Evolutionary Algorithms

Machine Learning, Fall 2018

Jürgen Schmidhuber, Michael Wand, **Raoul Malm**

TAs: Robert Csordas, Aleksandar Stanic,
Xingdong Zuo, Francesco Faccio

slides by Raoul Malm

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

Holland's Schema Theorem for GA

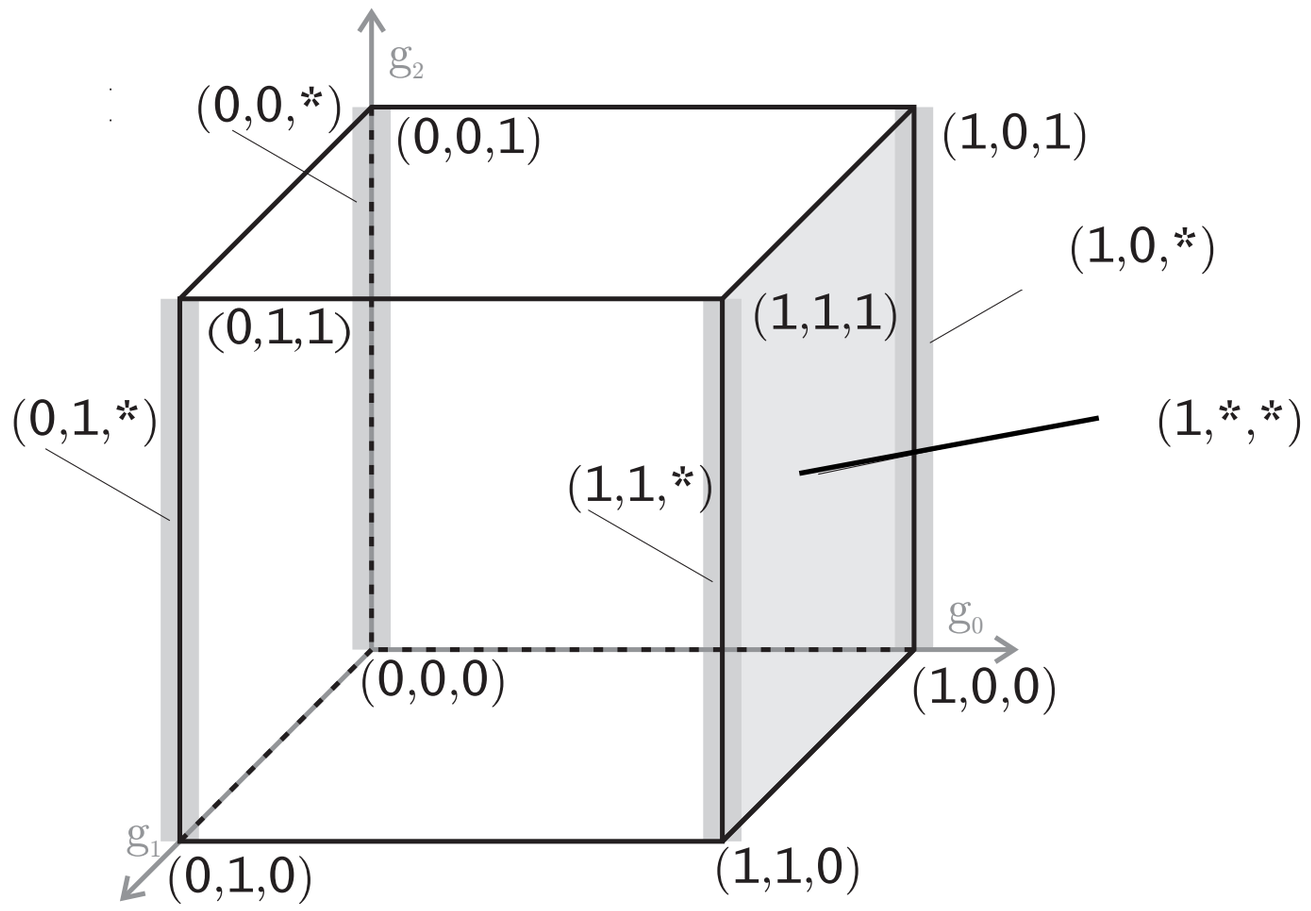
Holland's Schema Theorem

- Fundamental theorem of genetic algorithms
- Goal: provide a formal model for the effectiveness of the GA search process
- Assumptions for the GA
  - binary alphabet: 0,1
  - chromosomes of fixed length (bit strings)
  - fitness proportionate selection
  - recombination by single point crossover
  - gene-wise mutation (bit flipping)

Some details taken from
[T. M. Mitchell: Machine Learning]
[Weise: Global Optimisation Algorithm - Theory and Application]

A Schema

- A schema is a template that identifies a subset of strings with similarities at certain positions
- * is a wildcard symbol, which means that it can take either value 0 or 1 (don't care)
- Example (length 3): schema [1 * *] implies the individuals
  - [1 0 0]
  - [1 1 0]
  - [1 0 1]
  - [1 1 1]
- a schema like [* 0 *] contains less information than [0 0 *]
- there are 3^L possible schemas for bit strings of length L



[Weise: Global Optimisation Algorithm - Theory and Application]

Notation and Definitions

- l, fixed length of each individual
- P_t, population at time t of size n (containing n individuals)
- $o(s)$, the order of s, is the number of non "*" genes in schema s, e.g.

$$o([***11*0]) = 3$$

- $d(s)$, schema defining length, is the distance between first and last non "*" genes in schema s, e.g.

$$d([***11*0]) = 7 - 4 = 3$$

- $f(h)$, fitness score of some individual $h \in P_t$
- $m(s, t)$, number of instances of schema s in the population at time t

Derivation

- What is the goal ? Let us consider one particular schema s. We want to know if we start with m instances of schema s in a population at time t, how many instances of schema s survived in the next generation at time $t+1$?
- What is the probability of selecting $h \in P_t$ via fitness proportional selection ? It is

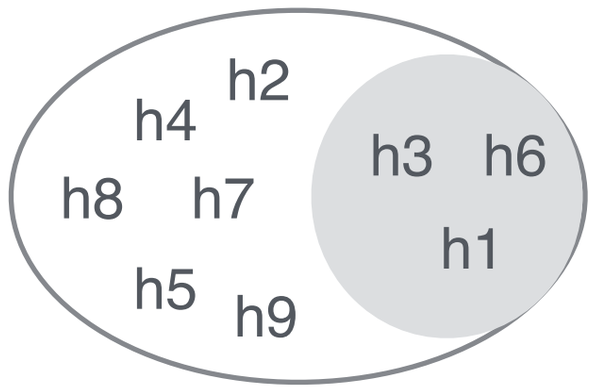
$$p(h) = \frac{f(h)}{\sum_{h' \in P_t} f(h')} = \frac{f(h)}{n\,\bar{f}(t)}\,, \tag{1}$$

where $\bar{f}(t)$ is the mean fitness of the entire population at time t.

- Suppose we have selected some arbitrary $h \in P_t$ according to (1), what is the probability that h is an instance of some schema s ? It is

$$p(h \in s) = \sum_{h \in s \cap P_t} p(h) = \frac{\sum_{h \in s \cap P_t} f(h)}{n\,\bar{f}(t)} = \frac{\bar{f}(s,t)}{\bar{f}(t)} \frac{m(s,t)}{n}\,,$$

where $\bar{f}(s,t)$ is the average fitness of schema s at time t, and $m(s,t)$ is the number of instances of schema s in the population at time t.



