Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

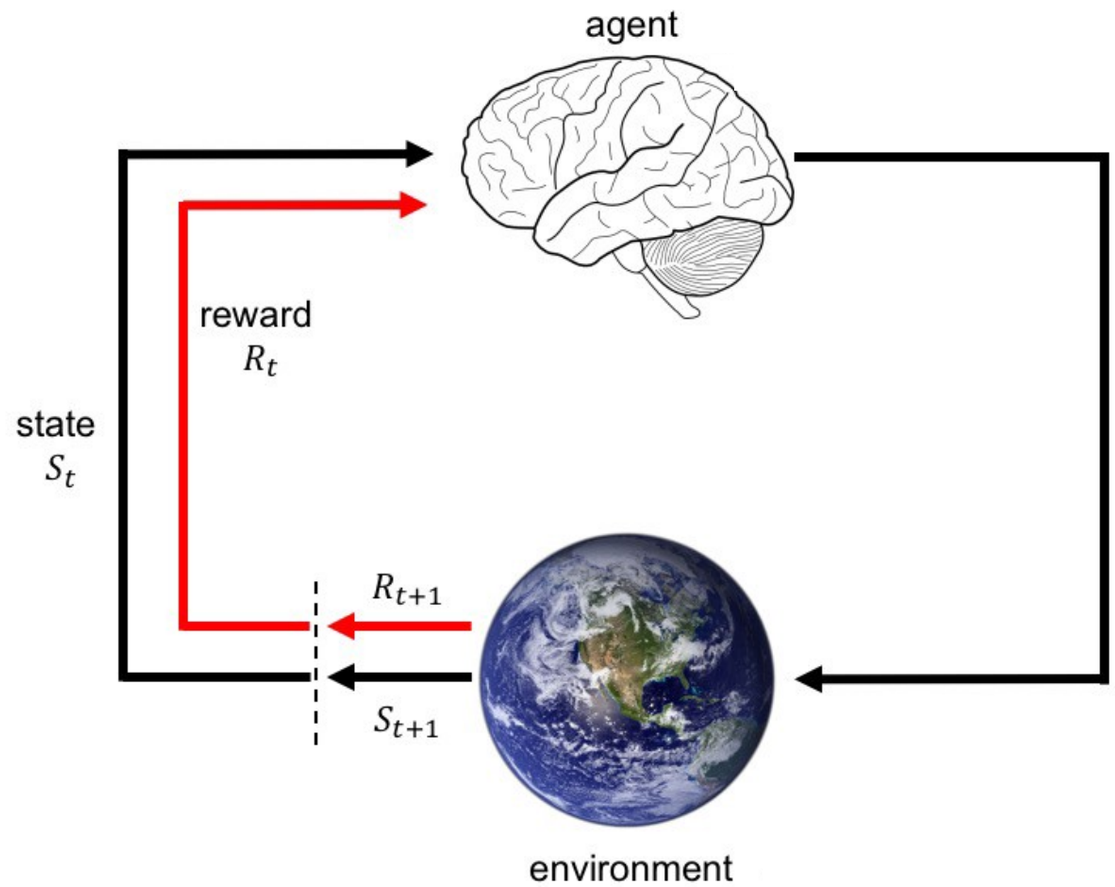
Evolutionary Methods for Policy Search

Spring 2021, CMU 10-403

Katerina Fragkiadaki

Reinforcement Learning

Learning behaviours from rewards while interacting with the environment



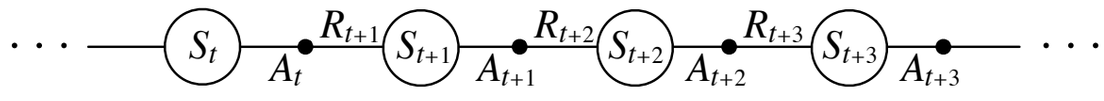
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

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: $S_{t+1} \in \mathcal{S}^+$



Markovian States

- A state captures whatever information is available to the agent at step t about its environment.

- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations, memories etc.

- A state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:
  $P\left[R_{t+1} = r, S_{t+1} = s' \,|\, S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\right] = P\left[R_{t+1} = r, S_{t+1} = s' \,|\, S_t, A_t\right]$

  for all $s' \in \mathrm{S}, r \in \mathrm{R}$, and all histories

- We should be able to throw away the history once state is known

Rewards reflect goals

Rewards are scalar values provided by the environment to the agent that indicate whether goals have been achieved, e.g., 1 if goal is achieved, 0 otherwise, or -1 for overtime step the goal is not achieved

- Goals specify what the agent needs to achieve, not how to achieve it.

- The simplest and cheapest form of supervision, and surprisingly general: All of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward):

$$r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Goal seeking behaviour, achieving purposes and expectations can be formulated mathematically as maximizing expected cumulative sum of scalar values…

The agent learns a Policy

Definition: A policy is a distribution over actions given states,

$$\pi(a \mid s) = \mathbf{Pr}(A_t = a \mid S_t = s), \forall t$$

- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes his policy as a result of experience

Special case: deterministic policies

$\pi(s) =$ the action taken with prob $= 1$ when $S_t = s$

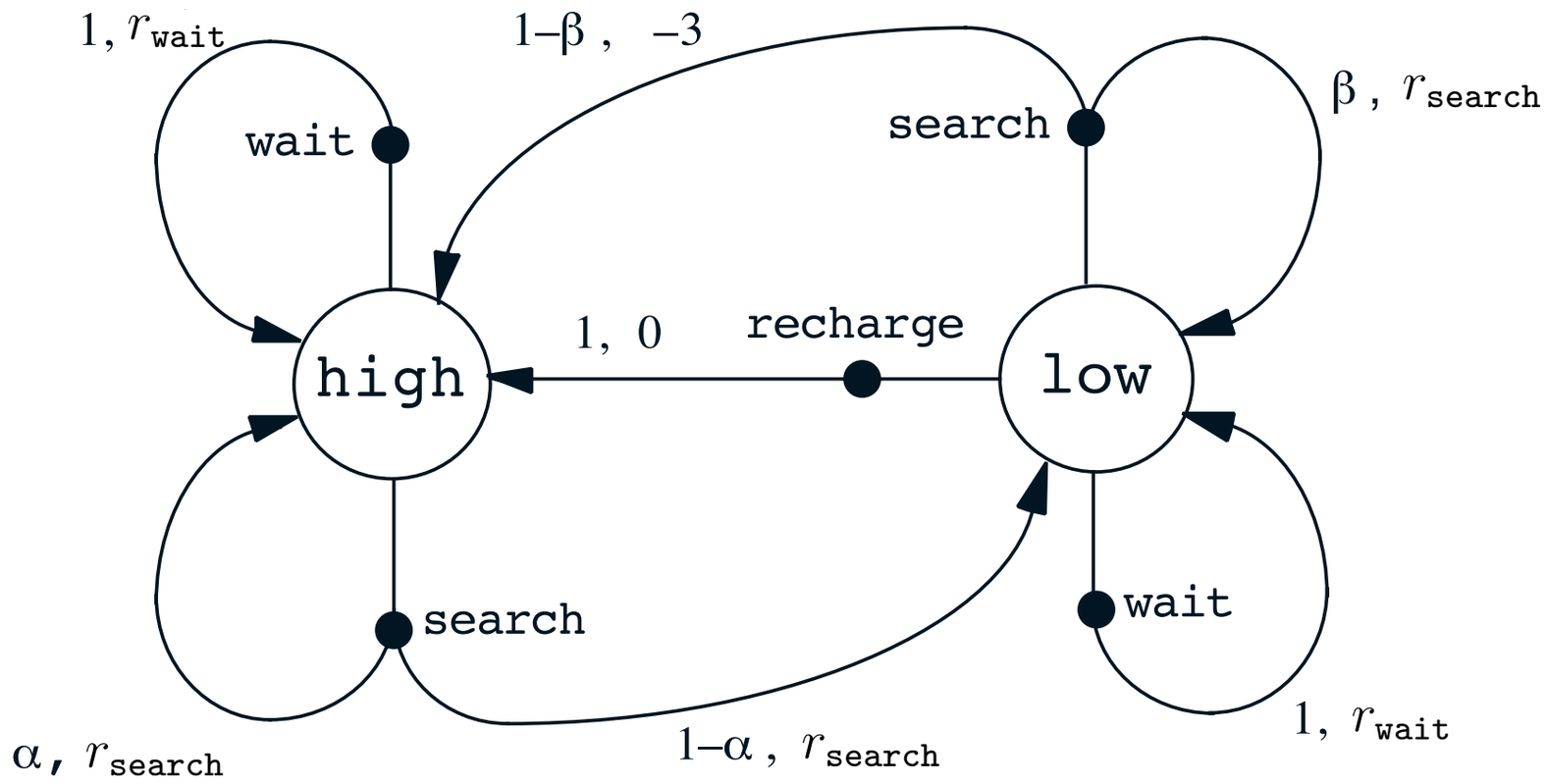
Example with few states: The recycling robot

$$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$$
$$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$$
$$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$$r_{\texttt{search}} > r_{\texttt{wait}}$$



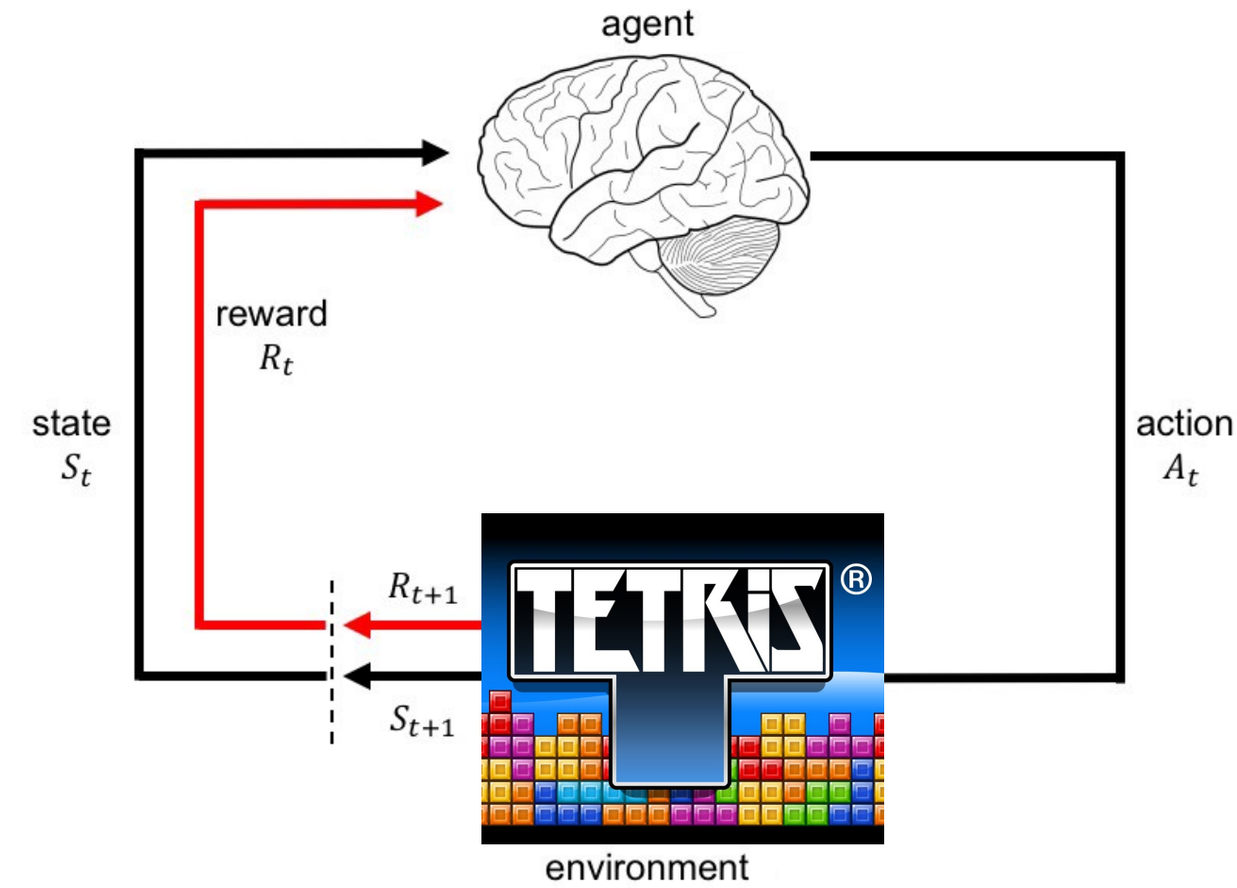
Q: what the robot will do depends on the number of cans he has collected thus far?

