sim P_\mu} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$
- The variable we are optimizing is the mean $\mu \in \mathbb{R}^d$

$$\nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\mathrm{F}(\theta) \right] = \nabla_\mu \int P_\mu(\theta) \mathrm{F}(\theta) d\theta$$

Natural Evolutionary Strategies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
- Goal: maximize $\mathbb{E}_{\theta \sim P_\mu} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$
- The variable we are optimizing is the mean $\mu \in \mathbb{R}^d$

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

Natural Evolutionary Strategies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
- Goal: maximize $\mathbb{E}_{\theta \sim P_\mu} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$
- The variable we are optimizing is the mean $\mu \in \mathbb{R}^d$

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

Natural Evolutionary Strategies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
- Goal: maximize $\mathbb{E}_{\theta \sim P_\mu} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$
- The variable we are optimizing is the mean $\mu \in \mathbb{R}^d$

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

Natural Evolutionary Strategies

- Consider a distribution of policy parameters: $\theta \sim P_\mu(\theta)$
- Goal: maximize $\mathbb{E}_{\theta \sim P_\mu} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$
- The variable we are optimizing is the mean $\mu \in \mathbb{R}^d$

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

We will approximate this expectation by sampling!

Sampling from a multivariate Gaussian

- 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 $\sigma^2 I$

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

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

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:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta) \right]$$

$$\approx \frac{1}{2} \left[\mathrm{F}(\theta_1) \frac{\theta_1 - \mu}{\sigma^2} + \mathrm{F}(\theta_2) \frac{\theta_2 - \mu}{\sigma^2} \right], \text{ where } \mathrm{F}(\theta_1) = R(\tau_1), \mathrm{F}(\theta_2) = R(\tau_2)$$

$$= \frac{1}{2\sigma} \left[\mathrm{F}(\theta_1)\epsilon_1 + \mathrm{F}(\theta_1)\epsilon_2 \right]$$

Natural Evolutionary Strategies

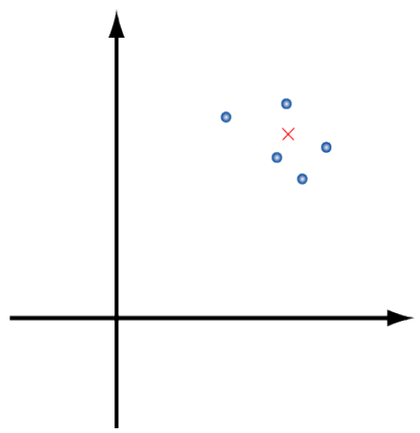
Algorithm 1 Evolution Strategies

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 $F_i = F(\mu_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\mu_{t+1} \leftarrow \mu_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Compare the two update rules for the mean

$$\mu_{t+1} = \mu_t + \frac{\alpha}{n\sigma} \sum_{i=1}^{n} \mathrm{F}(\theta_i)\epsilon_i$$

$$\mu_{t+1} = \sum_{i=1}^{n_{elit}} \theta_i^{elit,t}$$



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

Question

- Evolutionary methods work well on relatively low-dimensional problems (small number of parameters).
- Can they be used to optimize deep network policies?

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

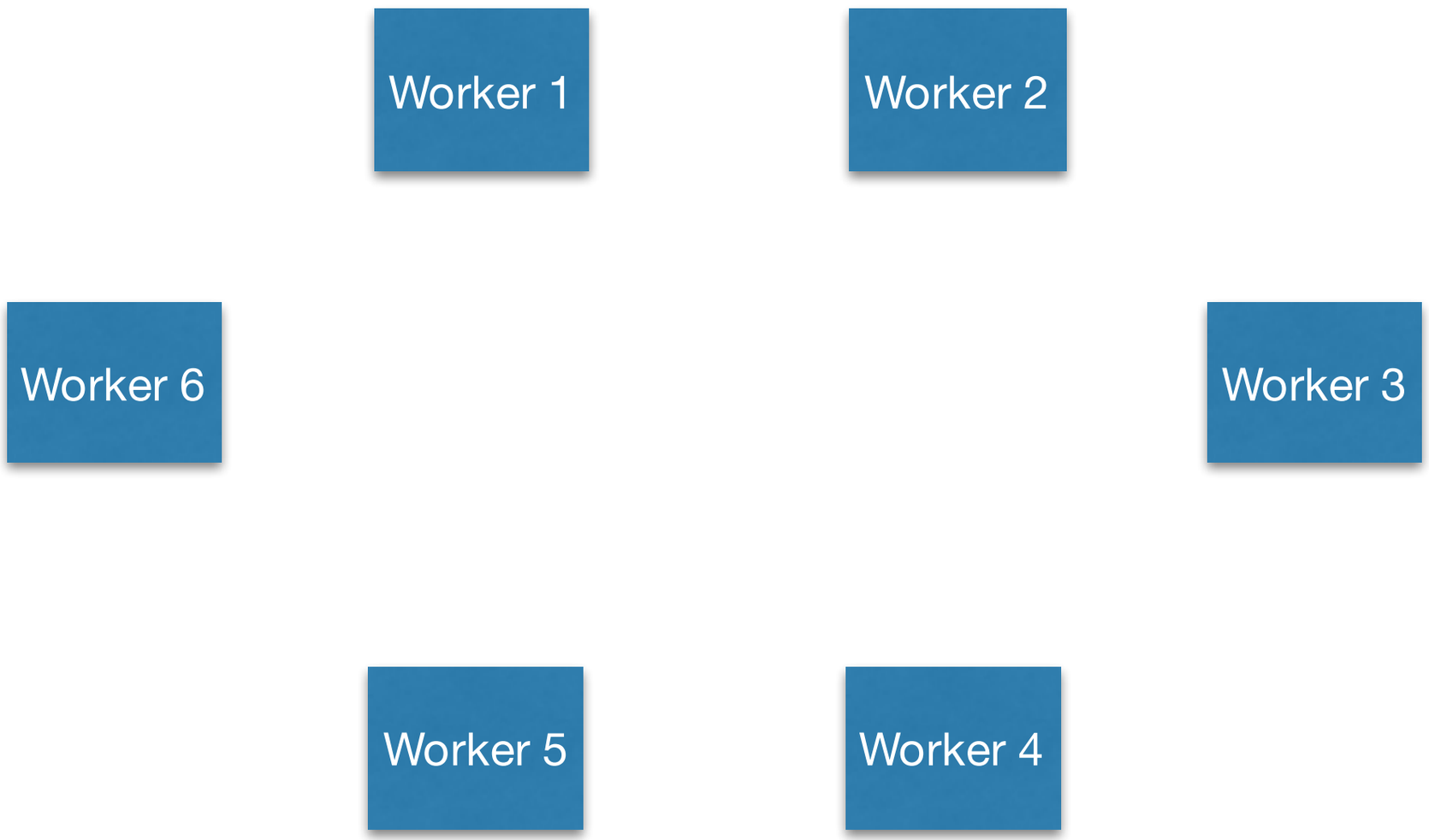
Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

