Natural Evolution Strategies

Conference Paper · June 2008

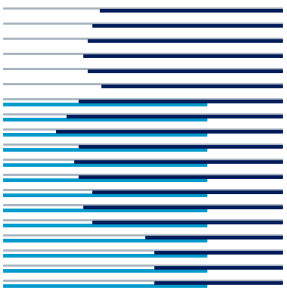
DOI: 10.1109/CEC.2008.4631255 · Source: DBLP

4 authors, including:

Jan Peters
Technische Universität Darmstadt
384 PUBLICATIONS **9,485** CITATIONS

Natural Evolution Strategies

Daan Wierstra, Tom Schaul, Jan Peters and Juergen Schmidhuber



Technical Report No. IDSIA-01-08
January 2008

IDSIA / USI-SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
Galleria 2, 6928 Manno, Switzerland

IDSIA was founded by the Fondazione Dalle Molle per la Qualità della Vita and is affiliated with both the Università della Svizzera italiana (USI) and the Scuola unversitaria professionale della Svizzera italiana (SUPSI).

Natural Evolution Strategies

Daan Wierstra, Tom Schaul, Jan Peters and Juergen Schmidhuber*

January 2008

Abstract

This paper presents Natural Evolution Strategies (NES), a novel algorithm for performing real-valued 'black box' function optimization: optimizing an unknown objective function where algorithm-selected function measurements constitute the only information accessible to the method. Natural Evolution Strategies search the fitness landscape using a multivariate normal distribution with a self-adapting mutation matrix to generate *correlated mutations* in promising regions. NES shares this property with Covariance Matrix Adaption (CMA), an Evolution Strategy (ES) which has been shown to perform well on a variety of high-precision optimization tasks. The Natural Evolution Strategies algorithm, however, is simpler, less ad-hoc and more principled. Self-adaptation of the mutation matrix is derived using a Monte Carlo estimate of the *natural gradient* towards better expected fitness. By following the natural gradient instead of the 'vanilla' gradient, we can ensure efficient update steps while preventing early convergence due to overly greedy updates, resulting in reduced sensitivity to local suboptima. We show NES has competitive performance with CMA on several tasks, while outperforming it on one task that is rich in deceptive local optima, the Rastrigin benchmark.

1 Introduction

Real-valued 'black box' function optimization is one of the major branches of modern applied machine learning research [1]. It concerns itself with optimizing the continuous parameters of some unknown objective function, also called a *fitness* function. The exact structure of the objective function is assumed to be unknown or unspecified, but specific function measurements, freely chosen by the algorithm, are available. This is a recurring problem setup in real-world domains, since often the precise structure of a problem is either not available to the engineer, or too expensive to model or simulate. Numerous real-world problems can be treated as real-valued black box function optimization problems. In order to illustrate the importance and prevalence of this general problem setup, one could point to a diverse set of tasks such as the classic nozzle shape design problem [2], developing an Aibo robot gait [3] or non-Markovian neurocontrol [4].

Now, since exhaustively searching the entire space of solution parameters is considered infeasible, and since we do not assume a precise model of our fitness function, we are forced to settle for trying to find a reasonably fit solution that satisfies certain pre-specified constraints. This, inevitably, involves using a sufficiently intelligent heuristic approach. Though this may sound crude, in practice it has proven to be crucial to find the right domain-specific trade-off on issues such as convergence speed, expected quality of the solutions found and the algorithm's sensitivity to local suboptima on the fitness landscape.

*Daan Wierstra, Tom Schaul and Juergen Schmidhuber are with IDSIA, Manno-Lugano, Switzerland (email: [daan, tom, juergen]@idsia.ch). Jan Peters is with the Max Planck Institute for Biological Cybernetics, Tuebingen, Germany (email: mail@jan-peters.net).

A variety of algorithms has been developed within this framework, including methods such as Simulated Annealing [5], Simultaneous Perturbation Stochastic Optimization [6], simple Hill Climbing, Particle Swarm Optimization [7] and the class of Evolutionary Algorithms, of which Evolution Strategies (ES) [8, 9, 10] and in particular its Covariance Matrix Adaption (CMA) instantiation [11] are of great interest to us.

Evolution Strategies, so named because of their inspiration from natural Darwinian evolution, generally produce consecutive generations of samples. During each generation, a batch of samples is generated by perturbing the parents' parameters – *mutating* their genes, if you will. (Note that ES generally uses asexual reproduction: new individuals typically are produced without crossover or similar techniques more prevalent in the field of genetic algorithms). A number of samples is *selected*, based on their fitness values, while the less fit individuals are discarded. The winners are then used as parents for the next generation, and so on, a process which typically leads to increasing fitness over the generations. This basic ES framework, though simple and heuristic in nature, has proven to be very powerful and robust, spawning a wide variety of algorithms.

In Evolution Strategies, mutations are generally drawn from a normal distribution with specific mutation sizes associated with every problem parameter. One of the major research topics in Evolution Strategies concerns the automated adjustment of the mutation sizes for the production of new samples, a procedure generally referred to as *self-adaptation* of mutation. Obviously, choosing mutation sizes too high will produce debilitating mutations and ensure that convergence to a sufficiently fit region of parameter space will be prevented. Choosing them too low leads to extremely small convergence rates and causes the algorithm to get trapped in bad local suboptima. Generally the mutation size must be chosen from a small range of values – known as the *evolution window* – specific to both the problem domain and to the distribution of current individuals on the fitness landscape. ES algorithms must therefore adjust mutation during evolution, based on the progress made on the recent evolution path. Often this is done by simultaneously evolving both problem parameters and the corresponding mutation sizes. This has been shown to produce excellent results in a number of cases [10].