Example with few states: The recycling robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.
- Reward = number of cans collected

Example with many states: Tetris

Learning behaviours from rewards while interacting with the virtual environment



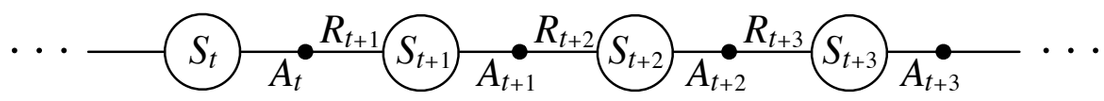
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

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

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

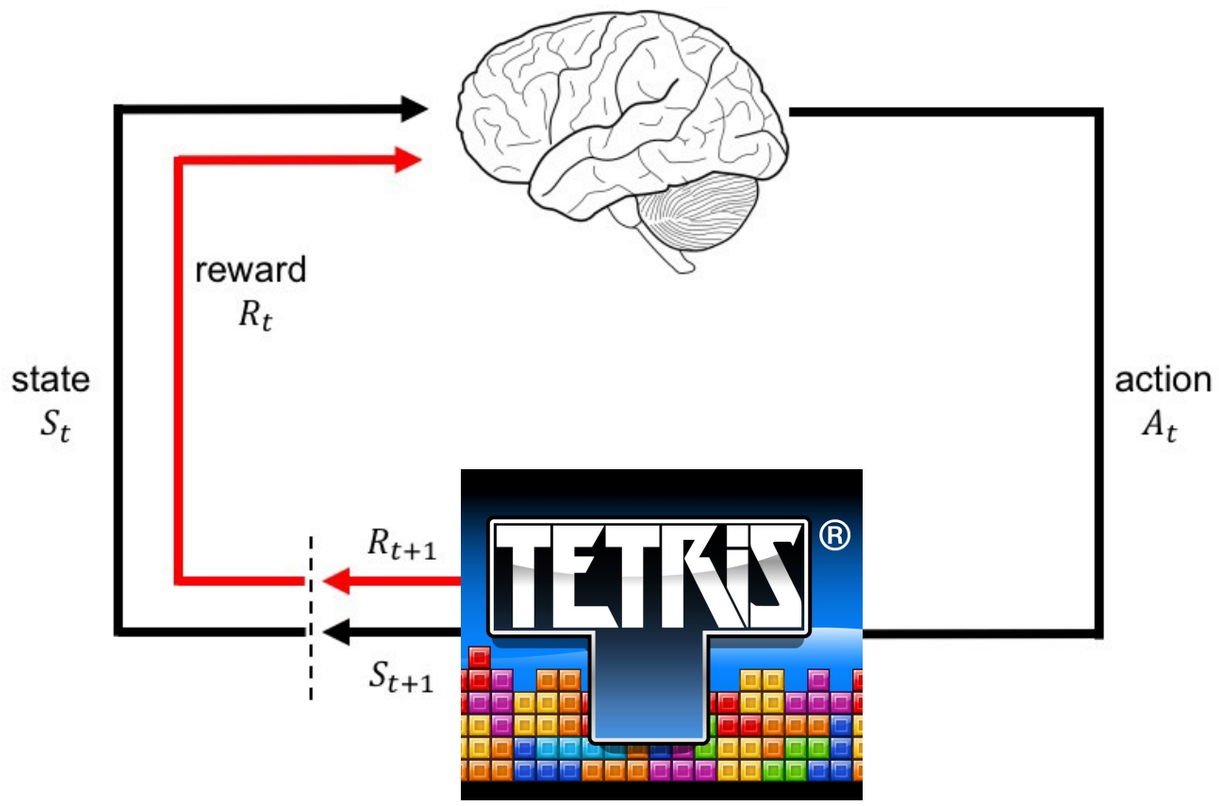
gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

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.

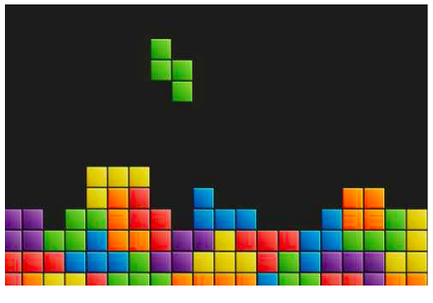


Our goal is to learn a policy that maximizes the score of the game *in expectation*.

- 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.

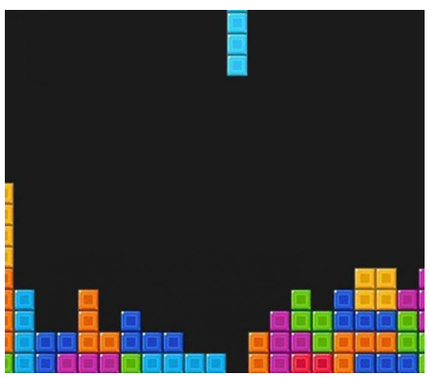
Policy search for Playing Tetris

Policy: a mapping from states to actions



Go right

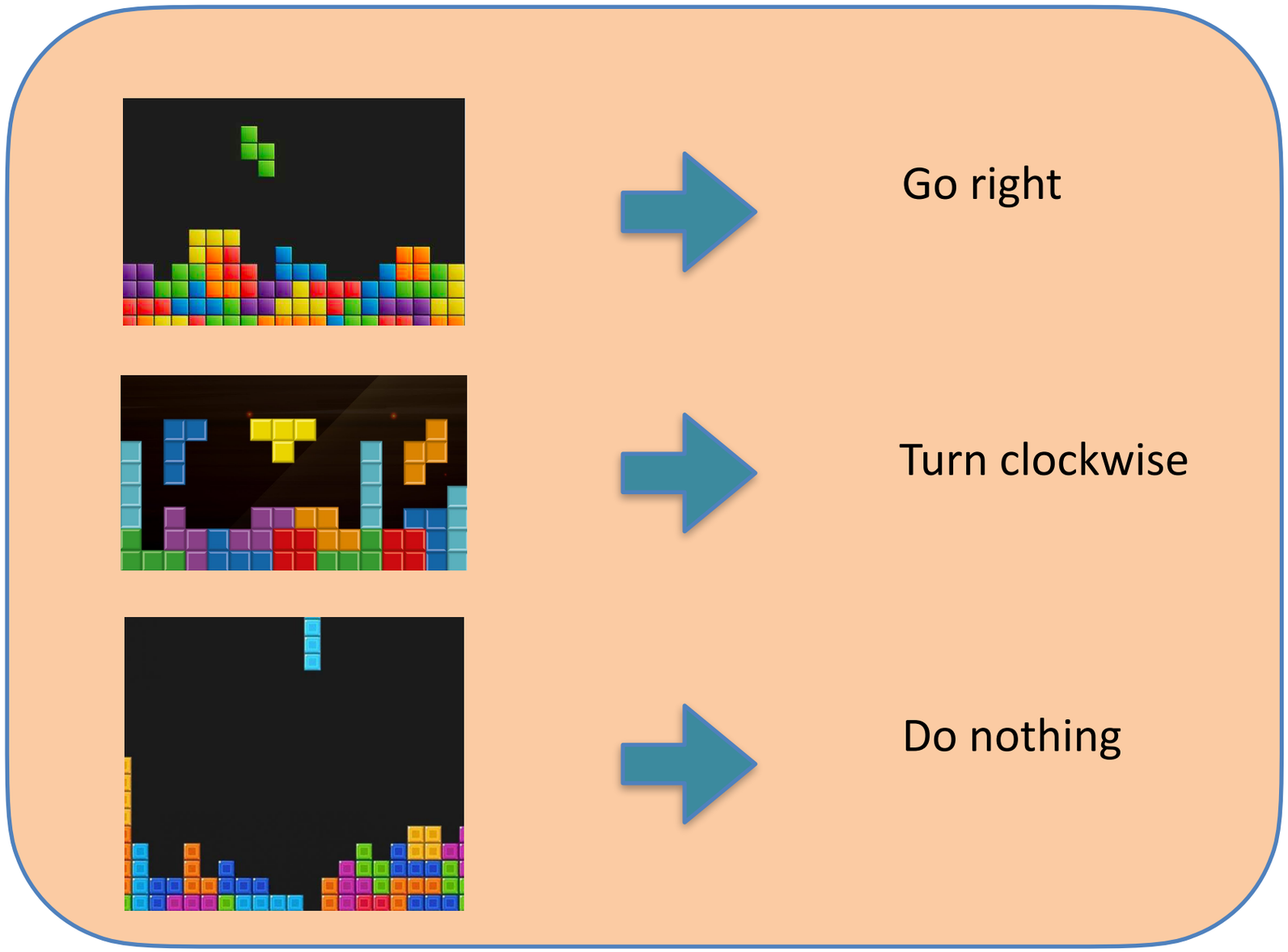
Turn clockwise



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 month of him playing 24/7.

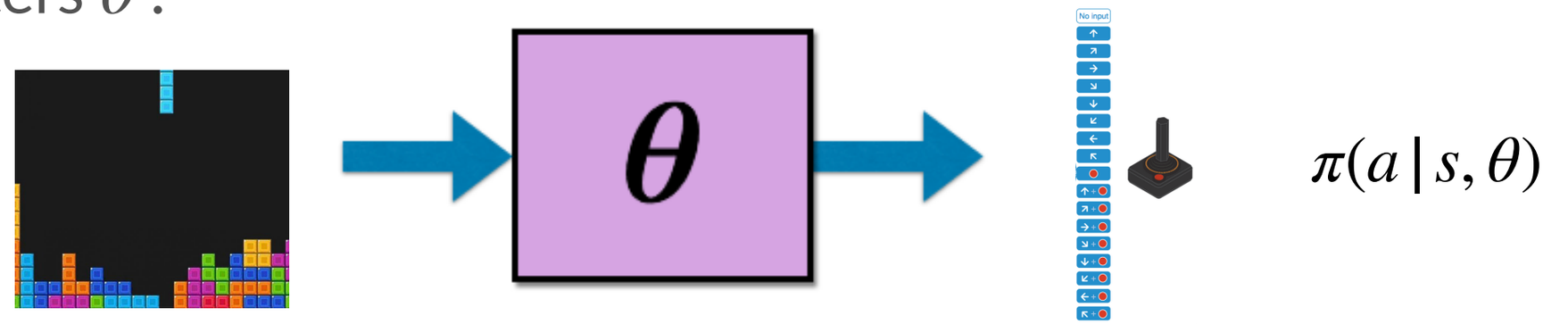
Q1: During the training period, could the agent see **all** states in Tertis, to figure out what is the corresponding best action and create the table?

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.