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 $F_i = F(\mu_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\mu_{t+1} \leftarrow \mu_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

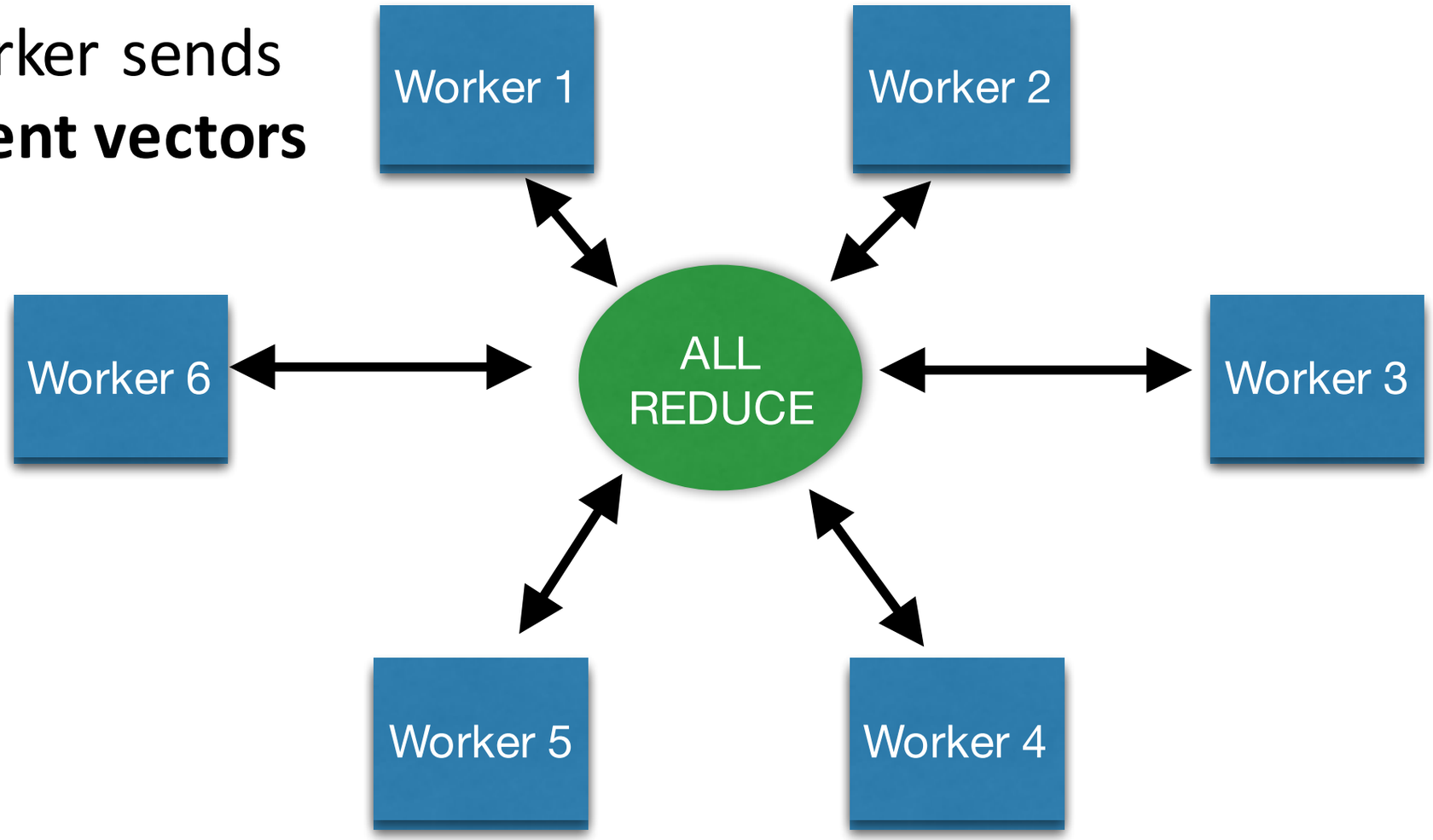
Main contribution: Parallelization with a need for tiny only cross-worker communication