The Covariance Matrix Adaptation algorithm constitutes a more sophisticated approach. CMA adapts a variance-covariance mutation *matrix* $\mathbf{\Sigma}$ used by a multivariate normal distributions from which mutations are drawn. This enables the algorithm to generate *correlated* mutations, speeding up evolution significantly for many real-world fitness landscapes. Self-adaptation of this mutation matrix is then achieved by integrating information on successful mutations on its recent evolution path, by making similar mutations more likely. CMA performs excellently on a number of benchmark tasks. One of the problems with the CMA algorithm, however, is its ad-hoc nature and relatively complex or even contrived set of mathematical justifications and 'rules of thumb'. Another problem pertains to its sensitivity to local optima.

The goal of this paper is to advance the state of the art of real-valued 'black box' function optimization while providing a firmer mathematical grounding of its mechanics, derived from first principles. Simultaneously, we want to develop a method that can achieve optimization up to arbitrarily high precision, and reduce sensitivity to local suboptima as compared to certain earlier methods such as CMA. We present an algorithm, Natural Evolution Strategies, that is both elegant and competitive, inheriting CMA's strength of correlated mutations, while enhancing the ability to prevent early convergence to local optima.

Our method, Natural Evolution Strategies (which can actually be seen as a $(1, \lambda)$-Evolution Strategy with 1 candidate solution per generation and λ samples or 'children'), adapts both a mutation matrix and the parent individual using a *natural gradient* based update step [12]. Every generation, a gradient towards better expected fitness is estimated using a Monte Carlo approximation. This gradient is then used to update both the parent individual's parameters and the mutation matrix. By using a natural gradient instead of a 'vanilla' gradient, we can prevent early convergence to local optima, while ensuring large update steps. We show that the algorithm has competitive performance with CMA on a number of benchmarks, while it outperforms CMA on the Rastrigin function benchmark, a task with many local suboptima.

The paper is organized as follows. The next section provides a quick overview of the general problem framework of real-valued black box function optimization. The ensuing sections describe the derivation of

the 'vanilla' gradient approach, the concept of 'fitness shaping', and the natural gradient instantiation of our algorithm. The section of experimental results shows our initial results with a number of benchmark problems, and compares the performance to the CMA algorithm. The paper concludes with a discussion on the advantages and problems of the method, and points out possible directions for future work.

2 Algorithm Framework

First let us introduce the algorithm framework and the corresponding notation. The objective is to optimize the n-dimensional continuous vector of objective parameters $\mathbf{x}$ for an unknown fitness function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The function is unknown or 'black box', in that the only information accessible to the algorithm consists of function measurements selected by the algorithm. The goal is to optimize $f(\mathbf{x})$, while keeping the number of function evaluations – which are considered costly – as low as possible. This is done by evaluating a number $1 \dots \lambda$ of separate individuals $\mathbf{z}_1 \dots \mathbf{z}_\lambda$ each successive generation g, using the information from fitness evaluations $f(\mathbf{z}_1) \dots f(\mathbf{z}_\lambda)$ to adjust both the current candidate objective parameters $\mathbf{x}$ and the mutation sizes.

In conventional Evolution Strategies, optimization is achieved by mimicking natural evolution: at every generation, parent solution $\mathbf{x}$ produces offspring $\mathbf{z}_1 \dots \mathbf{z}_\lambda$ by *mutating* string $\mathbf{x}$ using a multivariate normal distribution with zero mean and some variance σ. After evaluating all individuals, the best μ individuals are kept (*selected*), stored as candidate solutions and subsequently used as 'parents' for the next generation. This simple process is known to produce excellent results for a number of challenging problems.

Algorithm 1 Natural Evolution Strategies

$g = 0$, initialize population parameters $\theta^{(g)} = (\mathbf{x}, \mathbf{\Sigma})$
repeat
  for $k = 1 \dots \lambda$ **do**
    draw $\mathbf{z}_k \sim \pi(\mathbf{x}, \mathbf{\Sigma})$
    evaluate fitness of individual $f(\mathbf{z}_k)$
    $\nabla_{\mathbf{x}} \log \pi\left(\mathbf{z}_k\right) = \mathbf{\Sigma}^{-1}\left(\mathbf{z}_k - \mathbf{x}\right)$,
    $\nabla_{\mathbf{\Sigma}_m} \log \pi\left(\mathbf{z}_k\right) = 2\mathbf{A} - \operatorname{diag} \mathbf{A}$,
    where $\mathbf{A} \equiv \frac{1}{2}\mathbf{\Sigma}^{-1}\left(\mathbf{z}_k - \mathbf{x}\right)\left(\mathbf{z}_k - \mathbf{x}\right)^{\mathbf{T}} \mathbf{\Sigma}^{-1} - \frac{1}{2}\mathbf{\Sigma}^{-1}$
  end for
  $\mathbf{\Phi} = \begin{bmatrix} \nabla_\theta \log \pi(\mathbf{z}_1) & 1 \\ \vdots & \\ \nabla_\theta \log \pi(\mathbf{z}_\lambda) & 1 \end{bmatrix}$
  $\mathbf{R} = [f(\mathbf{z}_1), \dots, f(\mathbf{z}_\lambda)]^{\mathbf{T}}$
  $\delta\theta = (\mathbf{\Phi}^{\mathbf{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathbf{T}}\mathbf{R}$
  $\theta^{(g+1)} \leftarrow \theta^{(g)} + \beta \cdot \delta\theta$
  $g \leftarrow g + 1$
until stopping criterion is met