Derivation

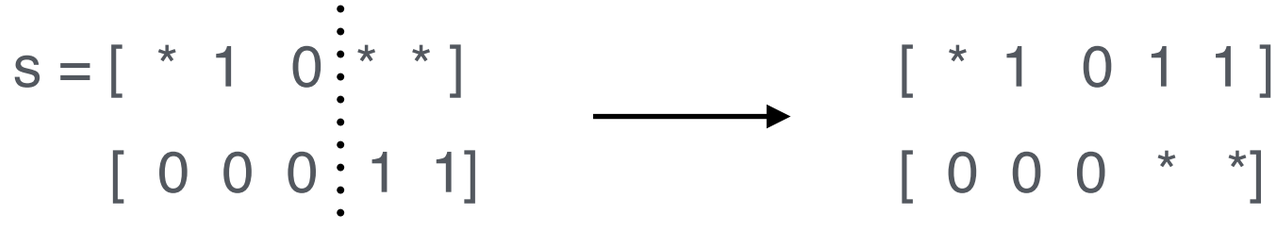
- Actually, we are interested in the expected number of instances of schema s resulting from n independent selection steps (with replacement) that create the entire generation at time step $t + 1$:

$$\mathbb{E}[m(s, t + 1)] = n\, p(h \in s) = \frac{\bar{f}(s, t)}{\bar{f}(t)}\, m(s, t)\,.$$

  Thus, we can expect schemas with above average fitness to be represented with increasing frequency in subsequent generations.
- But wait, what about crossover and mutations ?

Crossover

- Let us denote with p_c the probability that the single-point crossover operator will be applied to an arbitrary individual.
- Note, that even if the single-point crossover is applied on a parent belonging to schema s it does not necessarily mean that the child is not anymore part of the same schema s, e.g.



- In fact, the probability that the single-point crossover operator will be applied and cuts the schema s is given by

$$p_c \frac{d(s)}{l-1}.$$

e.g. d([* 1 0 * *])/(l-1) = 1/4

Schema with long defining length are more likely to be disrupted by single point crossover than schema using short defining lengths.

- Consequently, the probability that the schema s survives the crossover step is

$$\left(1 - p_c \frac{d(s)}{l-1}\right).$$

Mutation

- Mutation is applied gene by gene.
- Let us denote with p_m the probability that an arbitrary bit of an arbitrary individual will be mutated by the mutation operator.
- In order for schema s to survive, all non "*" genes in the schema must remain unchanged
- When the "*" positions of a schema instance are mutated, the instance still belongs to the same schema s
- Thus, the probability that a given instance of schema s will still belong to schema s after the mutation step is given by

$$(1 - p_m)^{o(s)}$$

Holland's Schema Theorem

- Finally, the expected number of instances of schema s resulting from n independent selection steps (with replacement) that create the entire generation at time step $t+1$ is given by

$$\mathbb{E}[m(s,t+1)] \geq \underbrace{\frac{\bar{f}(s,t)}{\bar{f}(t)}\left(1-p_c\frac{d(s)}{l-1}\right)(1-p_m)^{o(s)}}_{\alpha(s,t)} m(s,t)\,. \qquad (1)$$

  This lower bound means that a schema s with $\alpha(s,t) > 1$ will on average increase its instances in the next generation.

- how to achieve $\alpha(s,t) > 1$?
  - s must have above average fitness
  - defined bits in s should be grouped, i.e. small schema-defining length $d(s)$
  - s should contain a large number of "*" symbols, i.e. small order $o(s)$
- "The Schema Theorem says that short, low-order schemata with above-average fitness increase in frequency in successive generations"
- Note, (1) neglects the probability that a string belonging to the schema s will be created "from scratch" by mutation of a single string (or crossover of two strings) that did not belong to s in the previous generation. Therefore, (1) is an inequality.

Holland's Schema Theorem

Limitations

- The theorem fails to consider the positive effects of crossover and mutation
- The theorem makes no statement about that converging to a global optimum solution
- The theorem is described in terms of expectation, thus it is only valid if you perform an infinite number of "experiments" or if you let the population size increase to infinity. Otherwise the theorem is affected by sampling errors.
- The theorem makes only a statement about one generation step. Note, that the average-fitness of the population and of the schema is time-dependent, i.e. their ratio is not static (population drift). Consequently, the theorem cannot be generalised to multiple time steps.

[some interesting read on the crossover-mutation debate:
https://www-cs.stanford.edu/people/nuwans/docs/GA.pdf]

Evolution Strategies

Evolution Strategies (ES)

- Note: this comparison is historical, nowadays properties can overlap for both types

Properties	**ES**	**GA**
objective parameters “genes”	real values	binary values
mutation	Gaussian noise	bit-flip
parent selection	uniform random	various methods
recombination	averaging	crossover
what is evolved	objective parameters and mutation parameters	objective parameters

- Key properties of Evolution Strategies
  - recombination is done by averaging the parameters of parents
  - mutation parameters can evolve (self-adapt their values)
  - mutation is done by adding normally distributed values

Self-Adaptation (μ/ρ+,λ) ES Algorithm

- Each candidate **a** now contains objective (like in GA) and mutation (strategy) parameters (d is the dimensionality of the search space):

$$\mathbf{a} = (\quad \underbrace{\theta_1, \theta_2, ..., \theta_d}_{\text{objective parameters } \boldsymbol{\theta}} \quad , \quad \underbrace{\sigma_1, \sigma_2, ..., \sigma_d}_{\text{mutation parameters } \boldsymbol{\sigma}} \quad)$$

  The mutation parameter determines the amount of mutation which is applied to the corresponding objective parameter.

- the meta-parameters of ES are described by the $(\mu/\rho+, \lambda)$ notation
  - μ is the number of parents in the parent population $P_\mu = \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_\mu\}$
  - ρ is the number of parents that are averaged to create one offspring
  - λ is the number of offspring in the offspring population $P'_\lambda = \{\mathbf{a}'_1, \mathbf{a}'_2, ..., \mathbf{a}'_\lambda\}$
  - "," : select new parents for next generation from offspring (requires $\lambda \geq \mu$)
  - "$+$" : select new parents for next generation from offspring and previous parents
- simple examples: population with one individual
  - $(1/1, 1)$: next generation always contains the offspring from the parent
  - $(1/1+1)$: next generation contains the more fit candidate (offspring or parent)

Self-Adaptation (μ/ρ+,λ) ES Algorithm

Algorithm 1: $(\mu/\rho \overset{+}{,} \lambda)$ Self-Adaptation Evolution Strategy Algorithm

initialise the first generation $P_\mu^{g=1} = \{\mathbf{a}_1^{g=1}, \mathbf{a}_2^{g=1}, ..., \mathbf{a}_\mu^{g=1}\}$ and $P_\lambda = \{\}$
for *each generation* $g = 2, 3, ..., G$ **do**
  for *each offspring* $(1, 2, ..., \lambda)$ **do**
    select uniform randomly ρ parents from P_μ^{g-1}
    average the selected parents to form the candidate $\mathbf{a} = (\boldsymbol{\theta}, \boldsymbol{\sigma})$
    adapt the mutation parameter $\boldsymbol{\sigma}$ yielding the new value $\boldsymbol{\sigma}'$
    mutate the objective parameter $\boldsymbol{\theta}$ using $\boldsymbol{\sigma}'$ yielding the new value $\boldsymbol{\theta}'$
    add new offspring $(\boldsymbol{\theta}', \boldsymbol{\sigma}')$ to offspring population P_λ
  end
  select μ candidates for next generation P_μ^g via truncated selection from either
    - the offspring population P_λ ("$\mu/\rho, \lambda$" comma selection)
    - the offsprings P_λ and parents P_μ^{g-1} ("$\mu/\rho + \lambda$" plus selection)
  reset offspring population $P_\lambda = \{\}$
end