Distributed Deep Learning

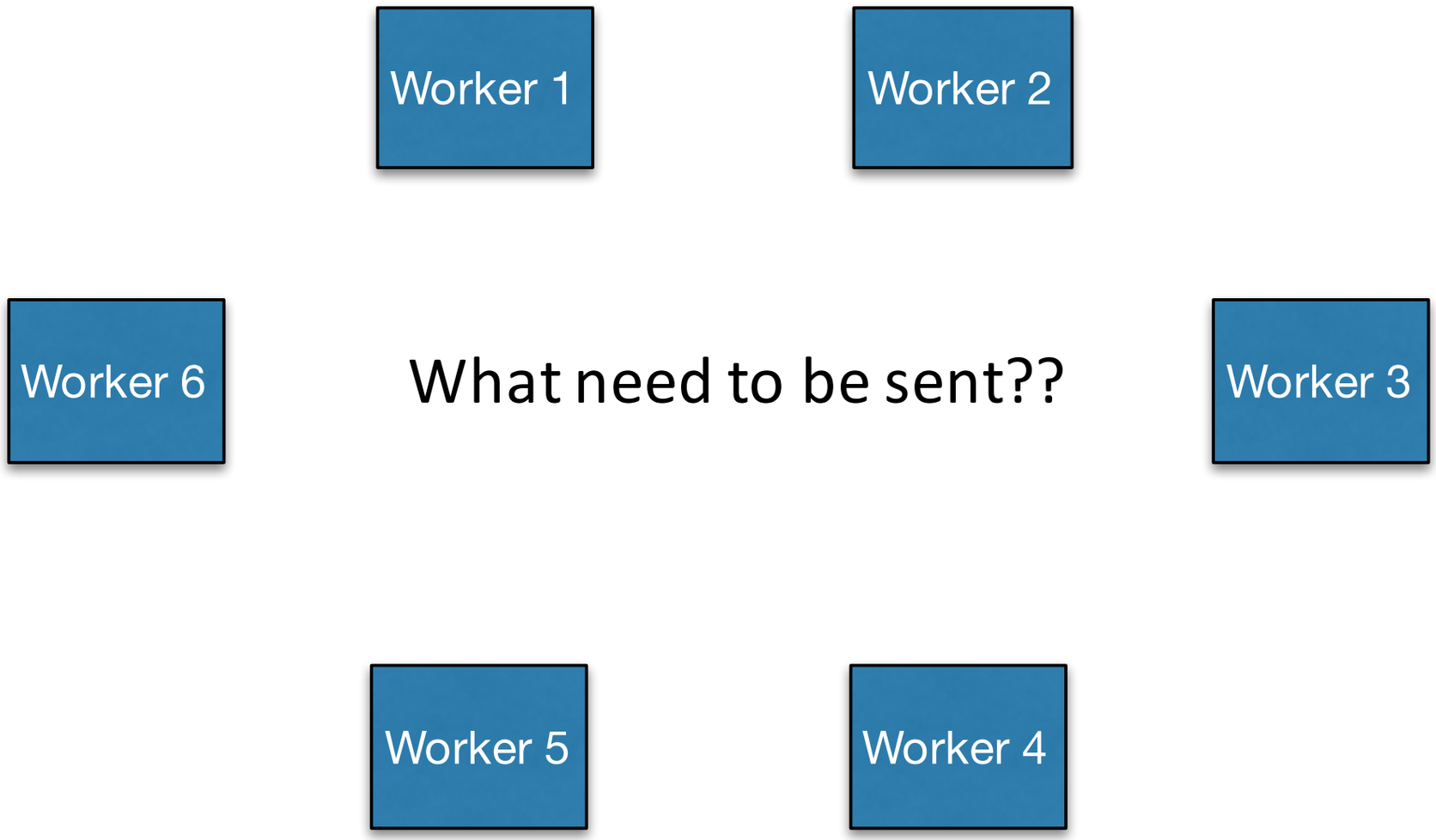


Distributed Deep Learning

Each worker sends
big gradient vectors



Distributed Evolution



Distributed Evolution

Worker 1

Worker 2

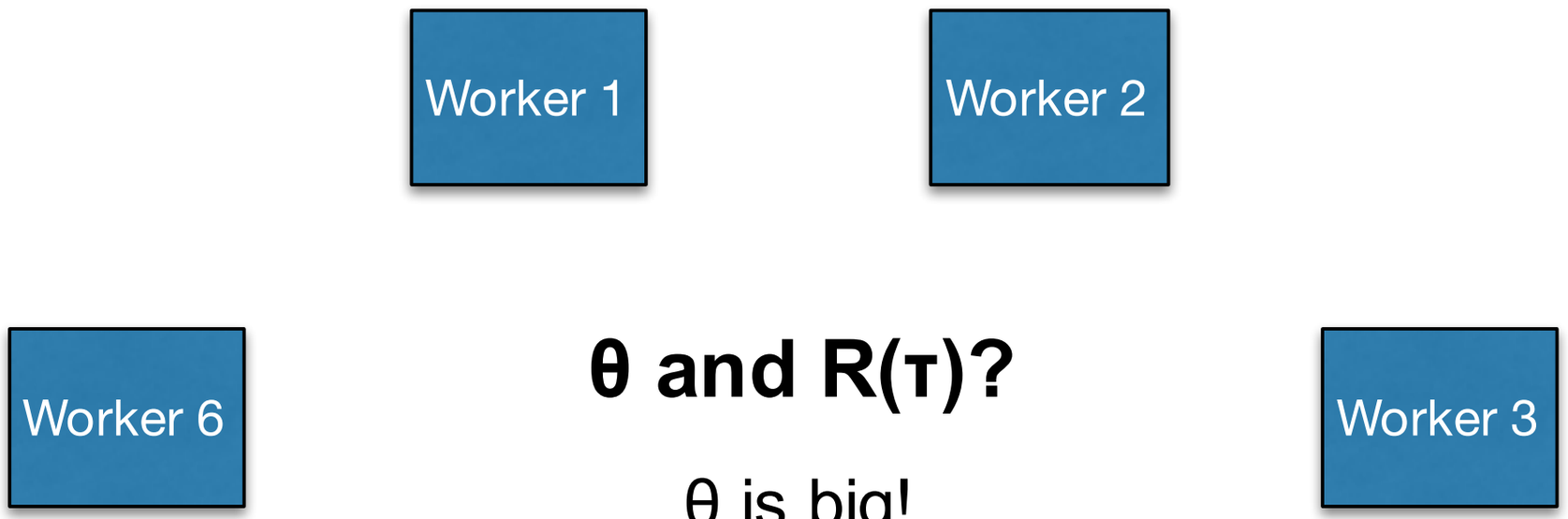
Algorithm 1 Evolution Strategies

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 $F_i = F(\mu_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\mu_{t+1} \leftarrow \mu_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Worker 5

Worker 4

Distributed Evolution



θ and R(τ)?

θ is big!

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

Same for all workers

Only need seed of random number generator!