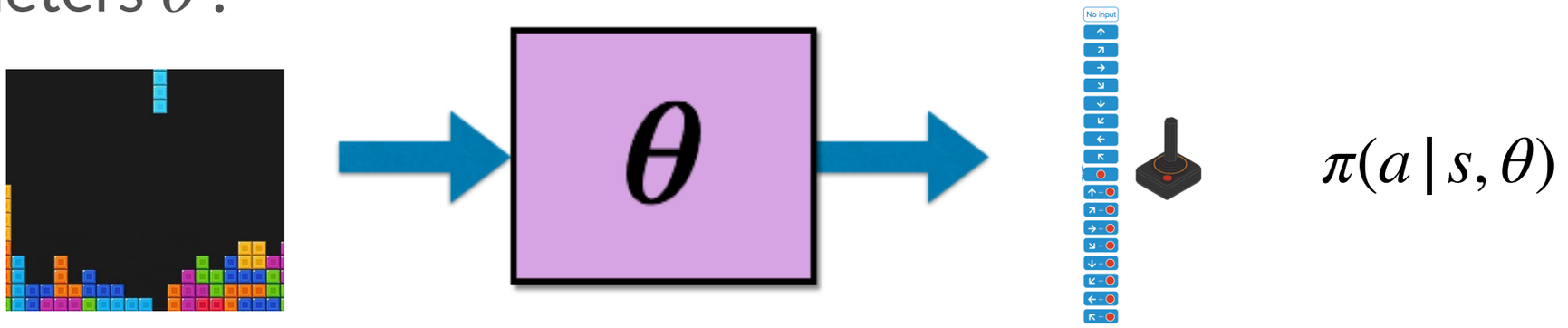
$$\pi(a \mid s, \theta)$$

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?

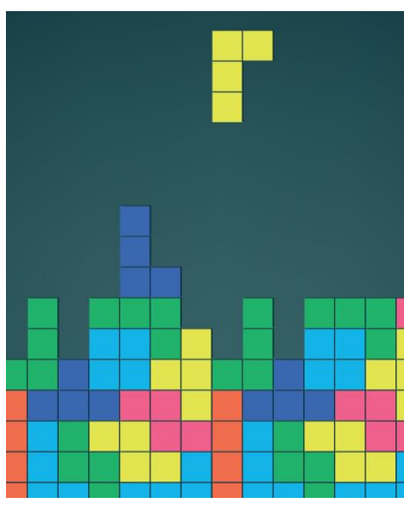
Policy search for Playing Tetris: function approximation

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



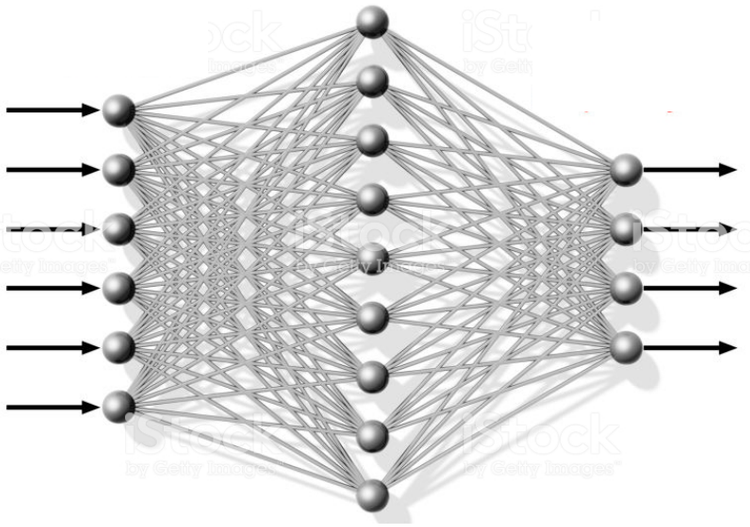
$$\pi(a \mid s, \theta)$$

- 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

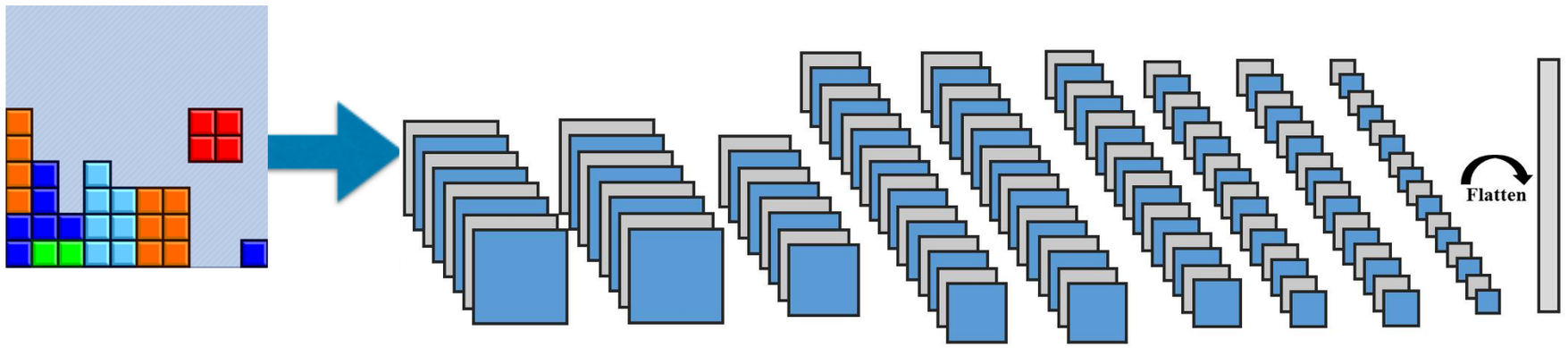


$$\pi(a \mid s, \theta)$$

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?

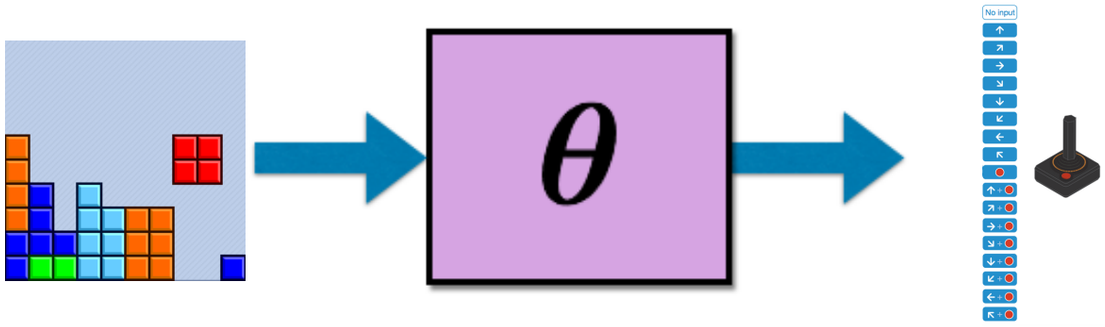


$$\pi(a \mid s, \theta)$$

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.

Reinforcement Learning



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

Given an initial state distribution $\mu_0(s_0)$, estimate parameters θ of a policy π_θ so that, the trajectories τ sampled from this policy have maximum returns, i.e., sum of rewards $R(\tau)$.

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

τ : trajectory, a sequence of state, action, rewards, a game fragment or a full game:

$$\tau : s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots s_T, a_T, r_T$$

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

$$R(\tau) = \sum_{t=1}^{T} r_t$$

Black-box policy optimization

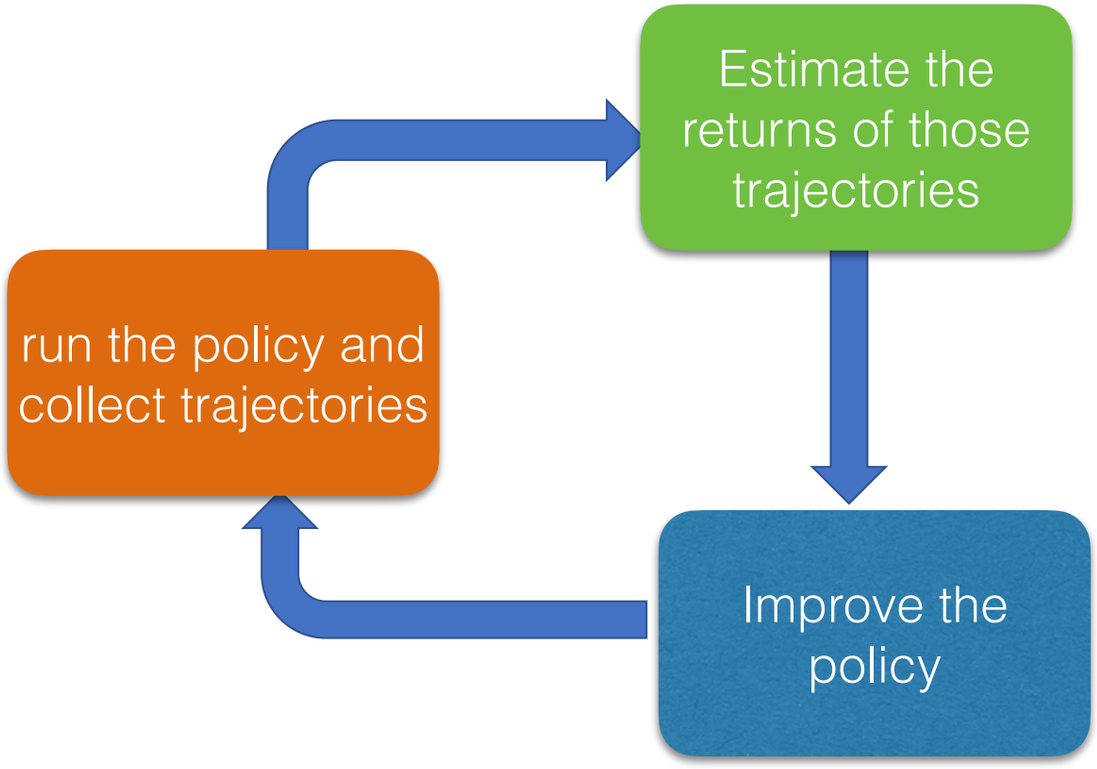
Initialize the policy parameters θ randomly.

1. Perturb policy parameters,

2. Run the resulting policy, collect trajectories and evaluate their returns.

3. Promote **the policy para**meters that resulted in trajectories that gave the largest return improvement.

4. GOTO 1.



- 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.

Evolutionary methods for policy search

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

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…

Gaussian Density

Perhaps the most common probability density

$$\mathcal{N}\left(y|\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$

σ^2 is the variance of the density and μ is the mean.

Multivariate Gaussian Density

Perhaps the most common probability density

$$\mathcal{N}(\mathbf{y} \mid \mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{y}-\mu)^\top \mathbf{\Sigma}^{-1}(\mathbf{y}-\mu)\right)$$

Cross-entropy method (CEM)

- Policy parameters are sampled from a multivariate Gaussian distribution with a diagonal covariance matrix.
- We will update the mean and variances of the parameter elements towards samples that have highest fitness scores.