Uncorrelated Mutation: Case 1

Uncorrelated mutation with 1 mutation parameter

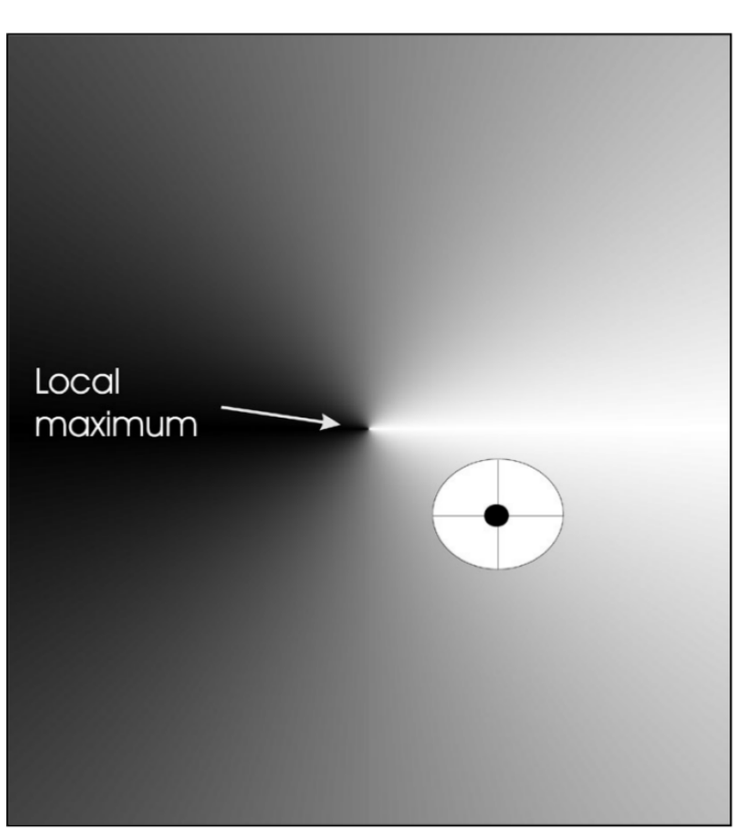
- look at one mutation step of the (parent average) candidate $(\boldsymbol{\theta}, \sigma)$

$$\sigma' = \sigma \exp(\epsilon)\,, \qquad \epsilon \sim N(0, \tau)$$
$$\theta_i' = \theta_i + \epsilon_i\,, \qquad \epsilon_i \sim N(0, \sigma'),\ i = 1, ..., d$$

- τ is the learning rate, typically $\tau \sim 1/d^{1/2}$
- we impose the boundary rule that if $\sigma' < \epsilon$ then $\sigma' = \epsilon$, where ϵ is some fixed threshold
- changing the mutation strength σ allows for a self-tuning of the mutation strength (σ self-adaptation)

isotropically distributed mutations

- mutants on the circle have the same probability of being created from the parent in the centre



[Eiben, Smith: Introduction to Evolutionary Computation]

Uncorrelated Mutation: Case 2

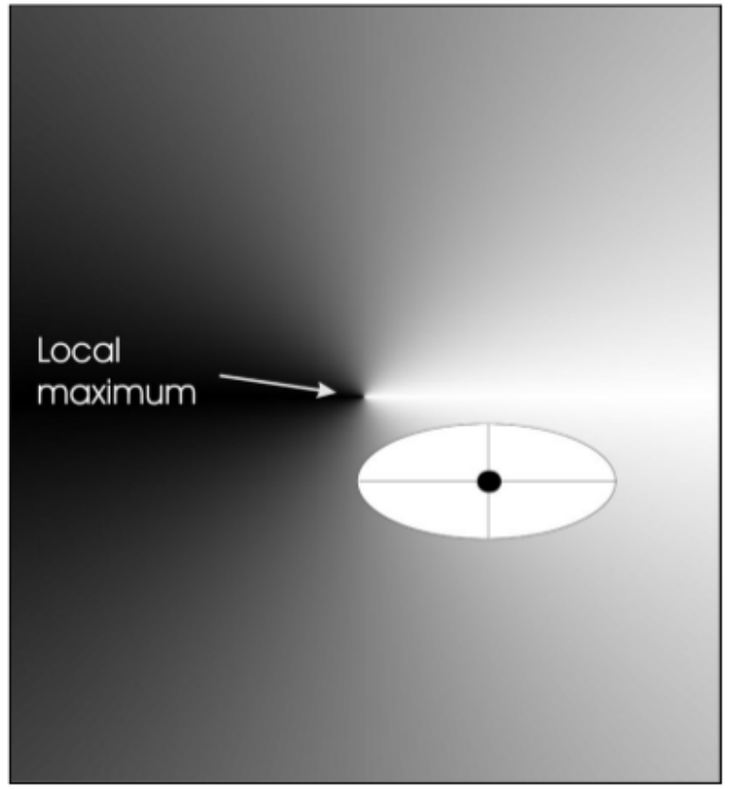
Uncorrelated mutation with d mutation parameters

- look at one mutation step of the (parent average) candidate $(\boldsymbol{\theta}, \boldsymbol{\sigma})$

$$\sigma_i' = \sigma_i \exp(\epsilon + \epsilon_i') \,, \qquad \epsilon \sim N(0, \tau) \,, \epsilon_i' \sim N(0, \tau'),$$
$$\theta_i' = \theta_i + \epsilon_i \,, \qquad \epsilon_i \sim N(0, \sigma_i'),\ i = 1, ..., d$$

- two learning rates
  - overall learning rate $\tau \sim 1/d^{1/2}$
  - coordinate-wise learning rate $\tau' \sim 1/d^{1/4}$
- we impose the boundary rule that if $\sigma' < \epsilon$ then $\sigma' = \epsilon$, where ϵ is some fixed threshold
- probability of mutation varies along the coordinates of the $\boldsymbol{\theta}$ space

mutants on the ellipse have the same probability of being created from the parent in the centre



[Eiben, Smith: Introduction to Evolutionary Computation]