Distributed Evolution

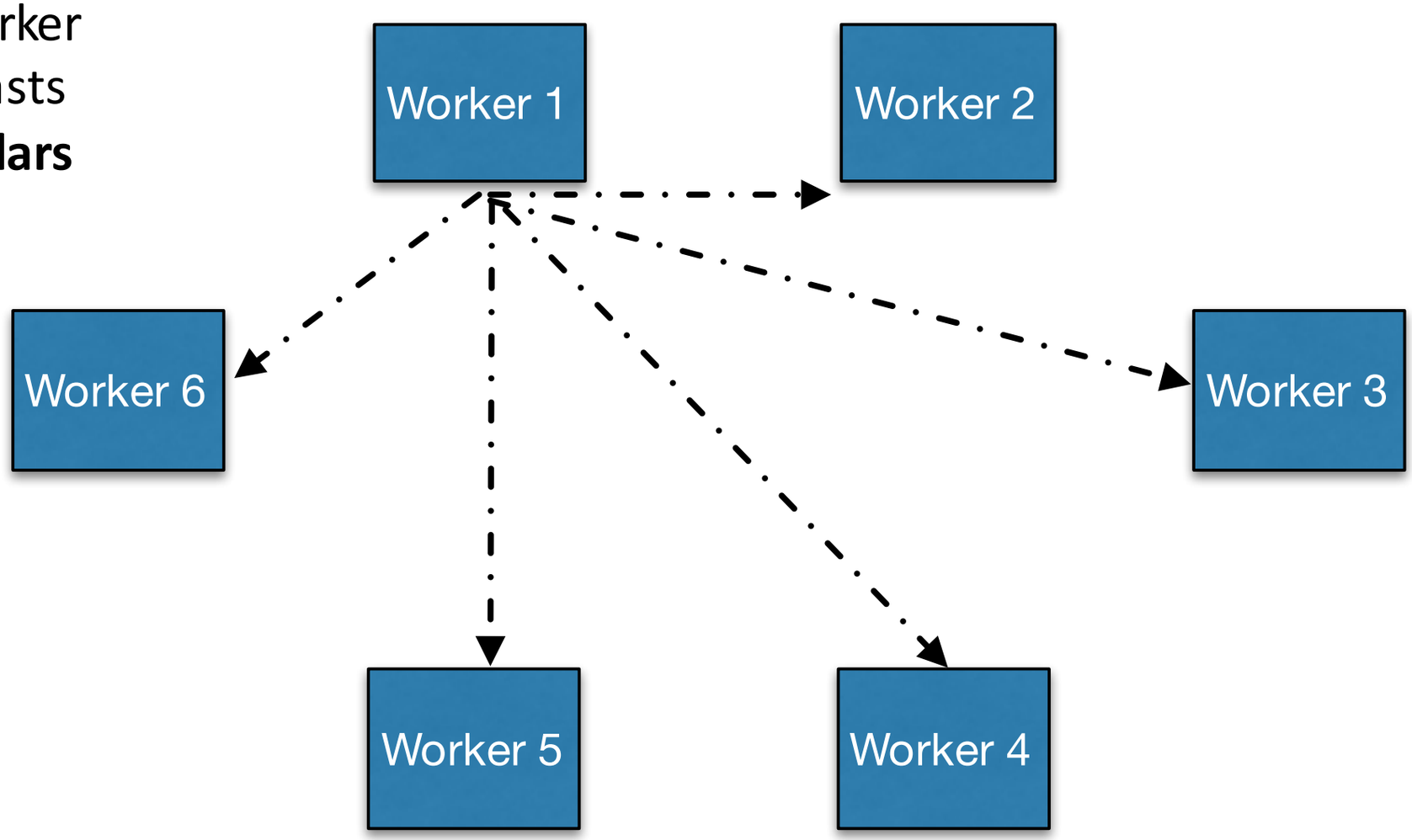
Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with [redacted] 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 $F_i = F(\mu_t + \sigma\epsilon_i)$
  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 $\mu_{t+1} \leftarrow \mu_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

[Salimans, Ho, Chen, Sutskever, 2017]