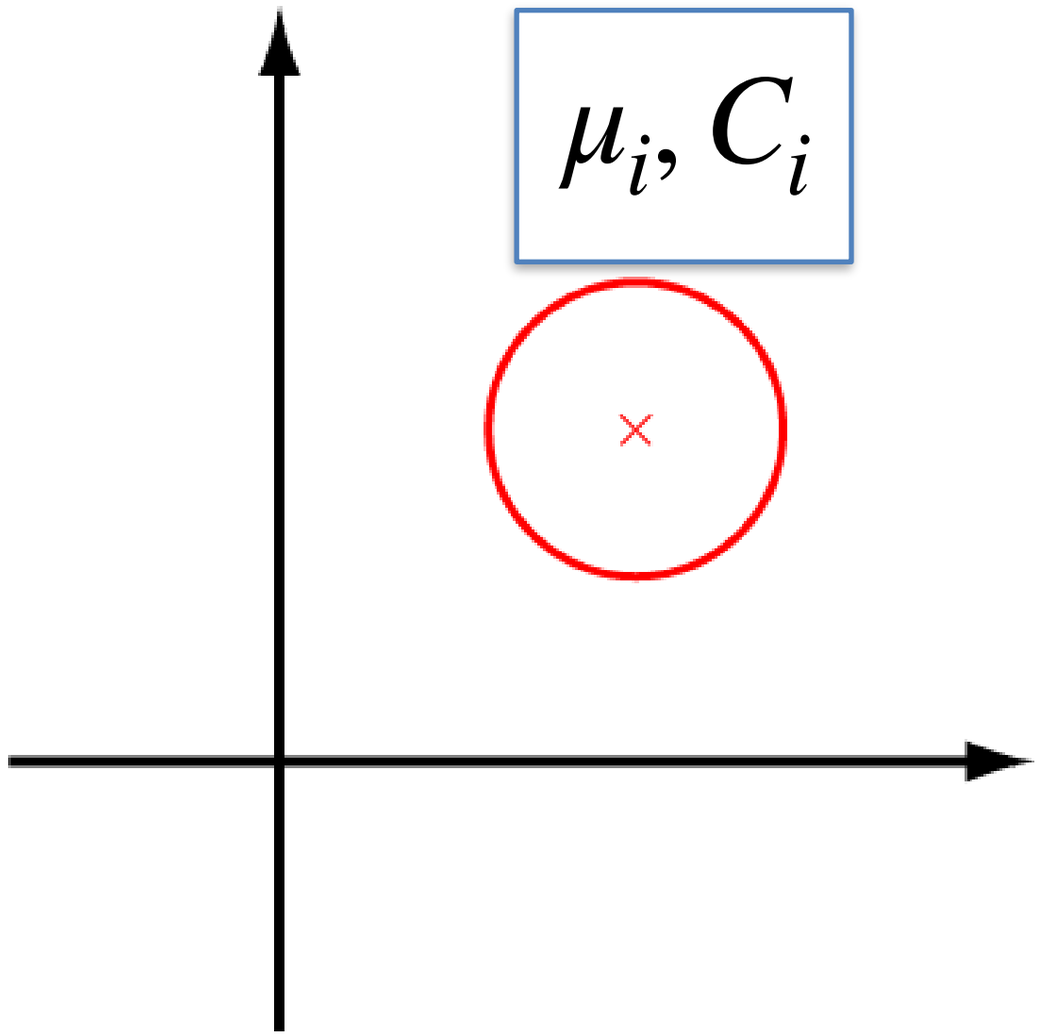
Input: parameter space Θ, number of parameter vectors n, proportion $\rho \leq 1$, noise η
Initialize: Set the parameter $\mu = \bar{0}$ and $\sigma^2 = 100I$ (I is the identity matrix)
for $k = 1, 2, \ldots$ **do**
  Generate a random sample of n parameter vectors $\{\theta_i\}_{i=1}^{n} \sim \mathcal{N}(\mu, \sigma^2 I)$
  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 (CMA-ES)

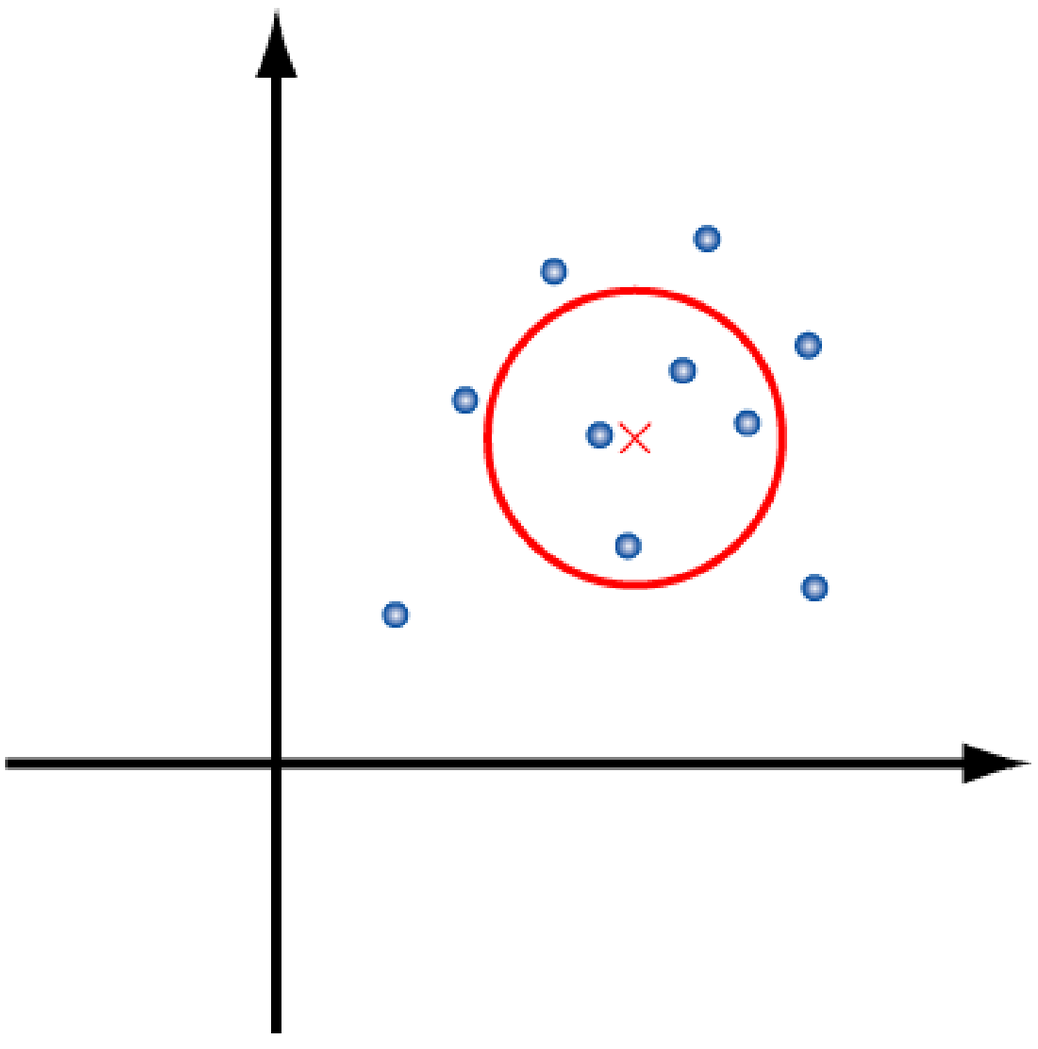
- Policy parameters are sampled from a multivariate Gaussian distribution with a full covariance matrix.
- We will update the mean and variances of the parameter elements towards samples that have highest fitness scores.

Covariance Matrix Adaptation



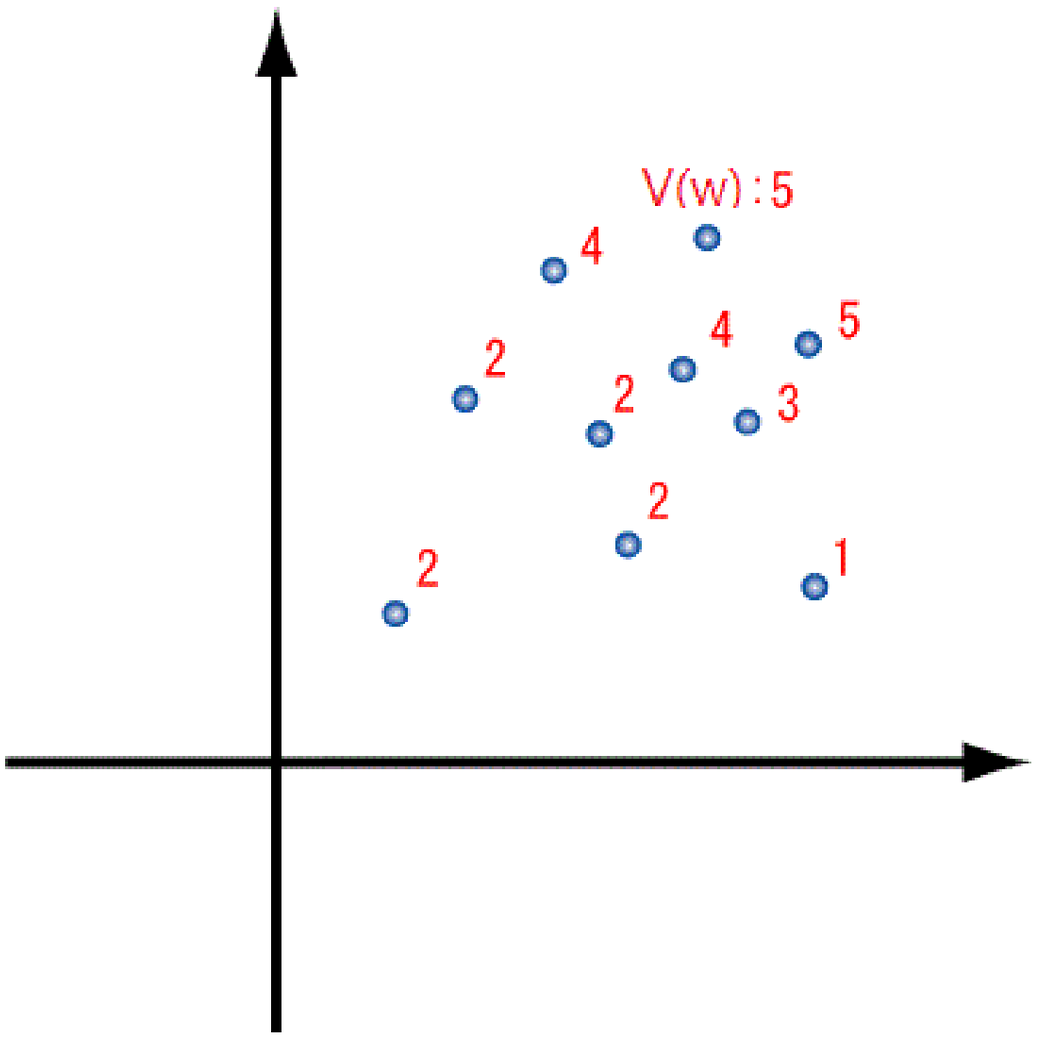
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



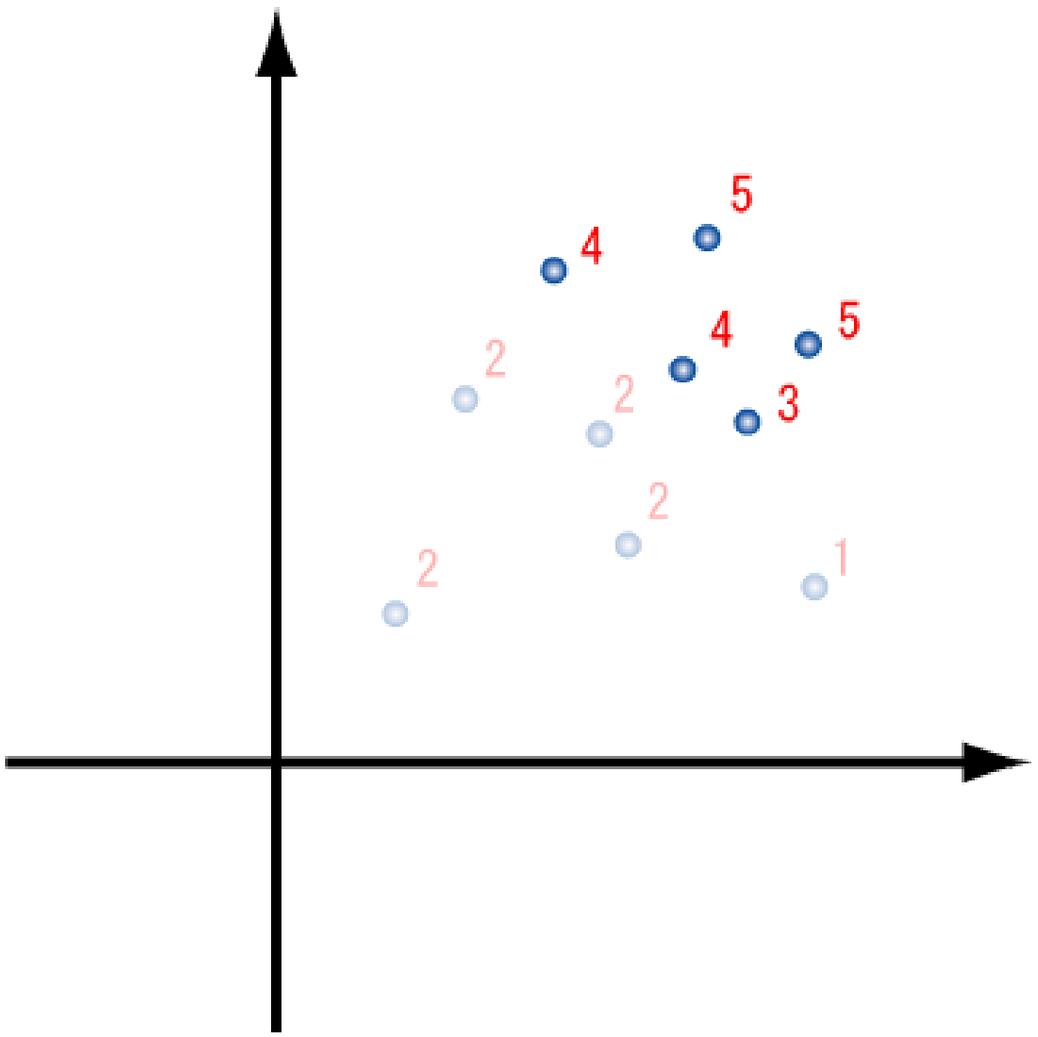
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