3 'Vanilla' Gradients for Evolution Strategies

Our approach, however, is different from conventional Evolution Strategies in one important respect. Instead of 'wasting' information by discarding low-fitness samples, we aim to use all available fitness values, even the bad ones, to generate a *gradient* for updating our population.

The core idea is that we want to optimize expected 'fitness' $J = \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})]$ of the next generation. We assume at every generation g a population $\pi^{(g)}$ parameterized by $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$, representing the current

candidate solution ('parent') $\mathbf{x}$ and mutation matrix $\mathbf{\Sigma}$ used for producing the next generation of search points.

In order to adjust parameters $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$ towards solutions that are likely more fit, we estimate a gradient on θ for the expected fitness. Now let $f(\mathbf{z})$ be the fitness at a particular search point $\mathbf{z}$, and, utilizing the familiar multivariate normal distribution, let

$$\begin{aligned} \pi(\mathbf{z}|\theta) &= \mathcal{N}(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}) \\ &= \frac{1}{(2\pi)^{n/2}|\mathbf{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{z}-\mathbf{x})^{\mathbf{T}}\mathbf{\Sigma}^{-1}(\mathbf{z}-\mathbf{x})\right] \end{aligned}$$

denote the probability density of search point $\mathbf{z}$ given the current population $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$. Expected fitness can then be expressed as

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

Taking the derivative of J with respect to θ of population π, we can write

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

using the 'likelihood-ratio' trick. Taking a Monte Carlo approximation of this expectation by choosing λ search points, we get

$$\begin{aligned} \nabla_\theta J &= \mathbf{E}_{\mathbf{z}}\left[\nabla_\theta \log \pi(\mathbf{z}|\theta) f(\mathbf{z})\right] \\ &\approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) f(\mathbf{z}_k) \end{aligned}$$

The population parameter vector $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$ is comprised of both the current candidate solution center and its mutation matrix, concatenated in one single vector. In order to calculate the derivatives of the log-likelihood with respect to individual elements of θ for this mixture of multivariate normal distributions, first note that

$$\begin{aligned} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) &= \frac{n}{2}\log(2\pi) - \frac{1}{2}\log\det\mathbf{\Sigma} \\ &- \frac{1}{2}\left(\mathbf{z}-\mathbf{x}\right)^{\mathbf{T}}\mathbf{\Sigma}^{-1}\left(\mathbf{z}-\mathbf{x}\right). \end{aligned}$$

We will need its derivatives, that is, $\nabla_{\mathbf{x}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right)$ and $\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right)$. The first is trivially

$$\nabla_{\mathbf{x}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) = \mathbf{\Sigma}^{-1}\left(\mathbf{z}-\mathbf{x}\right),$$

while the latter is more complex. For symmetric $\mathbf{\Sigma}$, we have

$$\begin{aligned}\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) &= \tilde{\nabla}_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) \\ &+ \tilde{\nabla}_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right)^{\mathbf{T}} \\ &- \operatorname{diag} \tilde{\nabla}_{\mathbf{\Sigma}} \log \pi\left(z|\mathbf{x}, \mathbf{\Sigma}\right)^{\mathbf{T}}\end{aligned}$$

where $\tilde{\nabla}_{\mathbf{\Sigma}}$ denotes the derivative which ignores the symmetry of $\mathbf{\Sigma}$ and $\operatorname{diag} \mathbf{A}$ denotes the diagonal of matrix $\mathbf{A}$. Now, we have

$$\begin{aligned}\tilde{\nabla}_{\Sigma} \log \pi\left(\mathbf{z}|\theta\right) &= \frac{1}{2}\mathbf{\Sigma}^{-\mathbf{T}}\left(\mathbf{z}-\mathbf{x}\right)\left(\mathbf{z}-\mathbf{x}\right)^{\mathbf{T}}\mathbf{\Sigma}^{-\mathbf{T}} - \mathbf{\Sigma}^{-\mathbf{T}} \\ &= \frac{1}{2}\mathbf{\Sigma}^{-1}\left(\mathbf{z}-\mathbf{x}\right)\left(\mathbf{z}-\mathbf{x}\right)^{\mathbf{T}}\mathbf{\Sigma}^{-1} - \mathbf{\Sigma}^{-1} \\ &\equiv \mathbf{A},\end{aligned}$$

as $\mathbf{\Sigma}$'s symmetry ensures $\mathbf{\Sigma}^{-\mathbf{T}} = \mathbf{\Sigma}^{-1}$. Due to this symmetry, we then have

$$\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) = 2\mathbf{A} - \operatorname{diag} \mathbf{A}.$$

Using these derivatives to calculate $\nabla_\theta J$, we can then update $\theta \leftarrow \theta + \beta \nabla_\theta J$ using learning rate β. This produces a new candidate solution $\mathbf{x}^{(g+1)}$ each generation, and simultaneously self-adapts the associated mutation matrix to $\mathbf{\Sigma}^{(g+1)}$. This simple update rule, which covers both object parameters and strategy parameters in one framework, is in marked contrast to the complicated procedure the CMA algorithm uses.