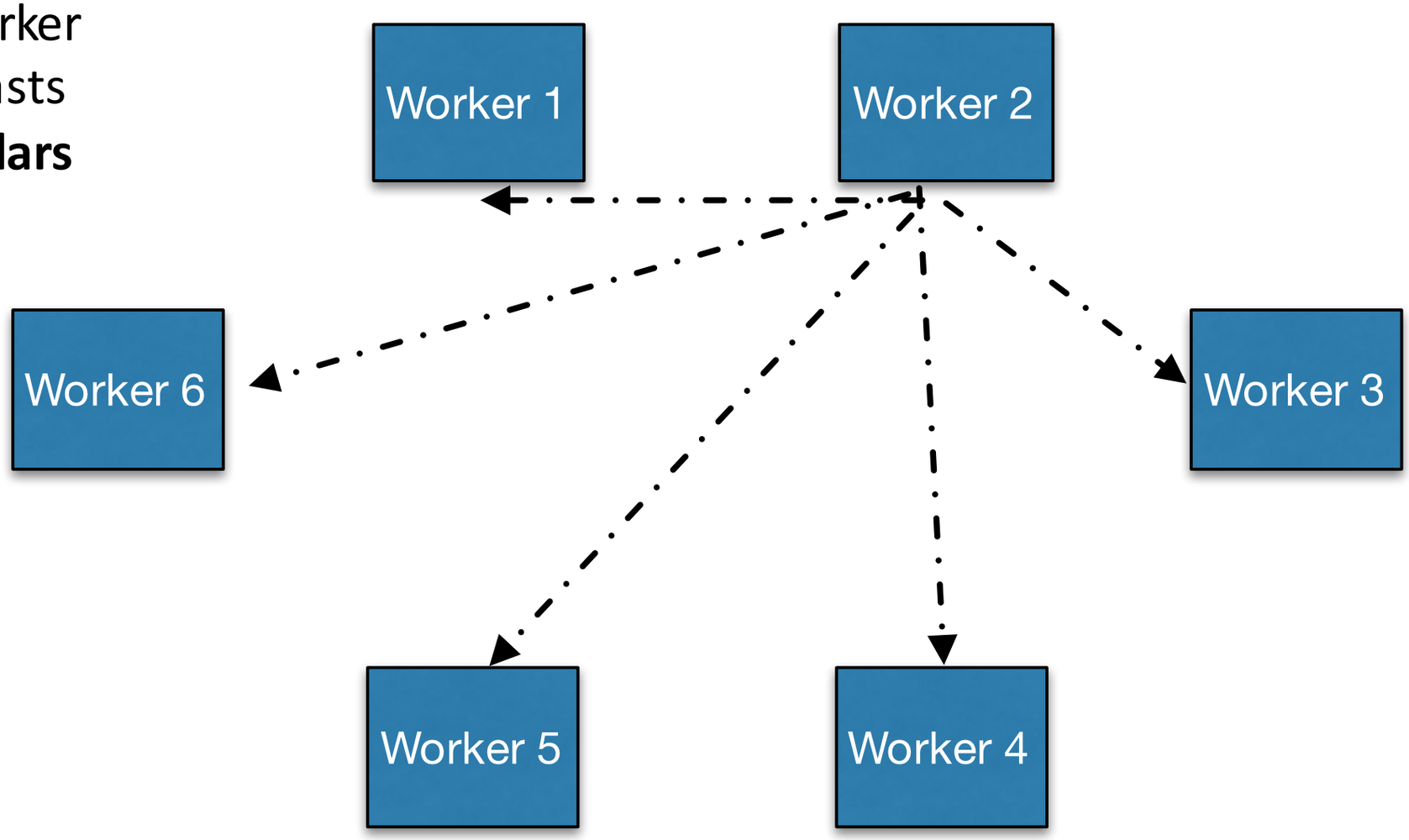
Distributed Evolution

Each worker broadcasts **tiny scalars**



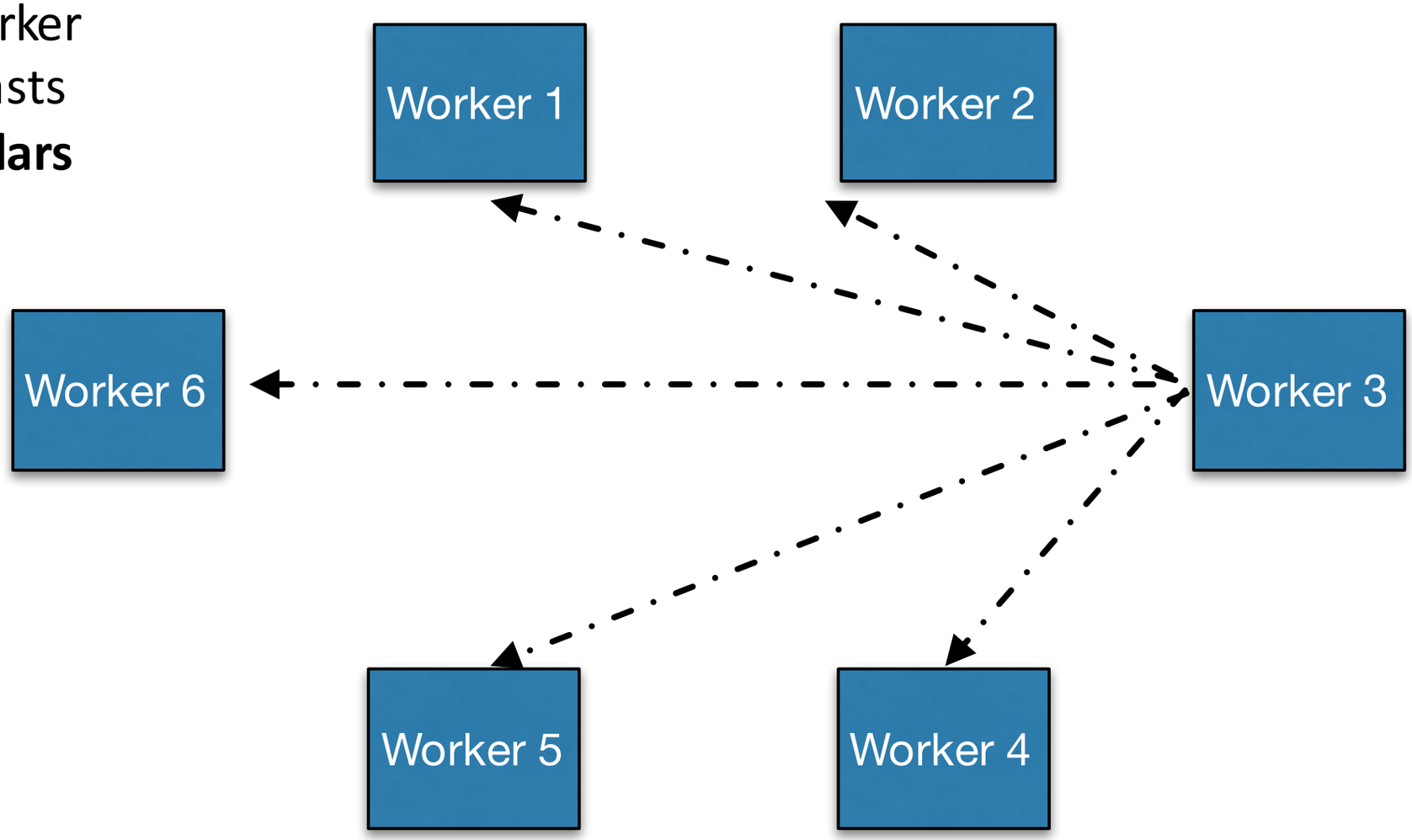
Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution Scales Very Well :-)