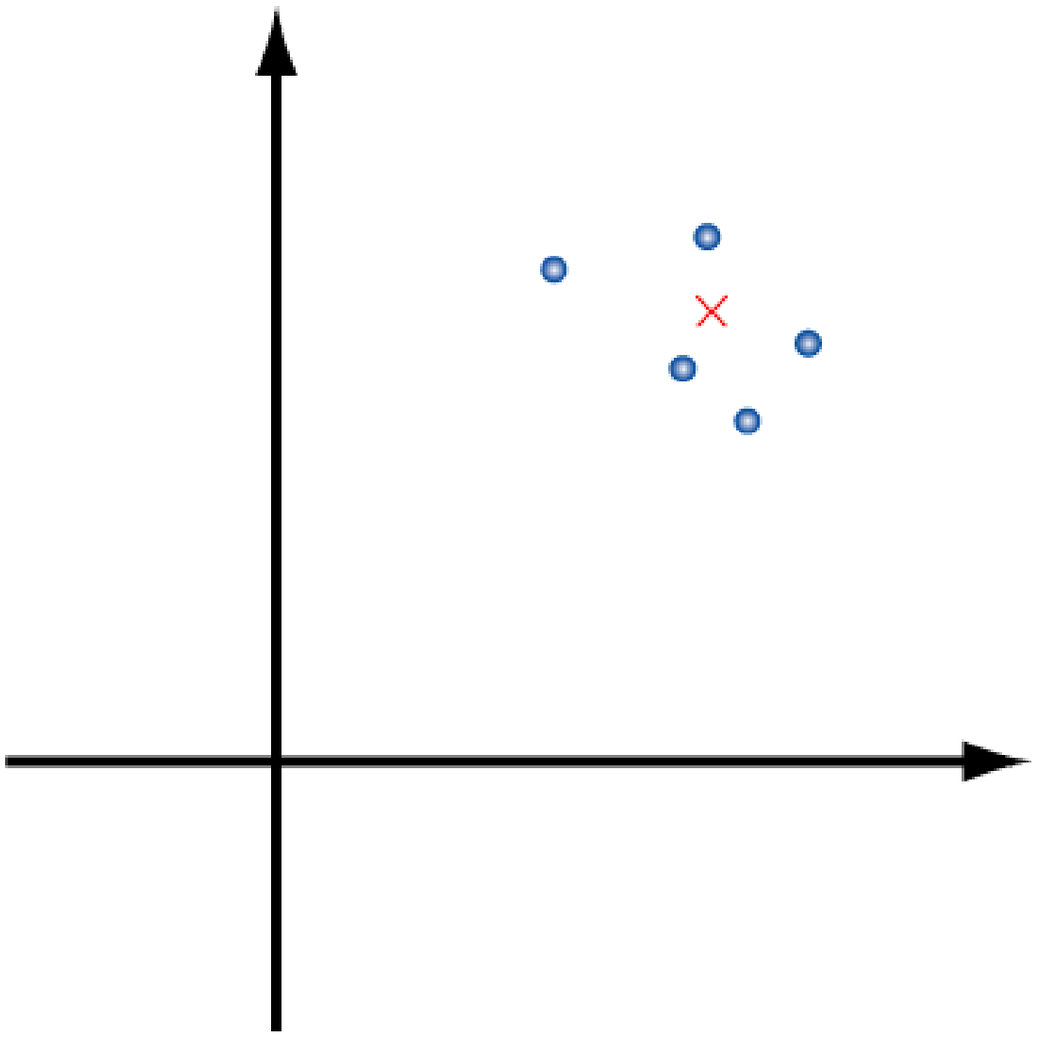
- Sample
- Select elites
- Update mean
- Update covariance
- iterate

Covariance Matrix Adaptation



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

Covariance Matrix Adaptation

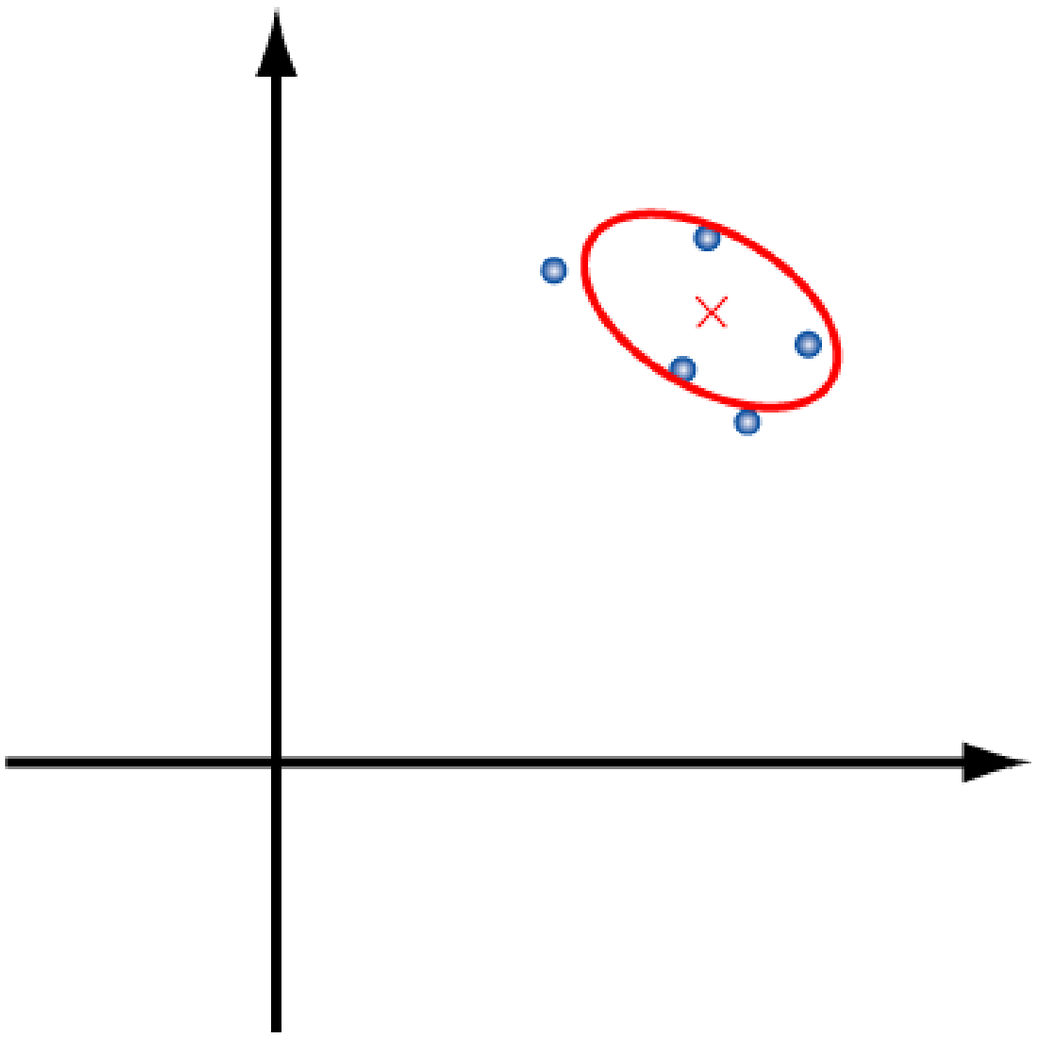


- Sample
- Select elites
- Update mean
- Update covariance
- iterate

$$\mu_{t+1} = \mu_t + \alpha \sum_{i=1}^{n_{elit}} w_i(\theta_i^{elit,t} - \mu_t)$$

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

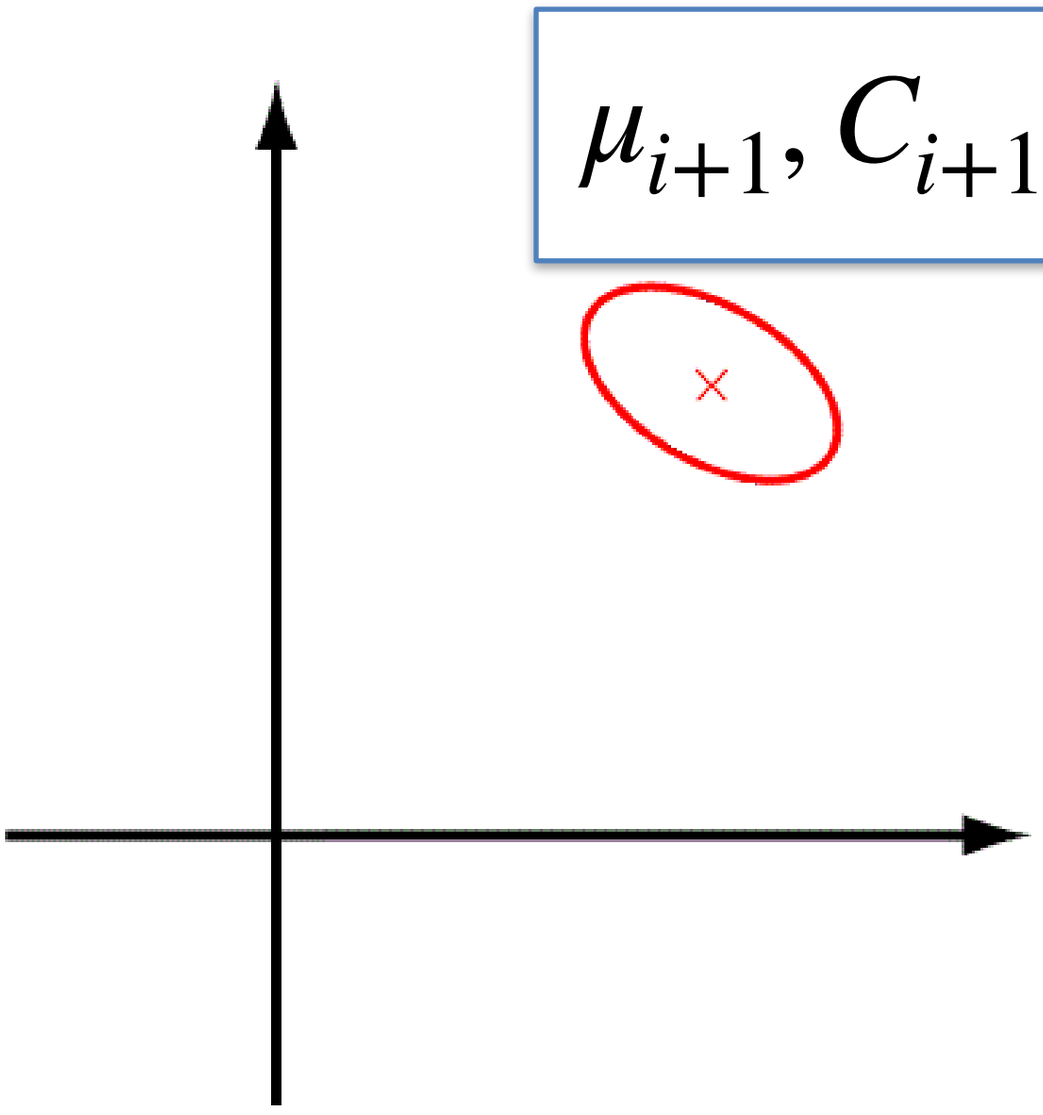
Covariance Matrix Adaptation



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

$$\Sigma_{t+1} = Cov(\theta_1^{elit,t}, \theta_2^{elit,t}, \dots) + \epsilon I$$

Covariance Matrix Adaptation



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

Natural Evolutionary Strategies (NES)

- In CEM and CMA-ES, we have been selecting the best (elite) parameter offsprings to update the parameter distribution.
- NES considers every offspring.