Correlated Mutation

Correlated mutation with a $d \times d$ covariance matrix $\mathbf{C}$

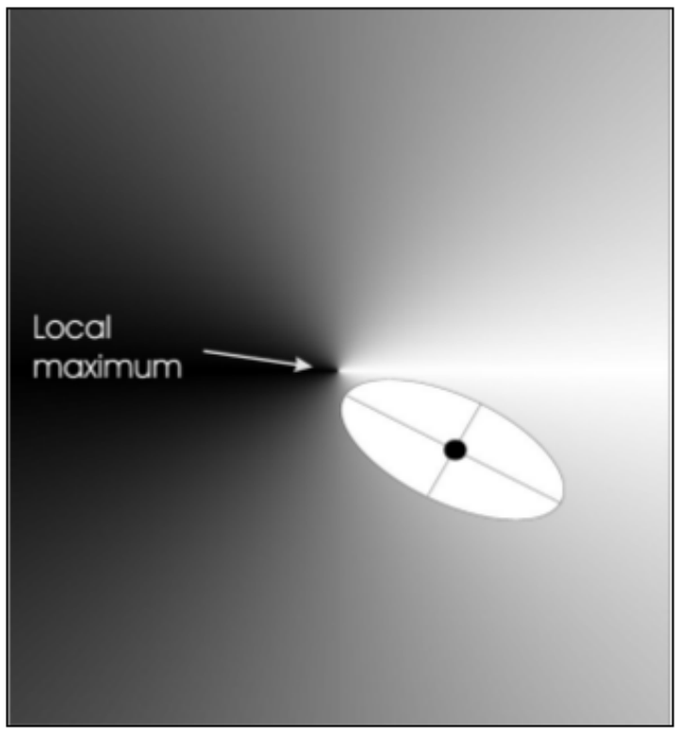
- $\mathbf{C}$ is a real, symmetric, positive semi-definite matrix: $\mathbf{C} = \mathbf{R}\,\mathbf{D}\,\mathbf{R}^T$ with diagonal positive semi-definite matrix $\mathbf{D}$ and orthogonal matrix $\mathbf{R}$
- $\mathbf{C}$ can be parametrised by d standard deviations $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d)$ and $k = d(d-1)/2$ rotation angles $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_k)$
- look at one mutation step for the candidate $(\boldsymbol{\theta}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$

$$\sigma_i' = \sigma_i \exp(\epsilon + \epsilon_i')\,, \qquad \epsilon \sim N(0, \tau),\ \epsilon_i' \sim N(0, \tau')$$
$$\alpha_i' = \alpha_i + \epsilon_i\,, \qquad \epsilon_i \sim N(0, \tau''),\ i = 1, ..., d$$

- we impose the same boundary rule as before
- two learning rates as before (τ and τ') and additionally τ''
- create $\mathbf{C}'$ from $\boldsymbol{\sigma}', \boldsymbol{\alpha}'$ and mutate objective parameters

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\epsilon}\,, \qquad \boldsymbol{\epsilon} \sim N(0, \mathbf{C}')$$

- probability of mutation (variance) can vary along any direction in the $\boldsymbol{\theta}$ space



[Eiben, Smith: Introduction to Evolutionary Computation]

ES Examples

Covariance Matrix Adaptation (CMA) - ES

- one of the most popular gradient-free optimisation algorithms
- useful in particular on non-convex, non-separable, ill-conditioned, multi-modal or noisy objective functions
- considers the **mean** and **covariance** of the current candidates in the population and adapts them in direction towards the fittest candidates for the next generation
- belongs to the class of **(μ/μ_w, λ) ES** algorithms where all parents are averaged (weighted) before computing offsprings by mutation
- can be used when the search space dimensionality is less than ~1000 due to O(d^2) behaviour

[https://arxiv.org/abs/1604.00772]

CMA-ES

Algorithm 1: Basic idea behind CMA-ES, see [arXiv:1604.00772] for details

1 initialise mean $\mathbf{m}^{(0)}$ and covariance matrix $\mathbf{C}^{(0)} = \sigma\,\mathbf{1}$

2 **for** *each generation* $g = 1, 2, ..., G$ **do**

3 sample λ offsprings

$$\boldsymbol{\theta}_i^{(g)} \sim \mathbf{m}^{(g-1)} + N(0, \mathbf{C}^{(g-1)}) \quad \text{for} \quad i = 1, 2, ..., \lambda$$

4 select parents via truncated selection and average them

$$\mathbf{m}^{(g)} = \frac{1}{\mu} \sum_{i=1}^{\mu} \boldsymbol{\theta}_{i:\lambda}^{(g)} \,,$$

where $\boldsymbol{\theta}_{i:\lambda}^{(g)}$ denotes the i^{th}-fittest offspring of generation g.

5 adapt the covariance matrix

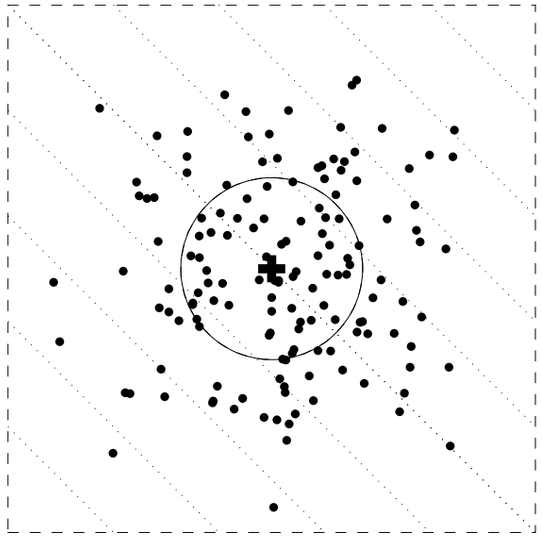
$$\mathbf{C}^{(g)} = \frac{1}{\mu} \sum_{i=1}^{\mu} (\boldsymbol{\theta}_{i:\lambda}^{(g)} - \mathbf{m}^{(g-1)}) \, (\boldsymbol{\theta}_{i:\lambda}^{(g)} - \mathbf{m}^{(g-1)})^T$$

use true mean !!!

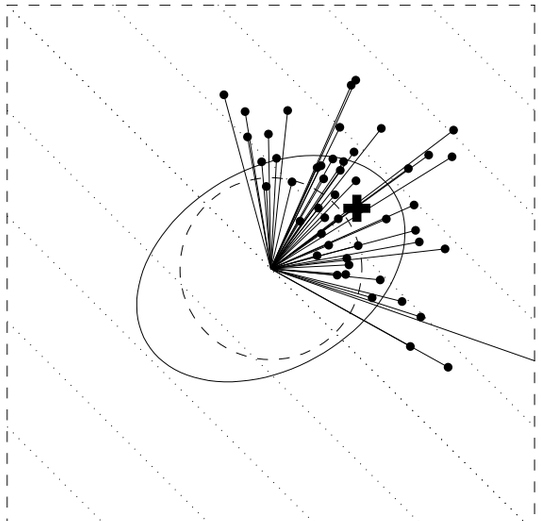
6 **end**

CMA-ES

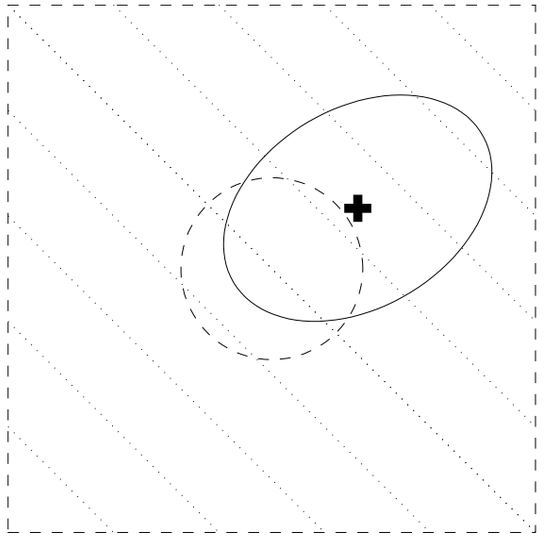
How covariance adaptation works [https://arxiv.org/abs/1604.00772]



sampling



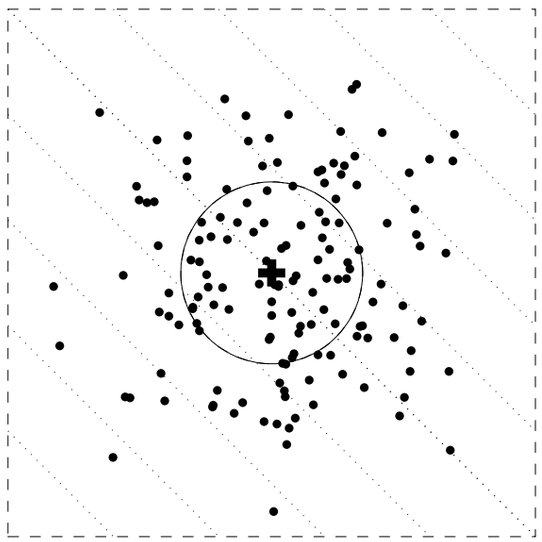
estimates variance of the selected steps



