DSA4212: Natural Evolution Strategies

Bao Jiaqi, A0211257N, bao.jiaqi@u.nus.edu

April 16, 2024

1 Introduction

Natural Evolution Strategies (NES) is a family of optimization algorithms that are well-suited for black-box optimization problems. In this report, we begin by introducing and outlining optimization algorithms and their variants. We delve into their derivation and establish the connection between different algorithms. Next, we apply and compare these algorithms to three classical optimization problems. Finally, we apply the most effective method to a real-world hyperparameter tuning problem.

2 Methods

In this section, we provide an introduction to the fundamental concept of the natural gradient and Evolution Strategy. We then integrate these concepts into the notion the basic Natural Evolution Strategy. Furthermore, we introduce two extensions to enhance the basic Natural Evolution Strategy.

2.1 Natural gradient

In gradient ascent, we aim to maximize the function $J(\boldsymbol{\theta})$ at the n-th iteration using a learning rate of η. The update rule is given by:

$$\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} + \eta \nabla J(\boldsymbol{\theta}^{(n)})$$

Specifically, $J(\boldsymbol{\theta})$ can be $\log p(\boldsymbol{x}|\boldsymbol{\theta})$, which equals the log-likelihood, $\boldsymbol{x}$ is the data observed and $\boldsymbol{\theta}$ is the model parameter to be estimated. It is important to note that gradient descent finds $\boldsymbol{\theta}^{(n+1)}$ in the neighborhood of $\boldsymbol{\theta}^{(n)}$ measured by Eucilean distance that minimize the objective function. However, the Euclidean distance between $\boldsymbol{\theta}$ values does not serve as a suitable metric for assessing the distances between the distributions $p(\boldsymbol{x}|\boldsymbol{\theta})$.

In each step of the natural gradient, the steepest direction is

$$\boldsymbol{d} = \underset{KL(p(.|\boldsymbol{\theta}^{(n)})||p(.|\boldsymbol{\theta}^{(n)}+\boldsymbol{d}))=\epsilon}{\arg\min} J(\boldsymbol{\theta}^{(n)} + \boldsymbol{d})$$

where $KL(p||q)$ denotes the Kullback–Leibler divergence from q to p. It can be derived that $KL(p(\boldsymbol{x}|\boldsymbol{\theta})||p(\boldsymbol{x}|\boldsymbol{\theta}+\boldsymbol{d})) \approx \frac{-1}{2}\boldsymbol{d}^T \boldsymbol{F}_{\boldsymbol{\theta}^{(n)}} \boldsymbol{d}$, where $\boldsymbol{F}_{\boldsymbol{\theta}^{(n)}} = \mathbb{E}_{\boldsymbol{x}}[\nabla \log(p(\boldsymbol{x}|\boldsymbol{\theta}^{(n)})) \nabla \log(p(\boldsymbol{x}|\boldsymbol{\theta}^{(n)}))^T]$ is the Fisher information matrix of distribution $p(\boldsymbol{x}|\boldsymbol{\theta}^{(n)})$. Using a Lagrangian multiplier, one can derive the steepest direction: $\boldsymbol{d}^* = \boldsymbol{F}^{-1}_{\boldsymbol{\theta}^{(n)}} \nabla J(\boldsymbol{\theta}^{(n)})$, which is called *natural gradient*. The detailed derivations can be found in [Wen19].

One advantage of the natural gradient is its invariance under linear transformations of the coordinates, similar to Newton's method. However, a significant drawback is the computational difficulty in calculating the inverse Fisher information matrix, particularly when the dimension of $\boldsymbol{\theta}$ is large and matrix inversion introduces considerable noise.

2.2 Evolution Strategy

In many real-life optimization problems, it can be extremely challenging to calculate the gradient and Hessian of the objective function. Now, we turn our attention to the black box optimization task of maximizing $f(\boldsymbol{x})$, where we only have access to the function values of f and no information about $\nabla^2 f(\boldsymbol{x})$ or $\nabla f(\boldsymbol{x})$.