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diaginal and fixed covariance matrix $\sigma^2 \mathbf{I}_d : \theta \sim P_\mu(\theta)$.

- Goal: $\max_{\mu} . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diaginal and fixed covariance matrix $\sigma^2 \mathbf{I}_d : \theta \sim P_\mu(\theta)$.

- Goal: $\max_\mu . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

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

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diaginal and fixed covariance matrix $\sigma^2 \mathbf{I}_d : \theta \sim P_\mu(\theta)$.

- Goal: $\max_\mu . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

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

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diagonal and fixed covariance matrix $\sigma^2 \mathbf{I}_d$: $\theta \sim P_\mu(\theta)$.

- Goal: $\max_\mu . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

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

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diagonal and fixed covariance matrix $\sigma^2 \mathbf{I}_d$: $\theta \sim P_\mu(\theta)$.

- Goal: $\max_\mu . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

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

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diagonal and fixed covariance matrix $\sigma^2 \mathbf{I}_d$: $\theta \sim P_\mu(\theta)$.

- Goal: $\max_\mu . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

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

We approximate expectations by sampling!

Natural Evolutionary Strategies

- Consider the parameters of our policy $\theta \in \mathbb{R}^d$ follow a Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diagonal and fixed covariance matrix $\sigma^2 \mathbf{I}_d$: $\theta \sim P_\mu(\theta)$.

- Goal: $\max_\mu . \ \mathbb{E}_{\theta \sim P_\mu(\theta)} \mathrm{F}(\theta)$, where fitness score $\mathrm{F}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, s_0 \sim \mu_0(s)} R(\tau)$.

$$
\begin{aligned}
\nabla_\mu \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\mathrm{F}(\theta) \right] &= \nabla_\mu \int P_\mu(\theta) \mathrm{F}(\theta) d\theta \\
&= \int \nabla_\mu P_\mu(\theta) \mathrm{F}(\theta) d\theta \\
&= \int P_\mu(\theta) \frac{\nabla_\mu P_\mu(\theta)}{P_\mu(\theta)} \mathrm{F}(\theta) d\theta \\
&= \int P_\mu(\theta) \nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta) d\theta \\
&= \mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta) \right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \left[\nabla_\mu \log P_\mu(\theta) \mid_{\theta=\theta_i} \mathrm{F}(\theta_i) \right] \quad \theta_i \sim P_\mu(\theta)
\end{aligned}
$$

Reminder: Sampling from a multivariate Gaussian

We want to generate samples $\theta_i \sim P_\mu(\theta)$ where P is the Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and diagonal and fixed covariance matrix $\sigma^2 \mathbf{I}_d$.

Imagine we have access to random vectors $\epsilon_i \in \mathbb{R}^d, \epsilon_i \sim \mathcal{N}(0, \mathbf{I}_d)$

$$\theta_1 = \mu + \sigma\epsilon_1$$

$$\theta_2 = \mu + \sigma\epsilon_2$$

The 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 $\sigma^2 \mathbf{I}_d$.

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

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

A concrete example

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

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

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

  - We draw two parameter samples θ_1, θ_2. For each sampled parameter vector, we run the policy and obtain a set of trajectories, or a single trajectory. Then:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta) \right] \approx \frac{1}{N} \sum_{i=1}^{N} \left[\nabla_\mu \log P_\mu(\theta) \big|_{\theta=\theta_i} \mathrm{F}(\theta_i) \right]$$

A concrete example

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

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

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

  - We draw two parameter samples θ_1, θ_2. For each sampled parameter vector, we run the policy and obtain a set of trajectories, or a single trajectory. Then:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta) \right] \approx \frac{1}{N} \sum_{i=1}^{N} \left[\nabla_\mu \log P_\mu(\theta) \big|_{\theta=\theta_i} \mathrm{F}(\theta_i) \right]$$

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

  - Q:Can we simplify this expression more?

A concrete example

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

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

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

  - We draw two parameter samples θ_1, θ_2. For each sampled parameter vector, we run the policy and obtain a set of trajectories, or a single trajectory. Then:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)}\left[\nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\left[\nabla_\mu \log P_\mu(\theta)\,|_{\theta=\theta_i} \mathrm{F}(\theta_i)\right]$$

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

$$\boxed{\begin{aligned}\theta_1 &= \mu + \sigma\epsilon_1 \\ \theta_2 &= \mu + \sigma\epsilon_2\end{aligned}}$$

A concrete example

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

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

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

  - We draw two parameter samples θ_1, θ_2. For each sampled parameter vector, we run the policy and obtain a set of trajectories, or a single trajectory. Then:

$$\mathbb{E}_{\theta \sim P_\mu(\theta)} \left[\nabla_\mu \log P_\mu(\theta) \mathrm{F}(\theta) \right] \approx \frac{1}{N} \sum_{i=1}^{N} \left[\nabla_\mu \log P_\mu(\theta) \big|_{\theta=\theta_i} \mathrm{F}(\theta_i) \right]$$

$$\boxed{\begin{aligned} \theta_1 &= \mu + \sigma\epsilon_1 \\ \theta_2 &= \mu + \sigma\epsilon_2 \end{aligned}}$$

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

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

Natural Evolutionary Strategies (NES)

- In CEM and CMA-ES, we have been selecting the best (elite) parameter offsprings.
- NES considers every offspring.

Algorithm 1: Evolutionary Strategies

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

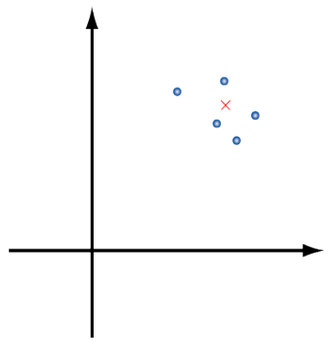
6. end for

Compare the two update rules for the mean

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

All offsprings participate

CMA-ES: $$\mu_{t+1} = \mu_t + \alpha \sum_{i=1}^{n_{elit}} w_i(\theta_i^{elit,t} - \mu_t)$$



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

Question: Can evolutionary methods scale?

- Evolutionary methods work well on relatively low-dimensional problems (small number of parameter dimensions).
- Can they be used to optimize policies represented as neural networks of thousands parameters?