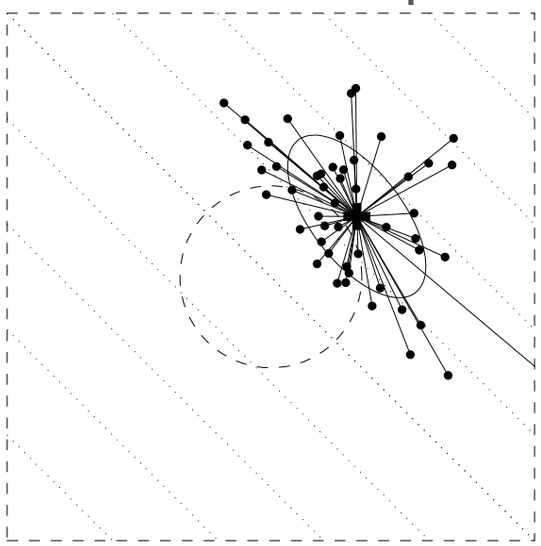
new covariance matrix

$C_{\mu}^{(g+1)}$

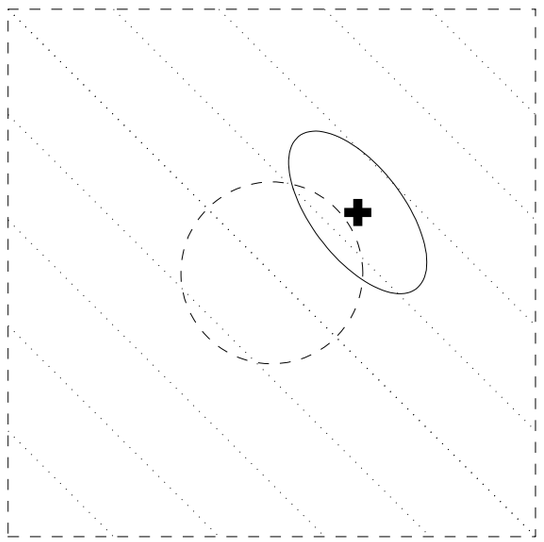
(used)



sampling



estimates variance within the selected population

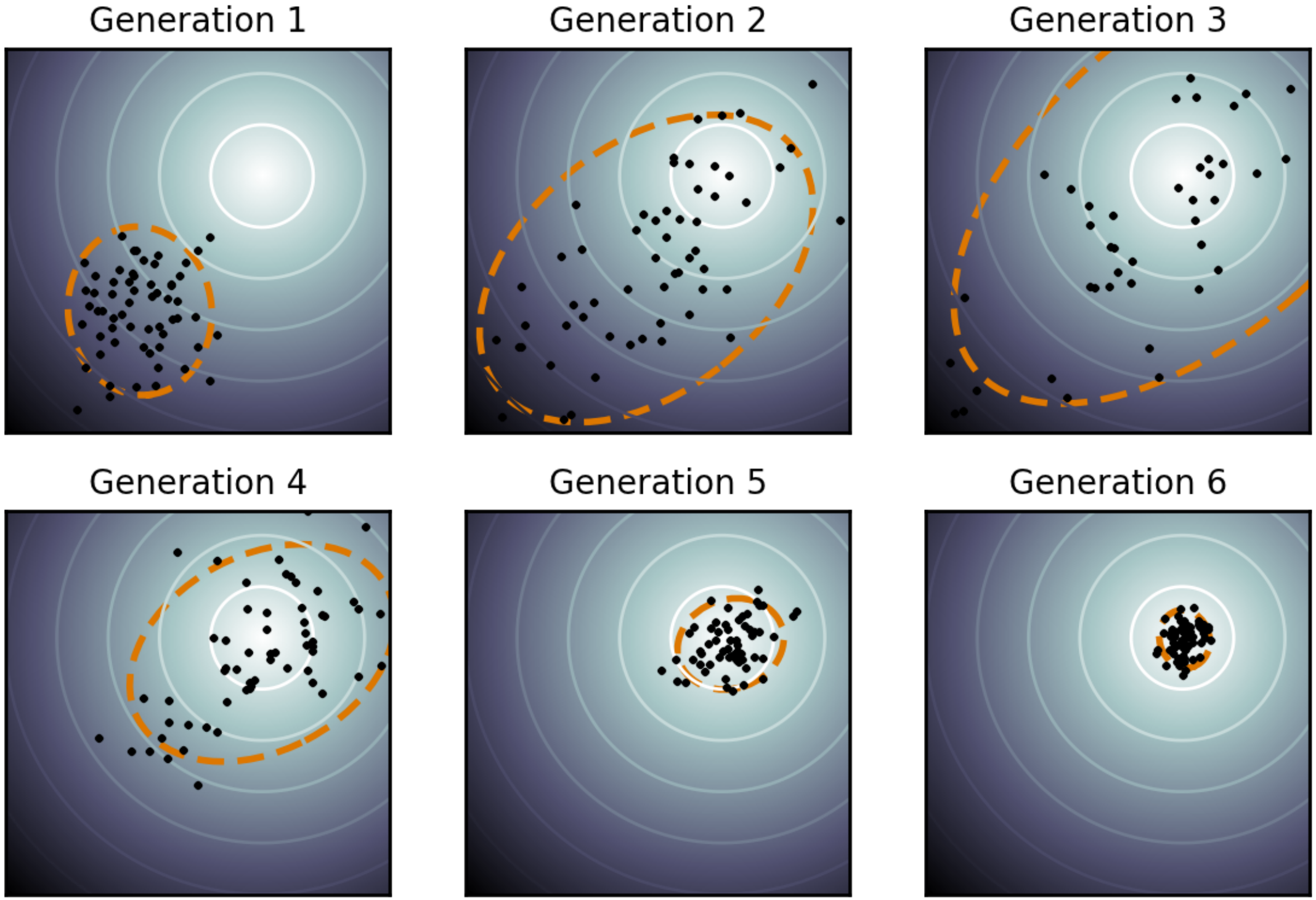


new covariance matrix

$C_{\mathrm{EMNA}_{global}}^{(g+1)}$

(not used)

CMA-ES



[https://en.wikipedia.org/wiki/CMA-ES]

ES with a Search Distribution

- So far, we have always discarded some fraction of the offspring, e.g. through truncated selection
- Now, we would like to use the information from all offspring, including those with smaller fitness
- We introduce a search distribution whose parameters θ are updated, e.g. a Gaussian distribution with mean and variance
- The new "objective function" $J(\theta)$ is now the expected fitness under this search distribution $\pi(z|\theta)$

$$J(\theta) = \mathbb{E}_{z\sim\pi(z|\theta)}[f(z)] = \int f(z)\,\pi(z|\theta)\,dz\,,$$

  where $f(z)$ is the original fitness function.