4 Nonlinear Fitness Shaping

Apart from slow convergence, one of the main problems encountered by the current 'vanilla' version of the algorithm described so far is the early convergence of the algorithm due to quickly decreased mutation sizes of Σ. To see why this happens, one has to imagine the curvature of some hypothetical fitness landscape around $\mathbf{x}$. If the fitness decreases quickly in one direction along the main axis of the hyperellipsoid defined by $\mathbf{\Sigma}$, while it increases only slowly in the immediate neighborbood along the opposite direction, the estimated gradient will tend to decrease $\mathbf{\Sigma}$ too much, even driving it towards 0.

To overcome this problem, we introduce *fitness shaping*, the use of a nonlinear fitness transformation function that, intuitively speaking, 'awards' better samples more than it 'punishes' bad samples. The choice of fitness shaping function is arbitrary as long as it is monotonically increasing with the original fitness, and should therefore be considered one of the tuning parameters of the algorithm, and chosen in a domain-dependent manner. We have empirically found that ranking-based shaping functions work best for various problems, also because they circumvent the problem of extreme fitness values disproportionately distorting the gradient, making careful adaptation of the learning rate during evolution unnecessary even for problems with wildly fluctuating fitnesses. The ranking function used for all experiments in this paper shapes fitness as $f(\mathbf{z}) = i + 5\lambda i^{20}$ where i is the relative rank of $\mathbf{z}$'s original fitness in the batch $\mathbf{z}_1 \ldots \mathbf{z}_\lambda$, scaled between $0 \ldots 1$.

5 Natural Evolution Strategies

Standard gradient methods have been shown to converge slowly, however, in optimization landscapes with ridges and plateaus. An ad-hoc and often-used method for overcoming this would be the use of momentum. Natural gradients constitute a more principled approach, however. First introduced by Amari [12], natural gradients have numerous advantages over 'vanilla' gradients. Natural gradients tend to avoid overly greedy

solutions, thus providing a 'safe' method for doing updates. Additionally, it avoids overfitting on the last seen example, and has the benefit of being covariant in its parameterizations.

The traditional gradient ∇J simply follows the steepest descent in the space of the actual parameters. While this might be a good idea in many problems, the main drawback comes if we need to maintain uncertainty as it generates the necessary exploration for the solutions. In this case, we need to stay close to the presented type of solutions while maximizing the fitness. As our solutions are presented by a random sample, we need to use a measure of distance $D(\hat{\theta}||\theta)$ between probability distributions $\pi_\theta(\mathbf{z})$ and $\pi_{\hat{\theta}}(\mathbf{z})$. The natural measure distance between two probability distributions is the Kullback-Leibler divergence, but alternatively one could use the Hellinger distance. In this case, the natural gradient is given by

$$\begin{aligned} \max_{\delta\theta} J(\theta + \delta\theta) &= \delta\theta^{\mathbf{T}} \nabla J, \\ s.t.\ D(\theta + \delta\theta||\theta) &= \varepsilon, \end{aligned}$$

where $J(\theta)$ is the expected fitness of population π parameterized by θ, $\delta\theta$ is the direction of constrained steepest descent, ∇J is the steepest descent or gradient, $D(\theta + \delta\theta||\theta)$ a measure of closeness on probability distributions and ε a small increment size.

If $D(\theta + \delta\theta||\theta)$ is the Kullback-Leibler divergence or the Hellinger distance, we have

$$D(\theta + \delta\theta||\theta) = \delta\theta^{\mathbf{T}} \mathbf{F}(\theta)\, \delta\theta + (\text{const}),$$

for small $\delta\theta \to 0$, where

$$\begin{aligned} \mathbf{F}(\theta) &= \int \pi(\mathbf{z}) \nabla \log \pi(\mathbf{z}) \nabla \log \pi(\mathbf{z})^{\mathbf{T}} d\mathbf{z}, \\ &= \mathbf{E}\left[\nabla \log \pi(\mathbf{z}) \nabla \log \pi(\mathbf{z})^{\mathbf{T}}\right] \end{aligned}$$

is the Fisher information matrix which yields the natural gradient $\delta\theta$ defined by the necessary condition

$$\mathbf{F}(\theta)\, \delta\theta = \beta \nabla J,$$

with β being the learning rate.