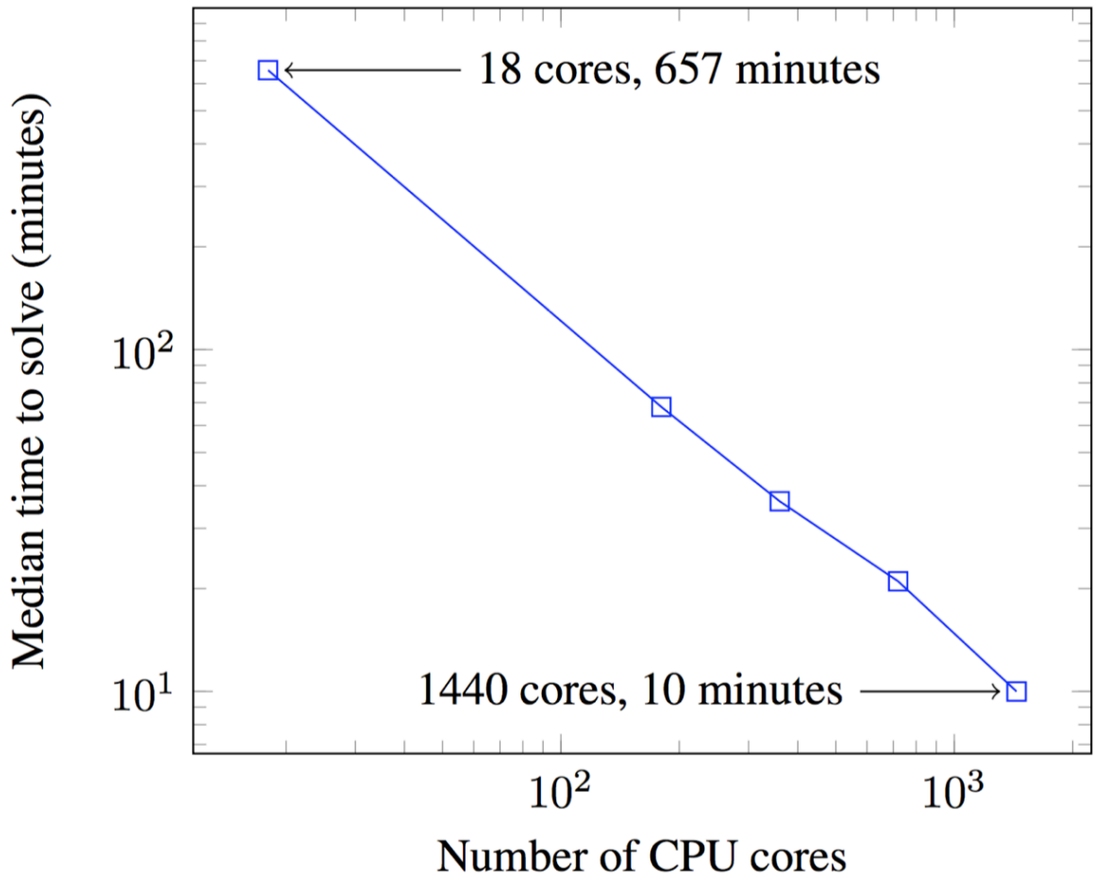
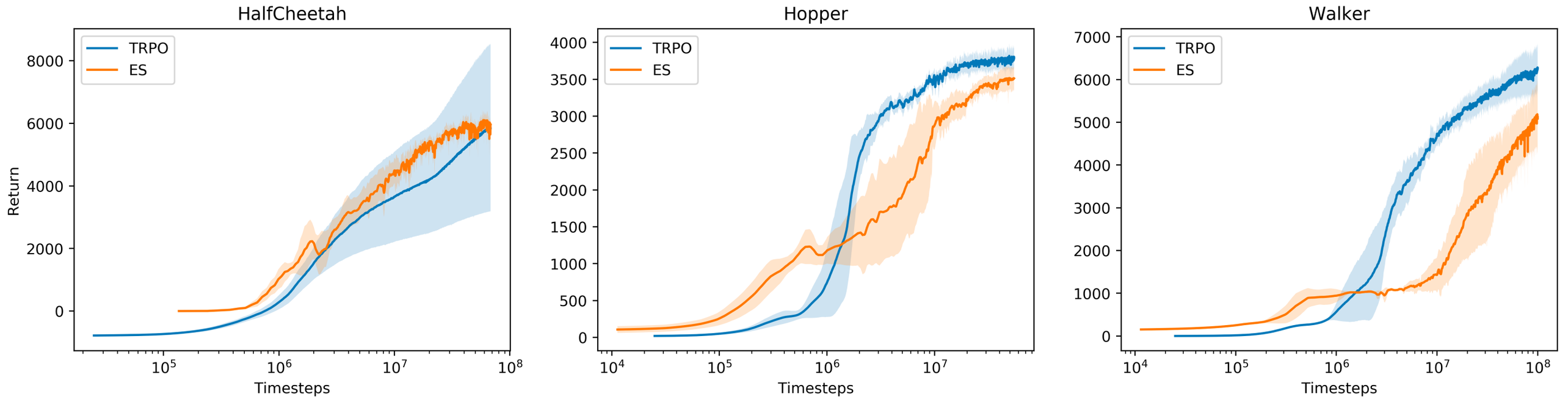


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

Salimans, Ho, Chen, Sustskever. 2017

Distributed Evolution Requires More Samples :-(



Salimans, Ho, Chen, Sustskever. 2017

Natural Evolution beyond RL

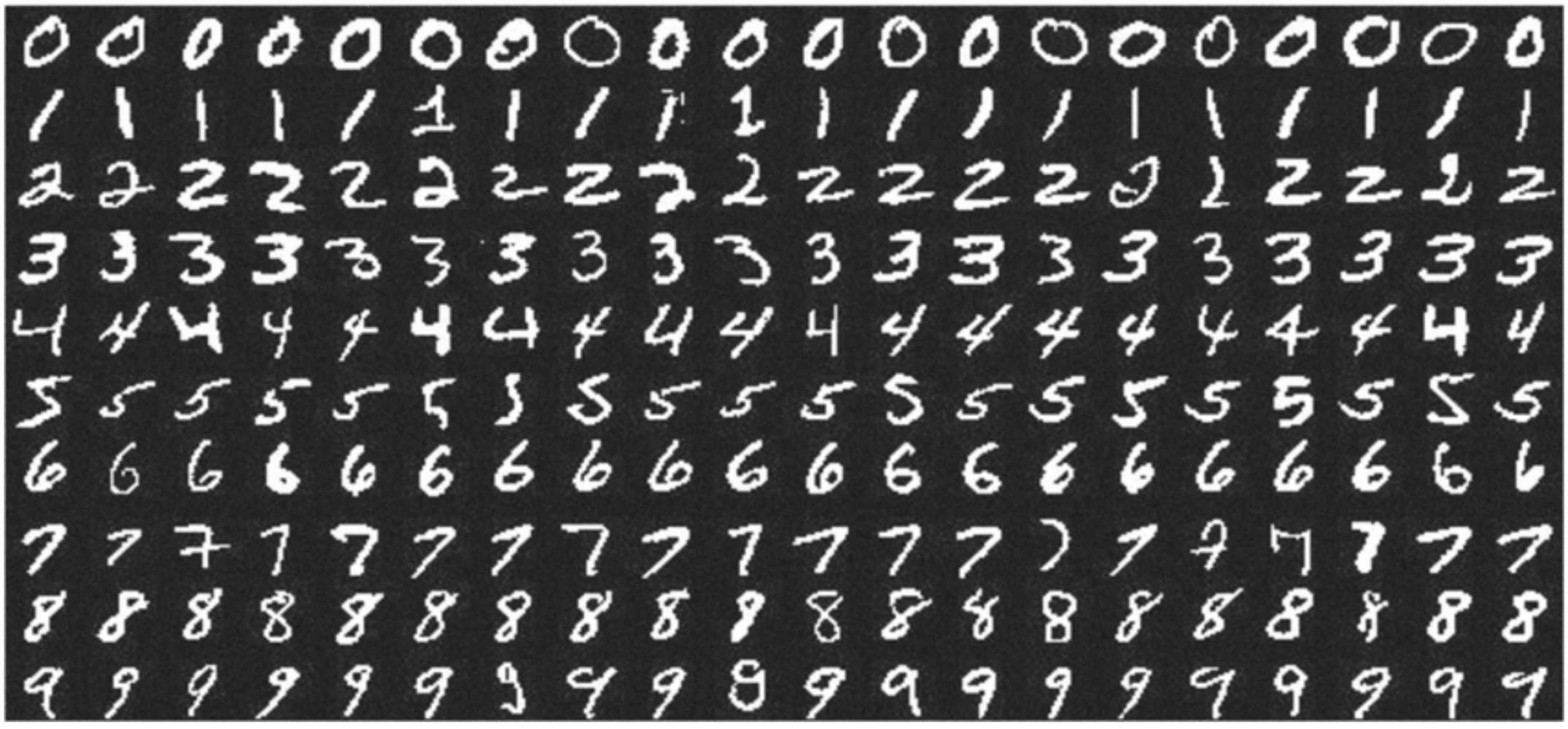
- Q: Why we do not use evolutionary methods for training image classifiers?
- A: In this case, the gradient is available: we have a direction to take a step in our parameter space, (in high dimensionas SGD is not easily trapped in local minima). In low dim non-convex problems, potentially such methods would also make sense, despite the fact we can differentiate.