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

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

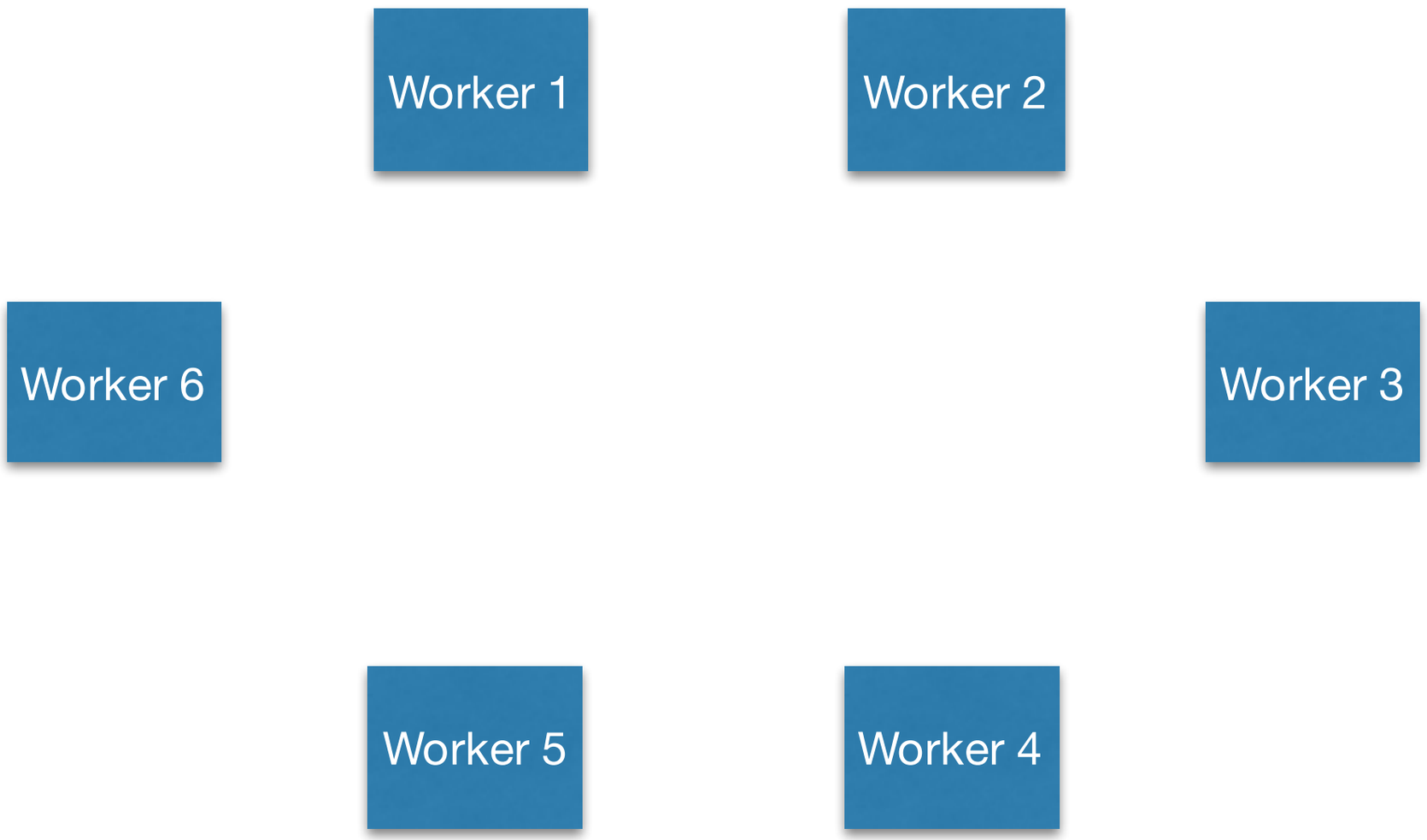
Algorithm 1: Evolutionary Strategies

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

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

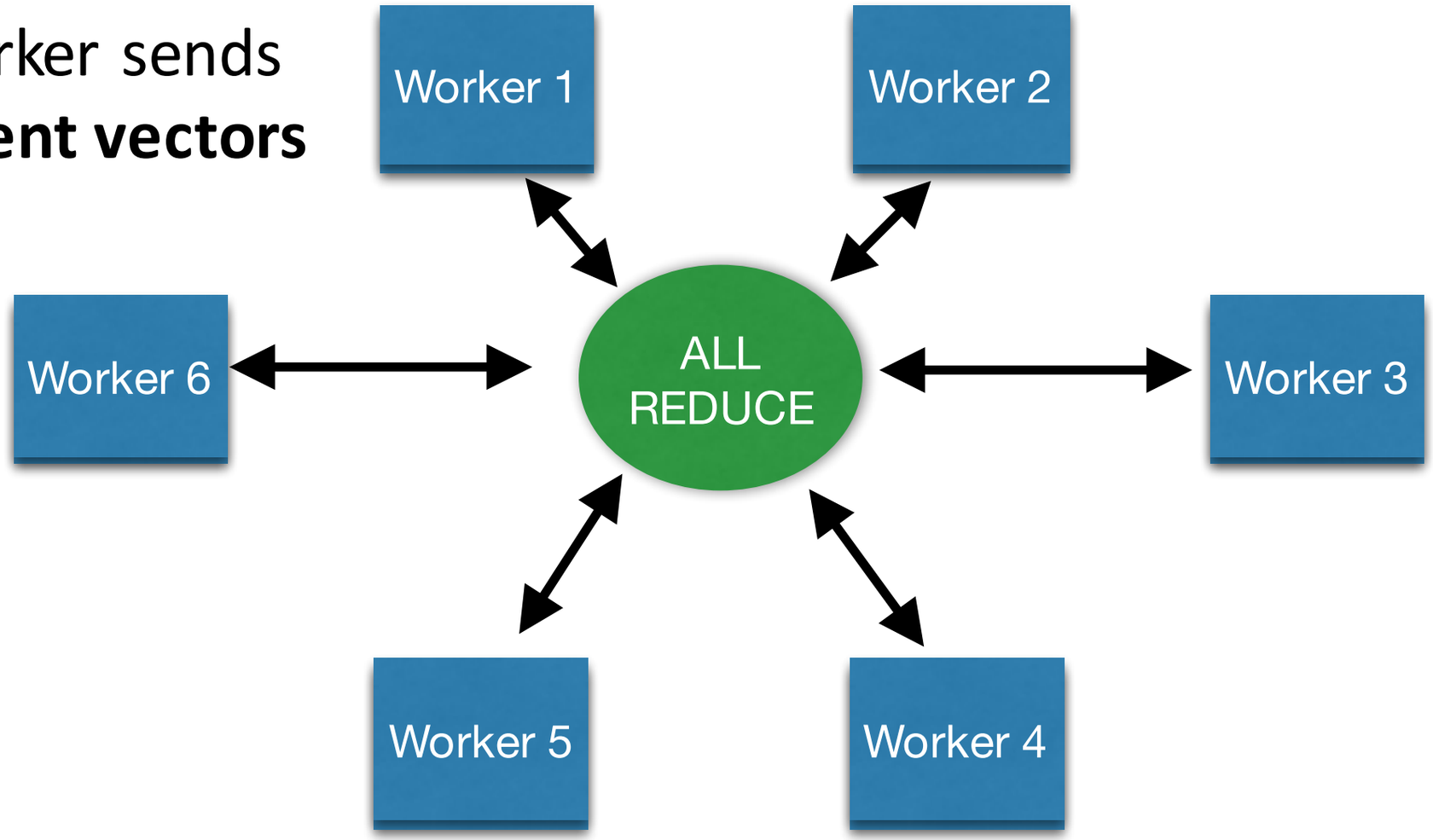
Distributed stochastic gradient descent

Every worker get a set of examples, and computes a gradient vector.
The master averages those gradient vectors.

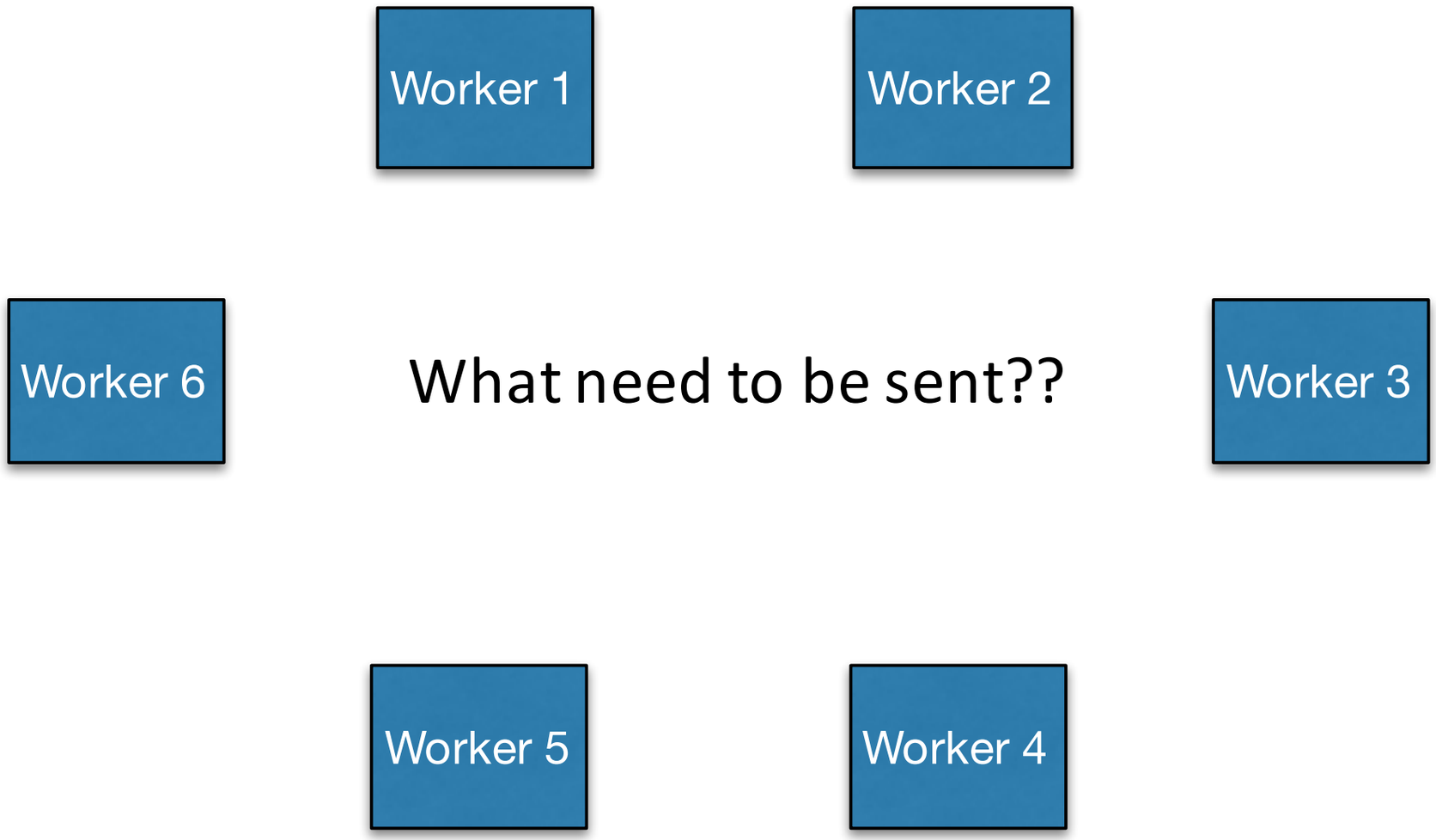


Distributed stochastic gradient descent

Each worker sends
big gradient vectors



Distributed Evolution



Distributed Evolution

Worker 1

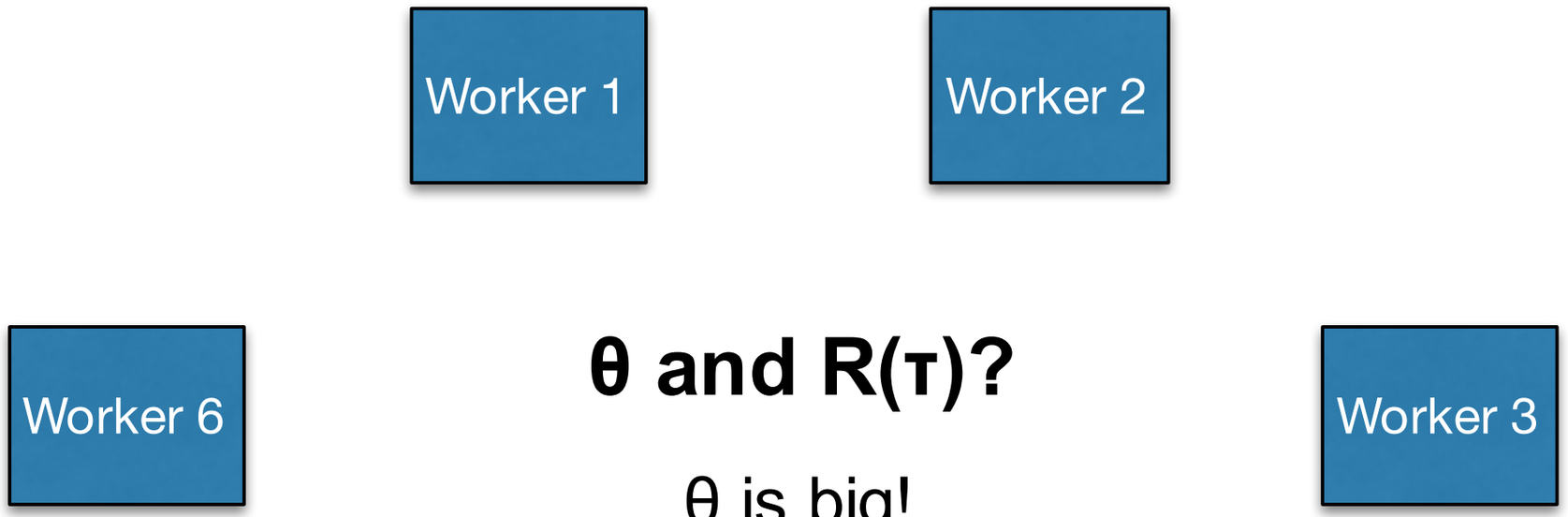
Worker 2

Algorithm 1: Evolutionary Strategies

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

6. end for

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

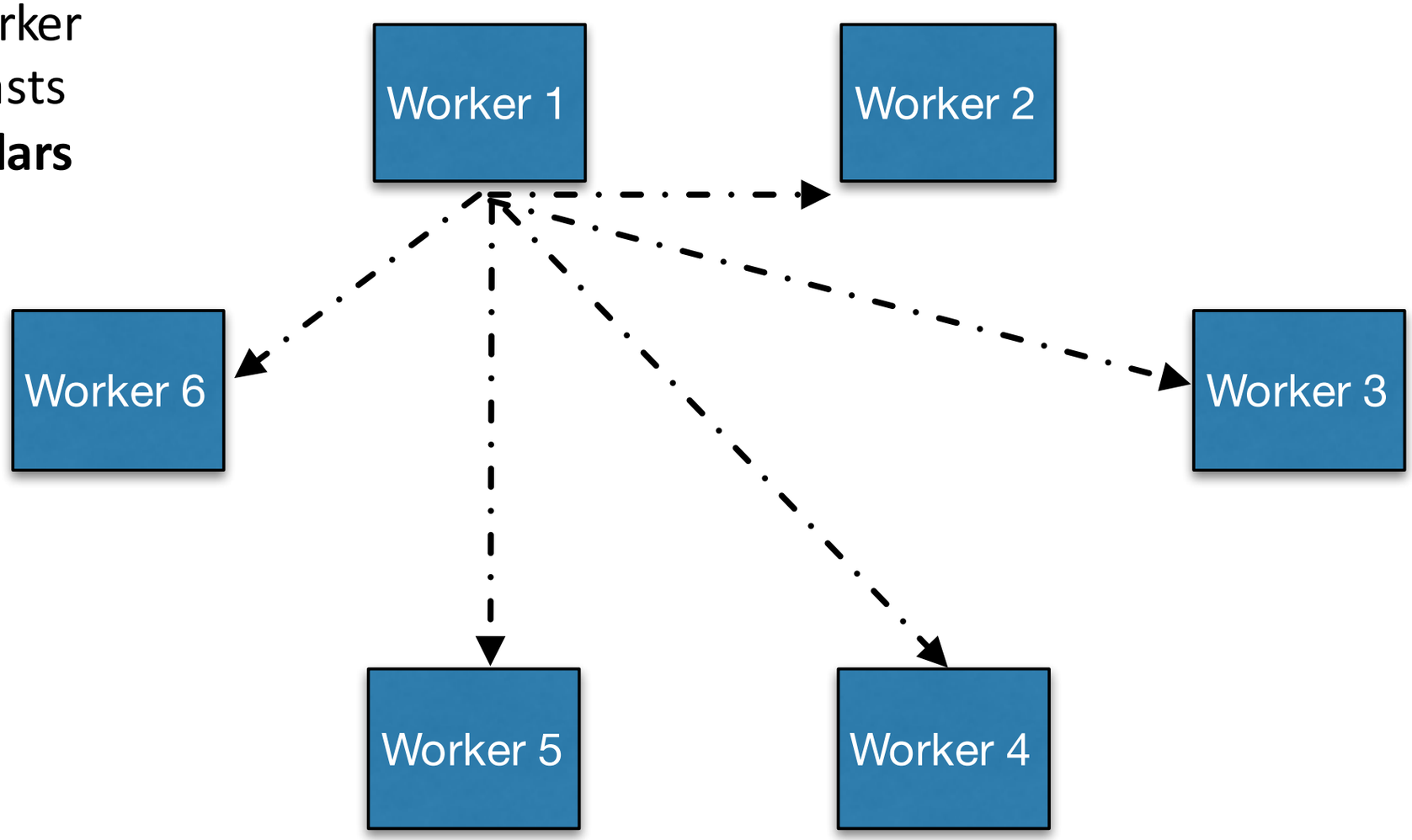
Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with 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 $F_i = F(\mu_t + \sigma\epsilon_i)$
  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]