Additionally, we can introduce a *fitness baseline* b as

$$\begin{aligned} \nabla J &= \int \nabla\pi(\mathbf{z})\, f(\mathbf{z})\, d\mathbf{z} + 0 \\ &= \int \nabla\pi(\mathbf{z})\, f(\mathbf{z})\, d\mathbf{z} + b\nabla \underbrace{\int \pi(\mathbf{z})\, d\mathbf{z}}_{=1} \\ &= \int \nabla\pi(\mathbf{z})\, f(\mathbf{z})\, d\mathbf{z} + b \int \nabla\pi(\mathbf{z})\, d\mathbf{z} \\ &= \int \nabla\pi(\mathbf{z}) \left[f(\mathbf{z}) - b\right] d\mathbf{z} \\ &= \int \pi(\mathbf{z}) \nabla \log \pi(\mathbf{z}) \left[f(\mathbf{z}) - b\right] d\mathbf{z} \\ &= \mathbf{E}\left[\nabla \log \pi(\mathbf{z}) \left[f(\mathbf{z}) - b\right]\right]. \end{aligned}$$

Thus, we have the fitness baseline parameter b which can be used to reduce the estimation variance $\mathrm{Var}\left[\nabla \log \pi(\mathbf{z}) \left[f(\mathbf{z}) - b\right]\right]$. Note that

$$\mathrm{Var}\left[\nabla \log \pi(\mathbf{z})\, Cf(\mathbf{z})\right] = \mathrm{Var}\left[\nabla \log \pi(\mathbf{z})\, f(\mathbf{z})\right] C^2,$$

that is, the variance grows quadratically with the average magnitude of the fitnesses. It can be significantly reduced if a proper fitness baseline is used, reducing the number of samples required to correctly estimate the gradient. This changes the equation to

$$\mathbf{E}\left[\phi\left(\mathbf{z}\right)\phi\left(\mathbf{z}\right)^{\mathbf{T}}\right]\delta\theta = \mathbf{E}\left[\phi\left(\mathbf{z}\right)f\left(\mathbf{z}\right)\right] - \mathbf{E}\left[\phi\left(\mathbf{z}\right)b\right]$$

with $\phi\left(\mathbf{z}\right) = \nabla \log \pi\left(\mathbf{z}\right)$ with one open parameter b. We obtain b by realizing the lower bound

$$\begin{aligned}
&\mathrm{Var}\left[\phi\left(\mathbf{z}\right)\left[f\left(\mathbf{z}\right) - b\right]\right] \\
&= \bar{f}^2\,\mathrm{Var}\left[\phi\left(\mathbf{z}\right)\frac{\left(f\left(\mathbf{z}\right) - \bar{f}\right)}{\bar{f}}\right] + \mathrm{Var}\left[\phi\left(\mathbf{z}\right)\left[\bar{f} - b\right]\right], \\
&\geq \bar{f}^2\mathbf{E}\left[\phi\left(\mathbf{z}\right)\phi\left(\mathbf{z}\right)^{\mathbf{T}}\right] + \mathrm{Var}\left[\phi\left(\mathbf{z}\right)\left[\bar{f} - b\right]\right],
\end{aligned}$$

where $\bar{f} = \mathbf{E}\left[f\left(\mathbf{z}\right)\right]$. Thus, we have a minimum at $b = \bar{f}$. However, this is not ideal as it ignores the interplay between $\delta\theta$ and b. However, as $\mathbf{E}\left[\phi\left(\mathbf{z}\right)\right] = 0$, we can obtain the equation

$$\begin{aligned}
0 + b &= \mathbf{E}\left[f\left(\mathbf{z}\right)\right], \\
\mathbf{E}\left[\phi\left(\mathbf{z}\right)\right]^{\mathbf{T}}\delta\theta + b &= \mathbf{E}\left[f\left(\mathbf{z}\right)\right].
\end{aligned}$$

Now, we have the equation system

$$\begin{aligned}
\mathbf{E}\left[\phi\left(\mathbf{z}\right)\phi\left(\mathbf{z}\right)^{\mathbf{T}}\right]\delta\theta + \mathbf{E}\left[\phi\left(\mathbf{z}\right)b\right] &= \mathbf{E}\left[\phi\left(\mathbf{z}\right)f\left(\mathbf{z}\right)\right] \\
\mathbf{E}\left[\phi\left(\mathbf{z}\right)\right]^{\mathbf{T}}\delta\theta + b &= \mathbf{E}\left[f\left(\mathbf{z}\right)\right].
\end{aligned}$$

This system can be solved straightforwardly as a linear regression problem using the pseudoinverse, and when replacing the $\mathbf{E}\left[\cdot\right]$ by sample averages, we obtain the general natural gradient estimator

$$\delta\theta = \left(\mathbf{\Phi}^{\mathbf{T}}\mathbf{\Phi}\right)^{-\mathbf{1}}\mathbf{\Phi}^{\mathbf{T}}\mathbf{R}$$

where

$$\begin{aligned}
\mathbf{\Phi} &= \begin{bmatrix} \nabla_\theta \log \pi(\mathbf{z}_1) & 1 \\ & \vdots \\ \nabla_\theta \log \pi(\mathbf{z}_\lambda) & 1 \end{bmatrix} \\
\mathbf{R} &= \left[f(\mathbf{z}_1), \ldots, f(\mathbf{z}_\lambda)\right]^{\mathbf{T}}
\end{aligned}$$

The resulting Natural Evolution Strategies algorithm is described in pseudocode in Algorithm 1

6 Experiments

To test the performance of the algorithm, we chose five different nonlinear test functions. Good test functions should be easy to interpret, but can scale up with n, be highly nonlinear or largely resistant to hill-climbing, or have deceptive local suboptima. We chose to minimize the classic SphereFunction, the Rosenbrock benchmark, the Tablet task, the SchwefelFunction and the Rastrigin benchmark. All functions were tested with both Natural Evolution Strategies and the CMA algorithm, as described in [11]. An overview of the function definitions can be seen in Table 1.