- Now, the core idea is to use search gradients to update the parameters θ of the search distribution $\pi(z|\theta)$

ES with a Search Distribution

If we use θ to denote the parameters of density $\pi(\mathbf{z} \,|\, \theta)$ and $f(\mathbf{z})$ to denote the fitness function for samples $\mathbf{z}$, we can write the expected fitness under the search distribution as

$$J(\theta) = \mathbb{E}_\theta[f(\mathbf{z})] = \int f(\mathbf{z})\ \pi(\mathbf{z} \,|\, \theta)\ d\mathbf{z}. \tag{1}$$

The so-called 'log-likelihood trick' enables us to write

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \int f(\mathbf{z})\ \pi(\mathbf{z} \,|\, \theta)\ d\mathbf{z} \\
&= \int f(\mathbf{z})\ \nabla_\theta \pi(\mathbf{z} \,|\, \theta)\ d\mathbf{z} \\
&= \int f(\mathbf{z})\ \nabla_\theta \pi(\mathbf{z} \,|\, \theta)\ \frac{\pi(\mathbf{z} \,|\, \theta)}{\pi(\mathbf{z} \,|\, \theta)}\ d\mathbf{z} \\
&= \int \Big[f(\mathbf{z})\ \nabla_\theta \log \pi(\mathbf{z} \,|\, \theta) \Big]\ \pi(\mathbf{z} \,|\, \theta)\ d\mathbf{z} \\
&= \mathbb{E}_\theta \left[f(\mathbf{z})\ \nabla_\theta \log \pi(\mathbf{z} \,|\, \theta) \right].
\end{aligned}$$

From this form we obtain the estimate of the search gradient from samples $\mathbf{z}_1 \ldots \mathbf{z}_\lambda$ as

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\ \nabla_\theta \log \pi(\mathbf{z}_k \,|\, \theta), \tag{2}$$

where λ is the population size. This gradient on expected fitness provides a search direction in the space of search distributions.

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

ES with a Search Distribution

Algorithm 1: Canonical Search Gradient algorithm

input: f, θ_{init}

repeat

  for $k = 1 \ldots \lambda$ **do**

    draw sample $\mathbf{z}_k \sim \pi(\cdot|\theta)$

    evaluate the fitness $f(\mathbf{z}_k)$

    calculate log-derivatives $\nabla_\theta \log \pi(\mathbf{z}_k|\theta)$

  end

  $$\nabla_\theta J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$$

  $$\theta \leftarrow \theta + \eta \cdot \nabla_\theta J$$

until *stopping criterion is met*

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

ES with a Search Distribution

Algorithm 2: Search Gradient algorithm: Multinormal distribution

input: f, $\boldsymbol{\mu}_{init}$,$\boldsymbol{\Sigma}_{init}$
repeat
  for $k = 1 \ldots \lambda$ **do**
    draw sample $\mathbf{z}_k \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
    evaluate the fitness $f(\mathbf{z}_k)$
    calculate log-derivatives:
    $\nabla_{\boldsymbol{\mu}} \log \pi\left(\mathbf{z}_k|\theta\right) = \boldsymbol{\Sigma}^{-1}\left(\mathbf{z}_k - \boldsymbol{\mu}\right)$
    $\nabla_{\boldsymbol{\Sigma}} \log \pi\left(\mathbf{z}_k|\theta\right) = -\frac{1}{2}\boldsymbol{\Sigma}^{-1} + \frac{1}{2}\boldsymbol{\Sigma}^{-1}\left(\mathbf{z}_k - \boldsymbol{\mu}\right)\left(\mathbf{z}_k - \boldsymbol{\mu}\right)^{\top}\boldsymbol{\Sigma}^{-1}$
  end
  $\nabla_{\boldsymbol{\mu}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
  $\nabla_{\boldsymbol{\Sigma}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\Sigma}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta \cdot \nabla_{\boldsymbol{\mu}} J$
  $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma} + \eta \cdot \nabla_{\boldsymbol{\Sigma}} J$
until *stopping criterion is met*

[http://www.jmlr.org/papers/volume15/wierstra14a/wierstra14a.pdf]

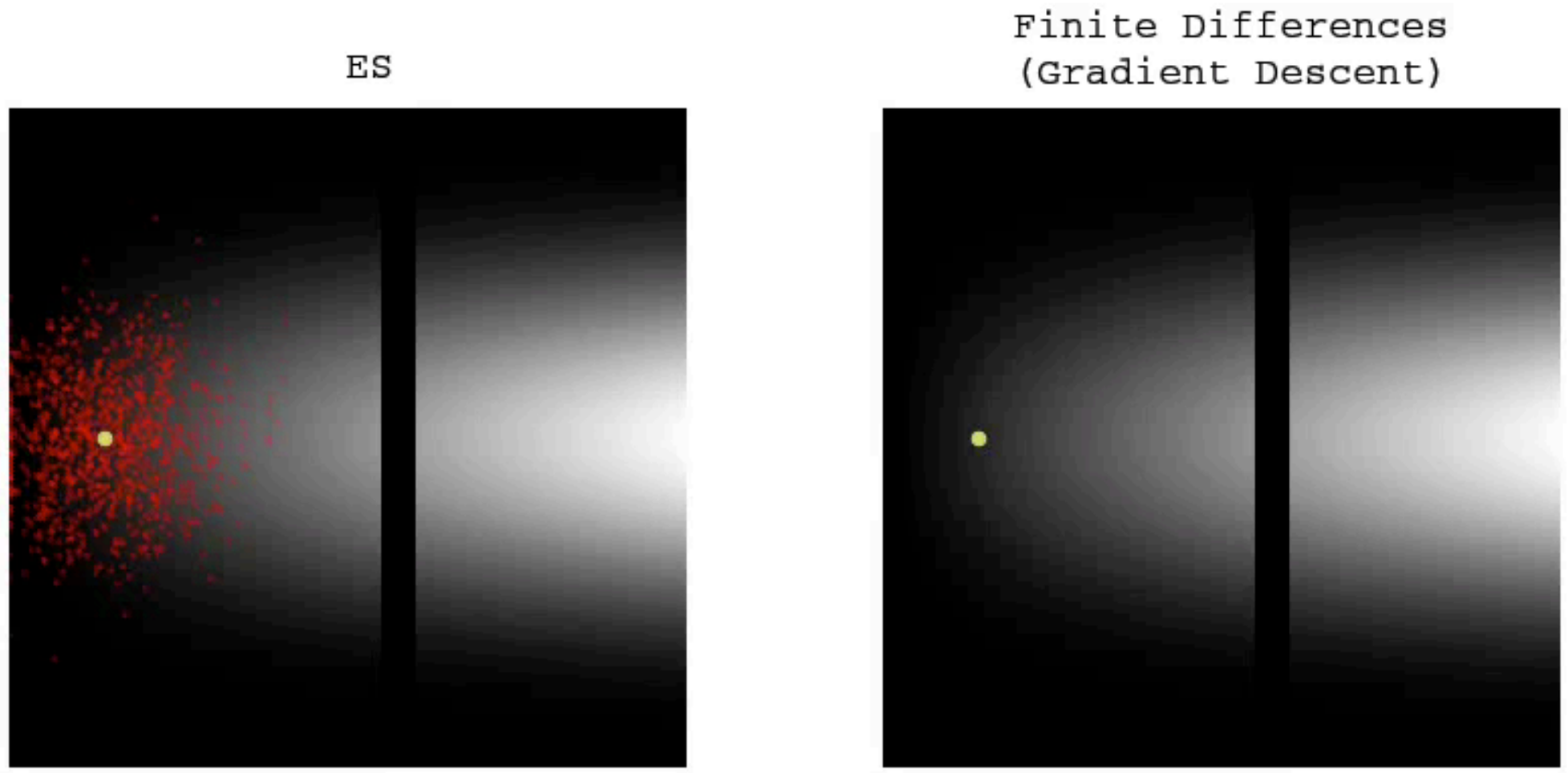
Tradeoff ES and SGD

Note

- ES optimises reward of a population of policies described by a probability distribution (a cloud in the search space)
- SGD optimises reward for a single policy (a point in the search space)