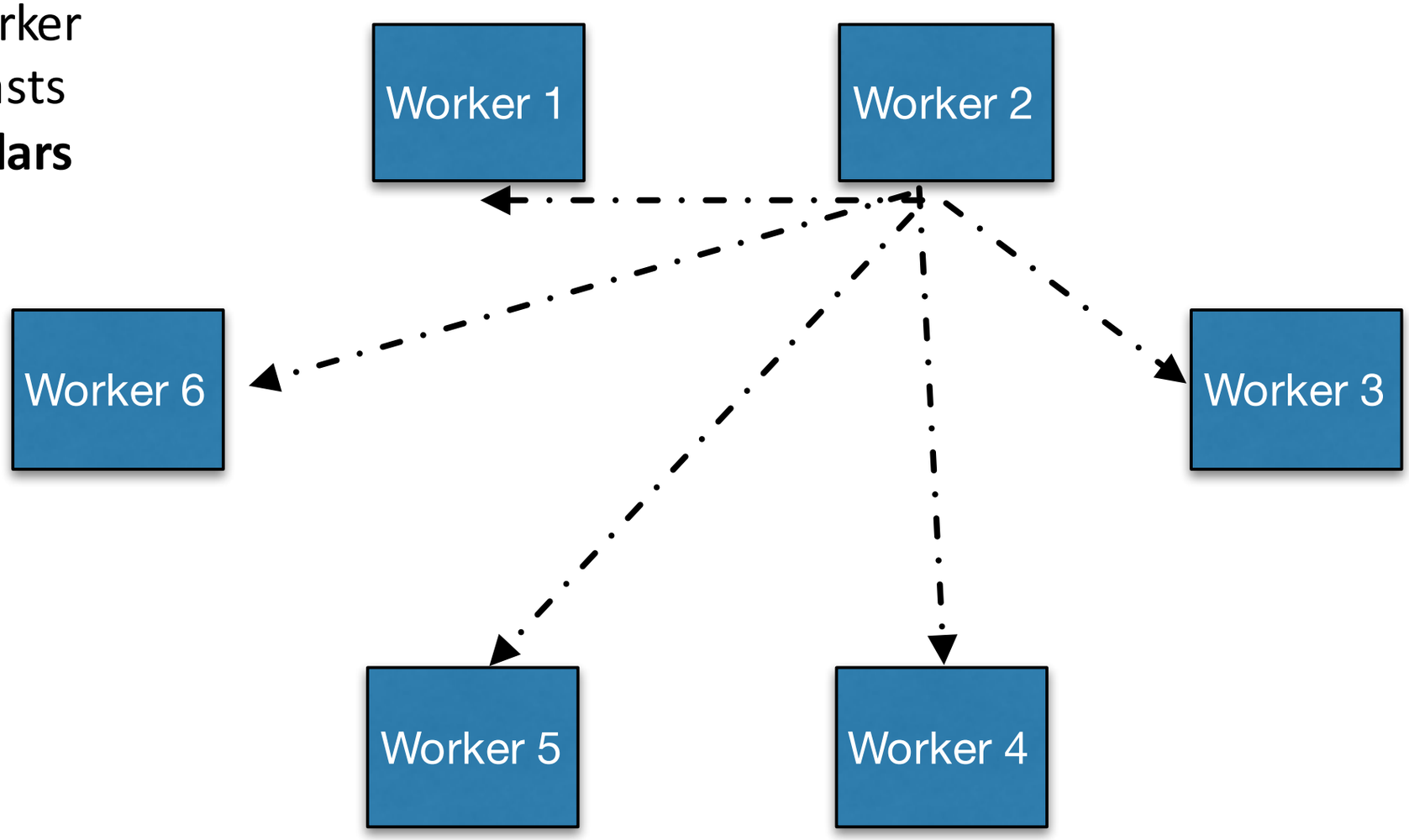
Distributed Evolution

Each worker broadcasts **tiny scalars**



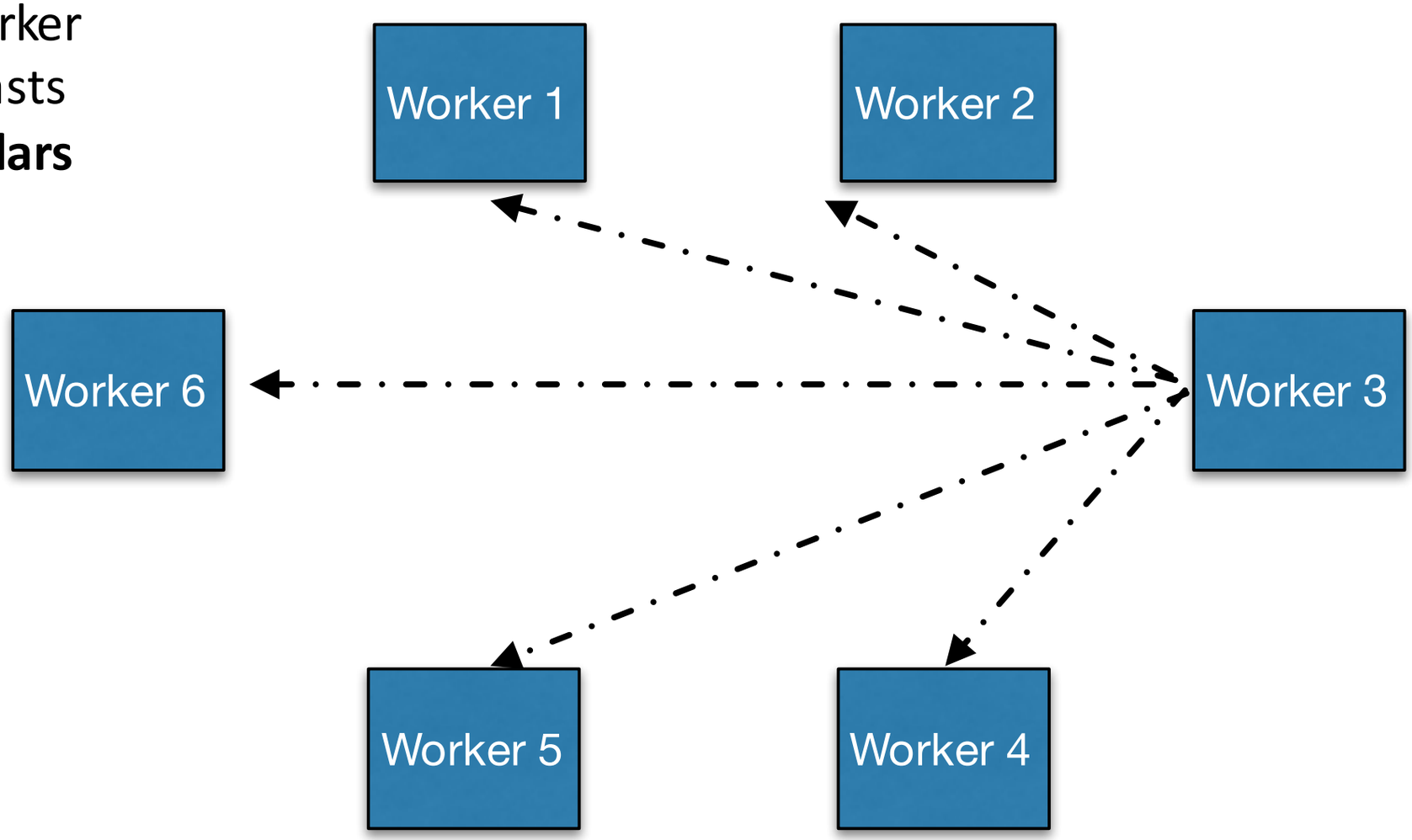
Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution

Each worker broadcasts **tiny scalars**



Distributed Evolution Scales Very Well :-)

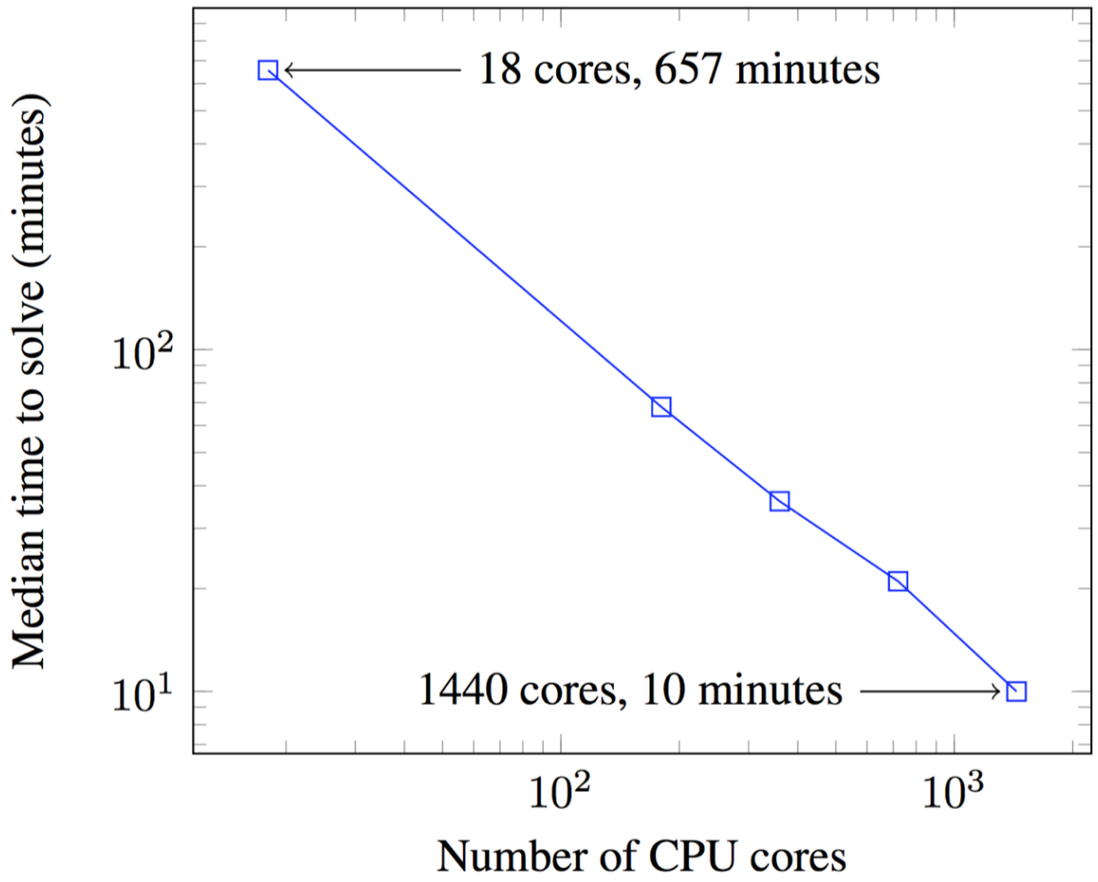


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

Salimans, Ho, Chen, Sustskever. 2017

Conclusions

- 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 the ES baseline for the same amount of resources.