The tunable parameters of the NES algorithm are comprised of A) population/batch size λ, B) the fitness shaping function f and C) the learning rate. Tuning these parameters instead of taking robust

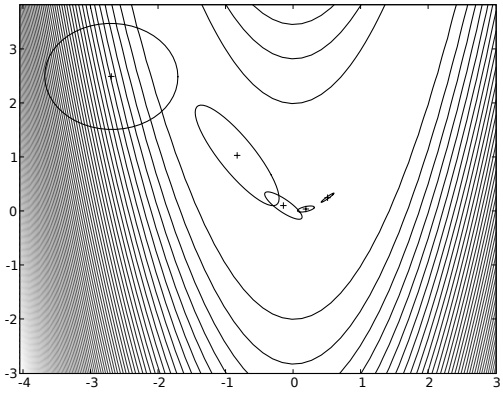


Figure 1: The evolution of the mutation matrices over the generations. Shown are the $\mathbf{\Sigma}$-defined ellipsoids of generations $0, 20, 40, 60$ and 80 imposed on the fitness landscape of the Rosenbrock benchmark. The function has its optimum at point $(1, 1)$. It can clearly be seen how the mutations 'learn' to align themselves in the local direction towards improved fitness.

Table 1: Test functions (to be minimized)

Name	Fitness Function
SphereFunction	$\sum_{j=1}^{n} z_j^2$
SchwefelFunction	$\sum_{j=1}^{n} \left[\sum_{k=1}^{j} z_j\right]^2$
Tablet	$(1000 z_1)^2 + \sum_{j=2}^{n} z_j^2$
Rosenbrock	$\sum_{j=1}^{n-1} \left[100 \left(z_j^2 - z_{j+1}\right)^2 + (z_j - 1)^2\right]$
Rastrigin	$10n + \sum_{j=1}^{n} \left[z_j^2 - 10\cos\left(2\pi z_j\right)\right]$

default settings can speed up the algorithm by roughly a factor 3. We did this for all experiments. The first parameter, the batch size, should be chosen such that underconstrained linear regression during the pseudoinverse does not occur. This entails a minimum batch size of $n + n(n + 1)/2$ (the size of the triangular matrix plus the length of the vector), but larger sizes, up to a factor 3 larger, have been found to converge sooner. The second parameter, the fitness shaping function, was taken to be $f(\mathbf{z}) = i+5\lambda i^{20}$ in all our experiments. The algorithm proved robust to changes in the specific parameters used in this function. The last parameter, the learning rate, should be set as high as possible without destabilizing the updates. Too large updates can cause destructive mutations to both $\mathbf{x}$ and $\mathbf{\Sigma}$, and the algorithm diverges or reaches early convergence especially if values of $\mathbf{\Sigma}$ are driven too close to 0. To accommodate this problem, we experimented with separate learning rates for $\mathbf{x}$ and $\mathbf{\Sigma}$, and found that, though problem-dependent, good performance is typically reached when the learning rate for $\mathbf{\Sigma}$ is roughly 5 times lower than for $\mathbf{x}$. High learning rates lead to quick convergence to the global optimum, but the trade-off consists of occasionally failing runs. As said above, tuning all these parameters can improve performance, but by about a factor 3.

The results for the first four functions Sphere, Schwefel, Tablet and Rosenbrock are depicted in Figure 6. All graphs show results averaged over multiple runs, for both NES and CMA. The Sphere function, on $n = 3$, was given $\lambda = 25$, $\beta_x = \beta_\Sigma = 0.015$, and succeeded $30/49$. The same function, on $n = 10$,

was given $\lambda = 125$, $\beta_x = 0.02$, $\beta_\Sigma = 0.005$, and succeeded $48/48$. The Rosenbrock function, on $n = 3$, was given $\lambda = 60$, $\beta_x = 0.05$, $\beta_\Sigma = 0.01$, and succeeded $22/50$. For $n = 15$, it was given $\lambda = 300$, $\beta_x = 0.01$, $\beta_\Sigma = 0.002$, and succeeded $5/7$. The Schwefel function, on $n = 15$, was given $\lambda = 200$, $\beta_x = 0.01$, $\beta_\Sigma = 0.002$, and succeeded $10/10$. The Tablet function, on $n = 5$, was given $\lambda = 100$, $\beta_x = 0.01$, $\beta_\Sigma = 0.002$, and succeeded $20/20$.

The results on these four functions indicate a performance that is about 5 times slower than CMA, also on higher dimensions. NES outperforms CMA on the Rastrigin benchmark, though. See Figure 6 for a graphic illustration of this deceptive test function. As can be seen in the performance histogram, taken over 100 runs, NES nearly always reaches the global optimum irrespective of the distance to the starting $\mathbf{x}$. CMA typically fails to converge to the global optimum and gets stuck in a local suboptimum.

Figure 5 shows the evolution of the mutation matrix during a run on the Rosenbrock function. It can be seen how, over the generations, the mutations learn to align themselves perfectly with the curvature of the test function.

To summarize, our experiments indicate NES is nearly competitive with CMA on the selected high-precision test functions, and outperforms CMA significantly on the Rastrigin benchmark due to natural gradient-induced prevention of premature convergence of $\mathbf{\Sigma}$. In all likelihood, the algorithm might outperform CMA on future real-world experiments with many deceptive local suboptima.