Comparing ES with SGD in MNIST

On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent

By: Xingwen Zhang, Jeff Clune, Kenneth O. Stanley (Uber AI labs)

Slides copied from William Saunders, UoT



Parameter Update for ES

- Generate n pseudo-offspring by adding perturbation v_i sampled from uniform Gaussian with mean 0, covariance I $\theta + \sigma v_i$
- Compute rewards for each offspring (r_i)
- Estimate gradient

$$g^{ES} = \frac{1}{n\sigma} \sum_{i=1}^{n} v_i r_i$$

- Update parameters using gradient estimate with any optimizer (ie. ADAM)

Comparing ES with SGD in MNIST

- We can use ES gradient as an inefficient way to train a classification network on MNIST, to compare it's performance to the SGD gradient
- We can compute the correlation between the SGD gradient and ES gradient for each minibatch
- Initially achieved 96.99% validation accuracy with 10,000 pseudo-offspring, 2000 iterations on a network with 3,274,634 parameters

Gradient Correlation

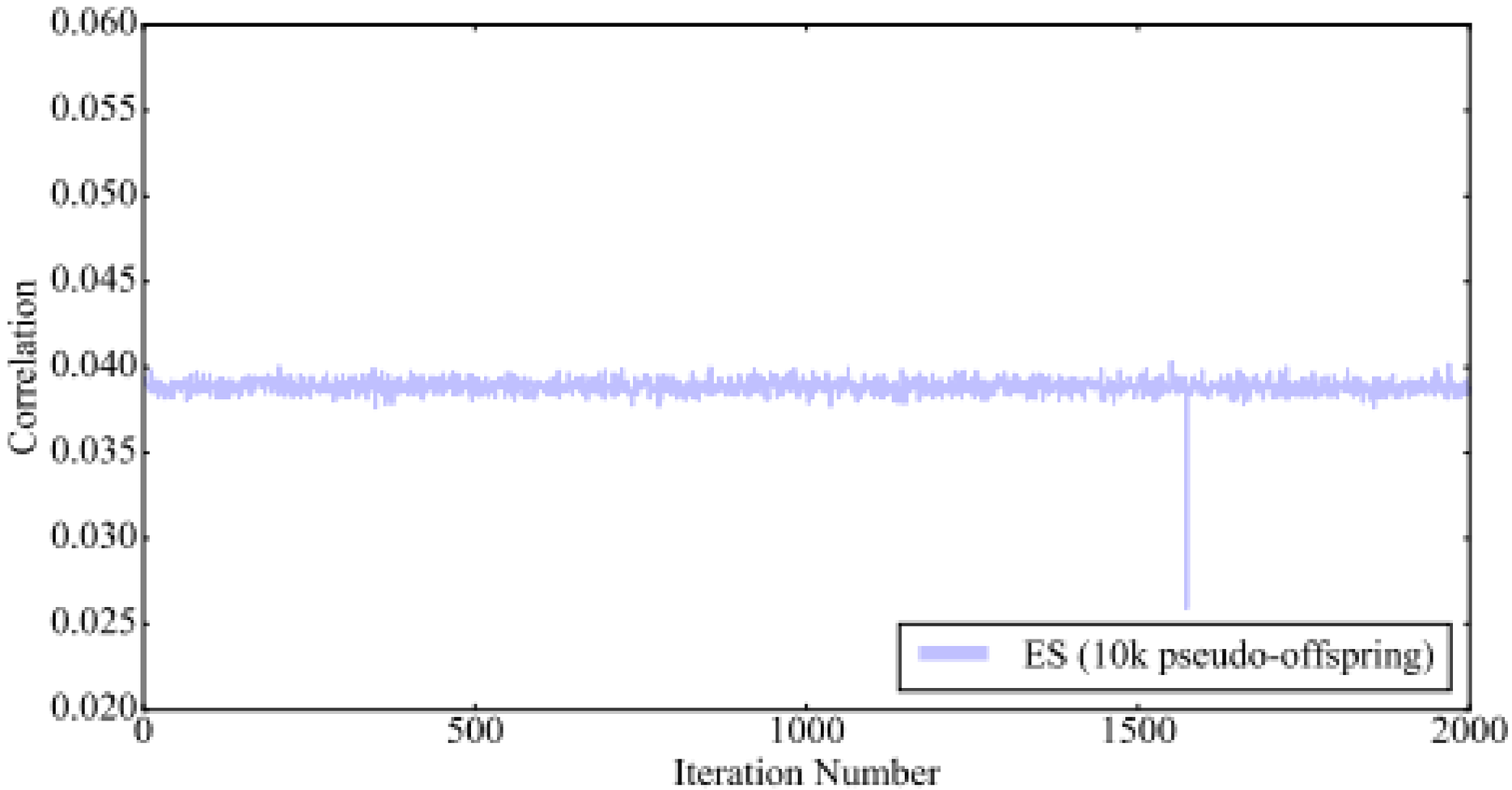


Figure 1: **Correlation of the gradients estimated by ES and the analytic gradients for the same sequence of mini-batches**. The correlation between ES and gradients used by SGD is remarkably stable.