Image

- traditional finite differences (gradient descent) cannot cross a narrow gap of low fitness while ES easily crosses it to find higher fitness on the other side

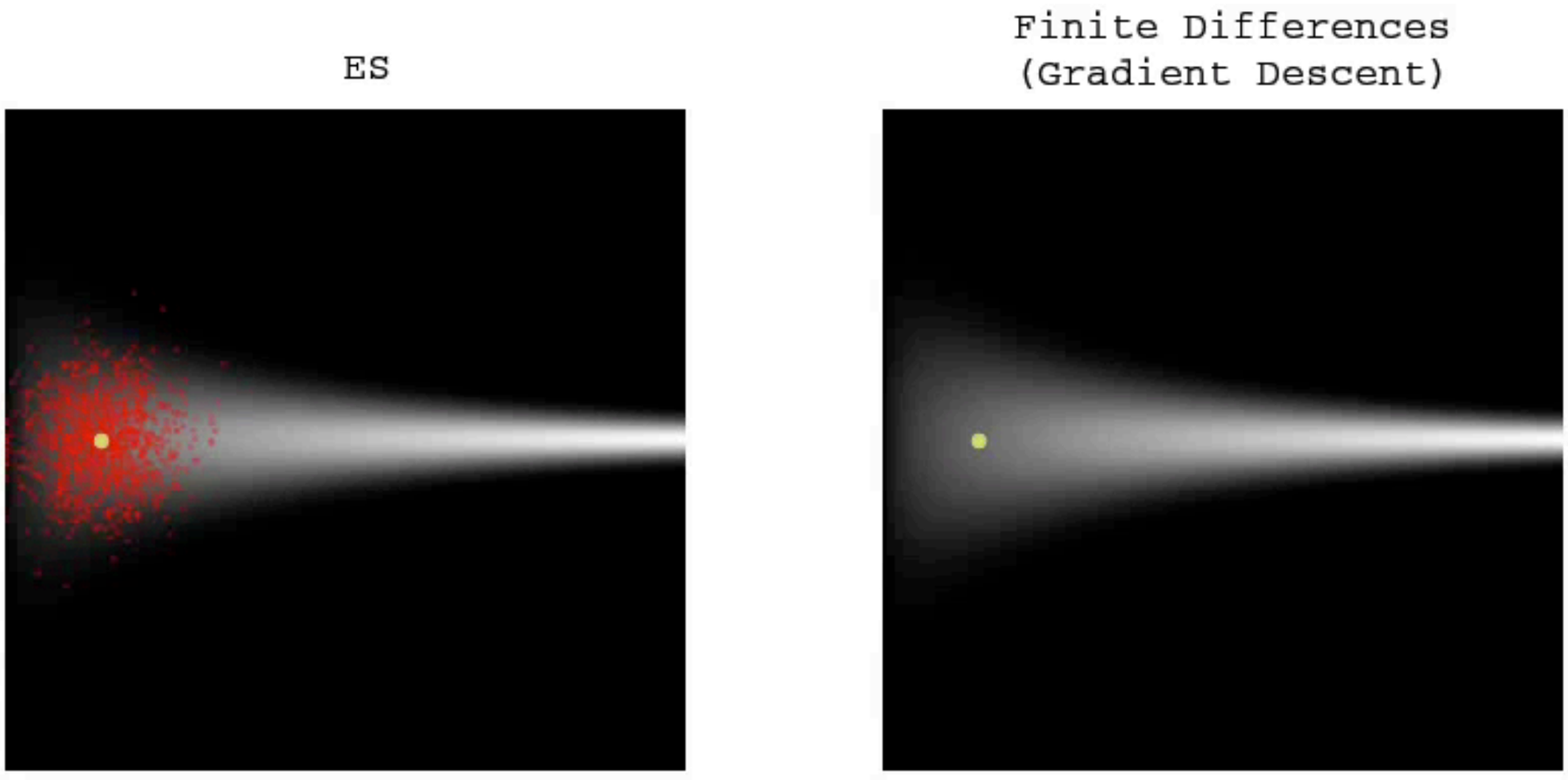


[https://eng.uber.com/deep-neuroevolution/]

Tradeoff ES and SGD

Image

- ES stalls as a path of high fitness narrows, while traditional finite differences (gradient descent) traverses the same path with no problem, illustrating along with the previous video the distinction and trade-offs between the two different approaches.



[https://eng.uber.com/deep-neuroevolution/]

ES in State-of-the Art RL

[arXiv: 1703.03864]

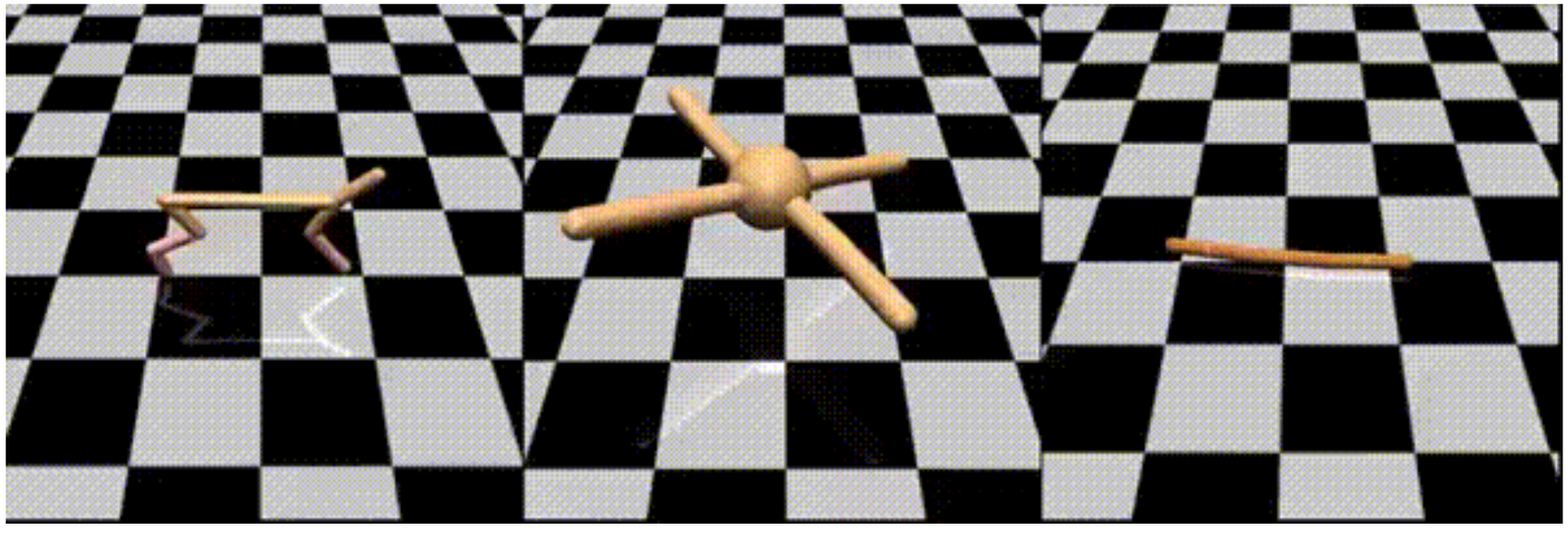
- shows that ES can be competitive with Q-learning/PG methods in RL in certain environments
- parameter space has roughly 1 million dimensions (policy network weights)
- make use of massive parallelisation