7 Discussion

Natural Evolution Strategies constitute a well-principled approach to real-valued black box function optimization with a relatively clean derivation from first principles. Its theoretical relationship to the field of Policy Gradients [13, 14], and in particular Natural Actor-Critic [15], should be clear to any reader familiar with both fields.

The experiments however show that, on most benchmarks, NES is still roughly 5 times slower in performance than CMA. This contrasts with the results on the Rastrigin function, on which NES clearly outperforms CMA. That result suggests NES might be less sensitive to local suboptima than CMA, possibly due to the use of natural gradients which can typically better preserve uncertainty in the production of new samples, causing a better exploration-exploitation trade-off.

The results also suggest that NES and CMA scale similarly with increased dimensionality. We argue, though, that a method such as Natural Evolution Strategies or CMA should be used only for problems with relatively small dimensionality (at most a few dozen), since the number of parameters in θ grows squared with n. In order to prevent underconstrained linear regression in computing the natural gradient, one needs a sufficient number of samples per generation before executing an update step. Also for this reason the dimensionality n needs to be kept low. Alternatively, instead of using the entire mutation matrix, one could use only the diagonal if n becomes infeasibly large.

NES does require the user to manually specify some algorithm parameters: the learning rate, the batch/generation size, and the fitness shaping function. In contrast, the CMA algorithm has a set of excellent 'magic' default settings.

Future work on Natural Evolution Strategies must determine whether NES can be made to outperform CMA consistently on typical benchmarks and real-world tasks. We suggest extending the algorithm from a single multinormal distribution as population representation to a mixture of Gaussians, thus further reducing its sensitivity to local suboptima. An other improvement could be achieved by replacing the pseudoinverse by Recursive Least Squares, enabling the algorithm to make update steps after every new sample, moving away from generational batches. Finally, even though NES uses *all* samples even the low-fitness ones (unlike conventional ES and CMA), we suspect further developing sound statistical methods for better exploiting the information in low-fitness samples might greatly improve performance.