Noisy SGD is a Good Proxy for Natural Evolution

- It's remarkable that the 3.9% correlation allows for validation accuracy within 1.7% of SGD accuracy
- The correlation between ES and SGD gradient is stable over time
- We can simulate ES gradient by adding uniform noise to SGD gradient to achieve the same correlation to the SGD gradient
- It's more efficient to run experiments on algorithm changes with this SGD + noise proxy

Noisy SGD is a Good Proxy for Natural Evolution

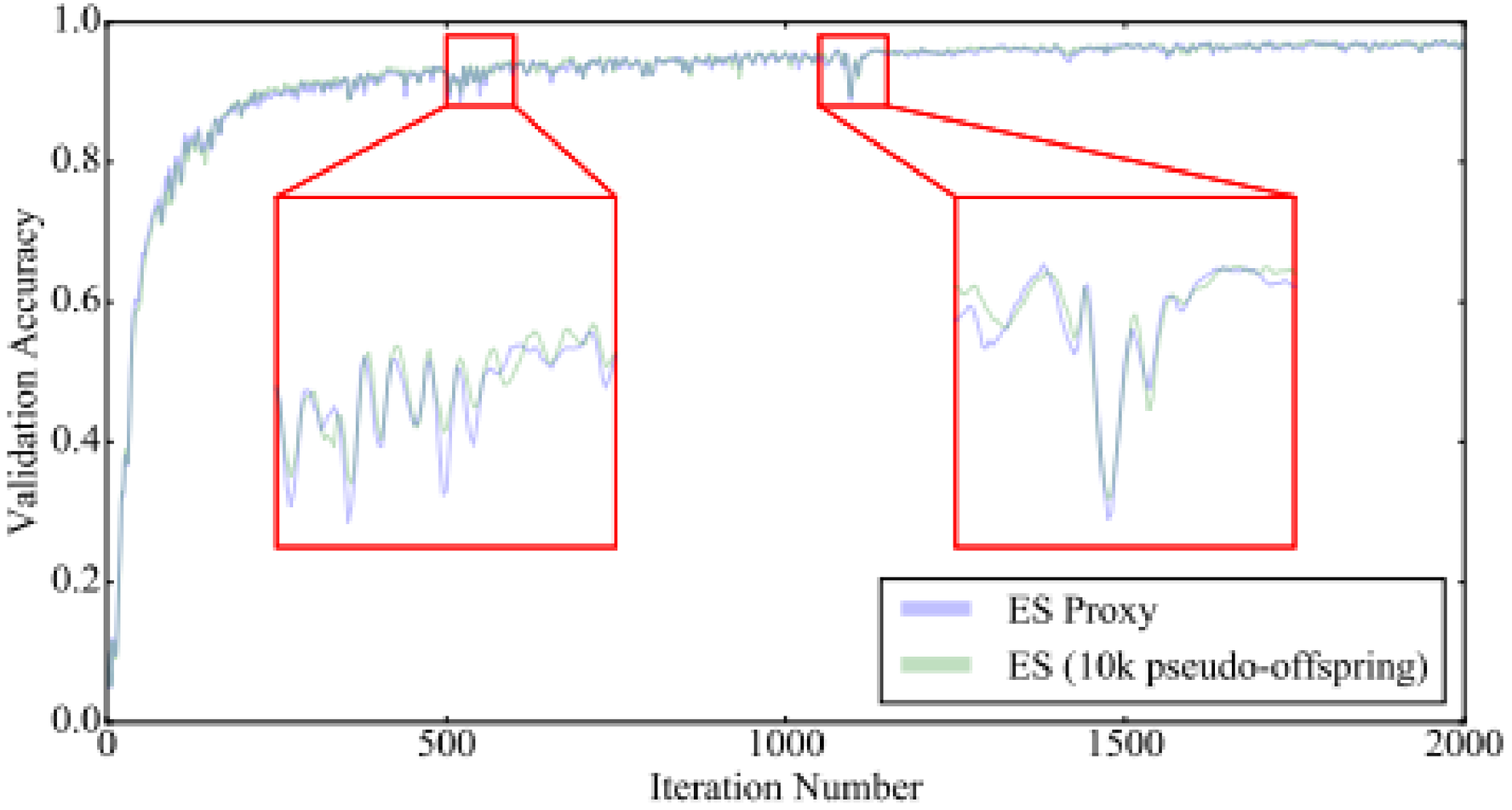


Figure 2: **Validation accuracy of ES and the ES proxy for the same sequence of mini-batches.** Notice how the fluctuations in performance by ES and its proxy are nearly identical throughout the run despite the randomness in the SGD proxy and the ES pseudo-offspring. Insets show local areas of the curves close up. The implication is that mini-batches are the primary driver of noise in the search, impacting ES and SGD in the same way. Recall that validation accuracy over iterations in this figure and throughout this paper is reported on the entire MNIST test set.

MNIST with Neuro-Evolution

- Limited the number of dimensions that are perturbed for each pseudo-offspring, improving performance but requiring more pseudo-offspring
- Achieve 99% accuracy on MNIST using 50,000 pseudo-offspring, 10,000 training batches
- But used a smaller network (2 layers, 28938 parameters)

Conclusions

- Evolutionary strategies are competitive to SGD since the gradient is in any case noisy in RL problems
- When optimizing differentiable objectives with deep nets, they do not have much to affer: much more expensive and more noisy than the true gradient.
- ES performance much depends on the number of workers for high dimensional problems.
- Smart trick to avoid a lot of communication between workers by sending scalar rewards and sharing the random seeds. Easier to scale up.
- We should always make sure our method beats this obvious baseline: NES with reasonable (not outrageous) resource allocation.