Algorithm 2 Parallelized Evolution Strategies

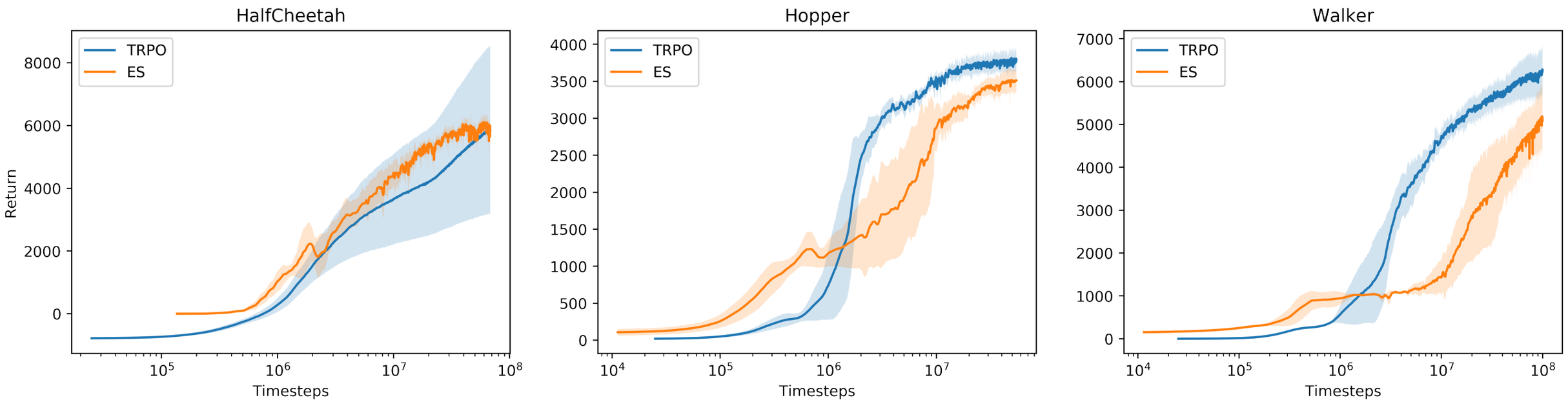
Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i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

ES in State-of-the Art RL

MuJoCo tasks with objective to move forward



ES (orange) can reach a comparable performance to TRPO (blue)



ES in State-of-the Art RL

Smoothing in parameter space versus smoothing in action space

can only be accessed via sampling. Explicitly, suppose we wish to solve general decision problems that give a return $R(\mathbf{a})$ after we take a sequence of actions $\mathbf{a} = \{a_1, \ldots, a_T\}$, where the actions are determined by a either a deterministic or a stochastic policy function $a_t = \pi(s; \theta)$. The objective we would like to optimize is thus

$$F(\theta) = R(\mathbf{a}(\theta)).$$

Since the actions are allowed to be discrete and the policy is allowed to be deterministic, $F(\theta)$ can be non-smooth in θ. More importantly, because we do not have explicit access to the underlying state transition function of our decision problems, the gradients cannot be computed with a backpropagation-like algorithm. This means we cannot directly use standard gradient-based optimization methods to find a good solution for θ.

In order to both make the problem smooth and to have a means of to estimate its gradients, we need to add noise. Policy gradient methods add the noise in action space, which is done by sampling the actions from an appropriate distribution. For example, if the actions are discrete and $\pi(s; \theta)$ calculates a score for each action before selecting the best one, then we would sample an action $\mathbf{a}(\epsilon, \theta)$ (here ϵ is the noise source) from a categorical distribution over actions at each time period, applying a softmax to the scores of each action. Doing so yields the objective $F_{PG}(\theta) = \mathbb{E}_\epsilon R(\mathbf{a}(\epsilon, \theta))$, with gradients

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

Evolution strategies, on the other hand, add the noise in parameter space. That is, they perturb the parameters as $\tilde{\theta} = \theta + \xi$, with ξ from a multivariate Gaussian distribution, and then pick actions as $a_t = \mathbf{a}(\xi, \theta) = \pi(s; \tilde{\theta})$. It can be interpreted as adding a Gaussian blur to the original objective, which results in a smooth, differentiable cost $F_{ES}(\theta) = \mathbb{E}_\xi R(\mathbf{a}(\xi, \theta))$, this time with gradients

$$\nabla_\theta F_{ES}(\theta) = \mathbb{E}_\xi \left\{ R(\mathbf{a}(\xi, \theta)) \nabla_\theta \log p(\tilde{\theta}(\xi, \theta); \theta) \right\}.$$ [arXiv: 1703.03864]

ES in State-of-the Art RL

3.1 When is ES better than policy gradients?

Given these two methods of smoothing the decision problem, which should we use? The answer depends strongly on the structure of the decision problem and on which type of Monte Carlo estimator is used to estimate the gradients $\nabla_\theta F_{PG}(\theta)$ and $\nabla_\theta F_{ES}(\theta)$. Suppose the correlation between the return and the individual actions is low (as is true for any hard RL problem). Assuming we approximate these gradients using simple Monte Carlo (REINFORCE) with a good baseline on the return, we have

$$\mathrm{Var}[\nabla_\theta F_{PG}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\mathbf{a};\theta)],$$
$$\mathrm{Var}[\nabla_\theta F_{ES}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\tilde{\theta};\theta)].$$

If both methods perform a similar amount of exploration, $\mathrm{Var}[R(\mathbf{a})]$ will be similar for both expressions. The difference will thus be in the second term. Here we have that $\nabla_\theta \log p(\mathbf{a};\theta) = \sum_{t=1}^{T} \nabla_\theta \log p(a_t;\theta)$ is a sum of T uncorrelated terms, so that the variance of the policy gradient estimator will grow nearly linearly with T. The corresponding term for evolution strategies, $\nabla_\theta \log p(\tilde{\theta};\theta)$, is independent of T. Evolution strategies will thus have an advantage compared to policy gradients for long episodes with very many time steps. In practice, the effective number of steps T is often reduced in policy gradient methods by discounting rewards. If the effects of actions are short-lasting, this allows us to dramatically reduce the variance in our gradient estimate, and this has been critical to the success of applications such as Atari games. However, this discounting will bias our gradient estimate if actions have long lasting effects. Another strategy for reducing the effective value of T is to use value function approximation. This has also been effective, but once again runs the risk of biasing our gradient estimates. Evolution strategies is thus an attractive choice if the effective number of time steps T is long, actions have long-lasting effects, and if no good value function estimates are available.

[arXiv: 1703.03864]

Recap: Evolutionary Algorithms

Advantages

- derivative-free optimisation method (no gradient required)
  - the fitness function can be noisy and non-smooth
  - optimisation is more robust to local minima (saddle points) than gradient-based methods
- only requires to know how to evaluate the objective function (fitness)
  - required for problems with little domain knowledge
  - straightforward parallelisation of computing the fitness score for each candidate
  - conceptual simplicity (flexible to adapt to specific problems)

Limitations

- if we know more about the objective function, than just how to compute it, then we can make use of this information, and construct an algorithm that performs better, e.g. using gradient information