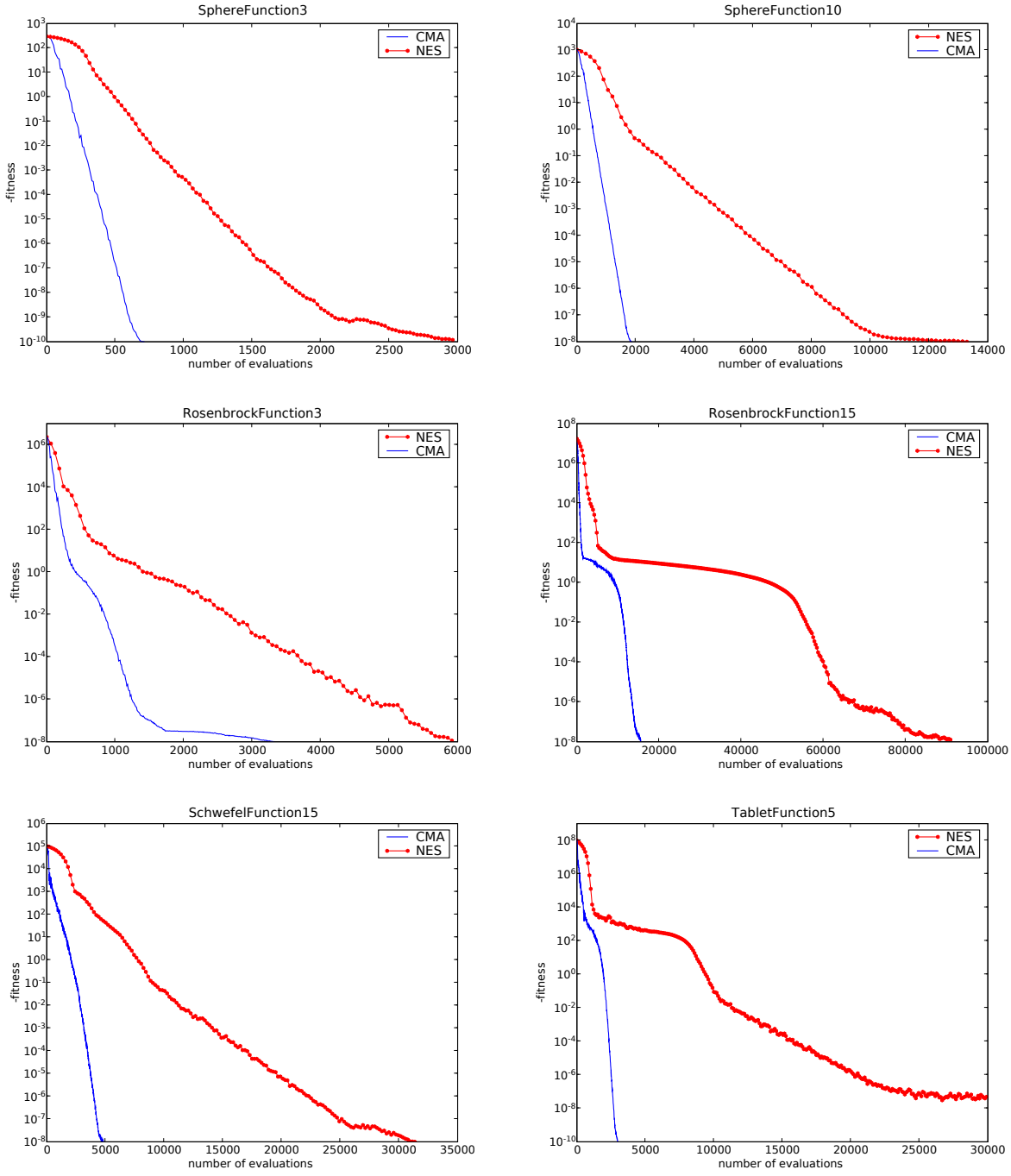


Figure 2: Results for six experiments on four different benchmark functions: the SphereFunction, the RosenbrockFunction, the TabletFunction, and the SchwefelFunction, on different dimensionalities n. Shown are averages over several runs. Notice how in most cases, NES performs roughly 5 times slower than CMA, regardless of problem dimensionality.

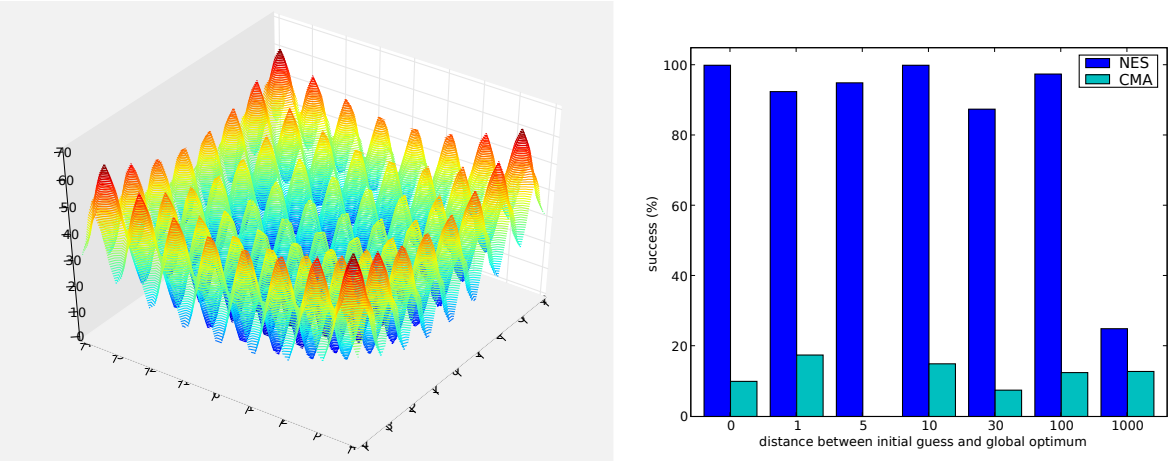


Figure 3: Performance of NES and CMA on the Rastrigin function. The histogram shows the percentage of runs that reaches the global optimum, for varying initial distances from the global optimum. Results are shown for 100 trials. NES greatly outperforms CMA, suggesting a much improved resistance to converging to bad local suboptima.

8 Conclusion

In this paper we presented Natural Evolution Strategies, a novel algorithm for tackling the important class of real-valued 'black box' function optimization problems. The proposed method is nearly competitive with the well-established Covariance Matrix Adaptation algorithm, which shares its property of producing *correlated* mutations that greatly help performance in real-world problem domains. Using a Monte Carlo-estimated *natural* gradient for updating both candidate solutions and the mutation matrix, one might suspect a reduced sensitivity to getting stuck in local suboptima, and our initial experimental results suggest that this might indeed be the case. Moreover, NES seems simpler and better-principled than CMA and other Evolution Strategies. Future work will determine whether NES can be shown to consistently outperform CMA on more realistic problem settings and benchmarks.

References

[1] J. Spall, S. Hill, and D. Stark, "Theoretical framework for comparing several stochastic optimization approaches," *Probabilistic and Randomized Methods for Design under Uncertainty*, pp. 99–117, 2006.

[2] J. Klockgether and H.-P. Schwefel, "Two-phase nozzle and hollow core jet experiments," in *Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics*, 1970, pp. 141–148.

[3] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.

[4] F. J. Gomez and R. Miikkulainen, "Solving non-Markovian control tasks with neuroevolution," in *Proc. IJCAI 99*. Denver, CO: Morgan Kaufman, 1999.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science, Number 4598, 13 May 1983*, vol. 220, 4598, pp. 671–680, 1983.

[6] J. C. Spall, "Stochastic optimization and the simultaneous perturbation method," in *WSC '99: Proceedings of the 31st conference on Winter simulation*. New York, NY, USA: ACM, 1999, pp. 101–109.

[7] J. Kennedy and R. C. Eberhart, *Swarm intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[8] I. Rechenberg, "Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution," Ph.D. dissertation, TU Berlin, 1971.

[9] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies: A comprehensive introduction," *Natural Computing: an international journal*, vol. 1, no. 1, pp. 3–52, 2002.

[10] H.-G. Beyer, "Toward a Theory of Evolution Strategies: Self-Adaptation," *Evolutionary Computation*, vol. 3, no. 3, pp. 311–347, 1996.

[11] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[12] S. ichi Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[13] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[14] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006, pp. 2219 – 2225.

[15] J. Peters, S. Vijayakumar, and S. Schaal, "Natural actor-critic," in *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, 2005, pp. 280–291.