Evolution Strategies (ES) provide a solution to the black box optimization problem. The aim is to find a probability distribution parameterized by $\boldsymbol{\theta}$, such that the sample $\boldsymbol{x} \sim p(\cdot|\boldsymbol{\theta})$ corresponds to a good solution for $f(\boldsymbol{x})$. Our goal is to maximize the *expected fitness* under $\boldsymbol{\theta}$, given by:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p(.|\boldsymbol{\theta})}[f(\boldsymbol{x})] = \int f(\boldsymbol{x}) p(\boldsymbol{x}|\boldsymbol{\theta}) d\boldsymbol{x}$$

We can derive $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p(\cdot|\boldsymbol{\theta})}[f(\boldsymbol{x}) \nabla \log p(\boldsymbol{x}|\boldsymbol{\theta})]$ (see details in [WSG+14]). To optimize $J(\boldsymbol{\theta})$, we can apply a standard gradient ascent scheme, and the outlined procedure is presented in Algorithm 1.

Algorithm 1 Evolution Strategies (ES)

1: **Input**: objective function f, population size λ, learning rate η, initialized parameters $\boldsymbol{\theta}^{(0)}$, number of generations N.
2: **for** each generation $n = 0, \ldots, N-1$ **do**
3: Generate samples $\{\boldsymbol{x}_i\}_{i=1}^{\lambda}$ identically and independently from $p(.|\boldsymbol{\theta}^{(n)})$
4: Evaluate $f(\boldsymbol{x}_i)$ for each sample $\boldsymbol{x}_i$
5: Compute $\nabla \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)})$ for each sample $\boldsymbol{x}_i$
6: Estimate the gradient from samples with Monte Carlo: $\hat{\nabla} J(\boldsymbol{\theta}^{(n)}) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \nabla \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)}) f(\boldsymbol{x}_i)$
7: Update $\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} + \eta \hat{\nabla} J(\boldsymbol{\theta}^{(n)})$
8: **end for**

2.3 Natural Evolution Strategy

In this subsection, we incorporate natural gradient into Evolution Strategies, referred to as Natural Evolution Strategy (NES). More specifically, we update $\boldsymbol{\theta}^{(n)}$ with the natural gradient in Step 7 of Algorithm 1.

Here, we assume that $p(\cdot|\boldsymbol{\theta})$ follows a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{C}$. We use the Cholesky decomposition $\boldsymbol{A}\boldsymbol{A}^T$ to represent the covariance matrix $\boldsymbol{C}$. The parameters $\boldsymbol{\theta}$ of the Gaussian distribution are expressed as a tuple $(\boldsymbol{\mu}, \boldsymbol{A})$. The log-likelihood is given by:

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) \propto -\frac{1}{2}|\boldsymbol{A}\boldsymbol{A}^T| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) \tag{1}$$

We can derive the gradient of $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\mu}$ as $\nabla_{\boldsymbol{\mu}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) = \boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ In addition, the gradient of $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ w.r.t. $\boldsymbol{A}_{ij}$ for each $i \leq j$ can be derived as $\nabla_{\boldsymbol{A}_{ij}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) = \boldsymbol{R}_{ij} - \delta(i,j)\frac{1}{\boldsymbol{A}_{ii}}$, where $\boldsymbol{R} = (\boldsymbol{A}^{-1})^T(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{C}^{-1}$ and $\delta(i,j) = 1$ if $i = j$, $\delta(i,j) = 0$ otherwise; see derivation details of the Gaussian log likelihood in [YWSS09].

Moreover, in order to ensure the algorithm's invariance under rank-preserving transformations of f, the fitness $f(\boldsymbol{x}_i)$ in the compuation of $\hat{\nabla} J(\boldsymbol{\theta}^{(n)})$ can be replaced by rank-based *fitness shaping*. Specifically, we replace the fitness values $\{f(\boldsymbol{x}_i)\}_{i=1}^{\lambda}$ with *utilities* $\{u_i\}_{i=1}^{\lambda}$. After sorting the samples in descending order based on $f(\boldsymbol{x}_i)$, u_i represents the utility for $\boldsymbol{x}_i$, and the ordering is such that $u_1 \geq \ldots \geq u_\lambda$.

The combination of the formulas for the gradient of log likelihood with Algorithm 1 gives an outline of NES with fitness shaping for Gaussian search distribution, as depicted in Algorithm 2. If, in Step 8 of Algorithm 2, we omit the Fisher matrix and update the parameters solely based on the gradient, it is equivalent to the Evolution Strategy outlined in Algorithm 1 with fitness shaping for Gaussian search distribution. Additionally, for smoother estimates of gradients obtained from small samples, we employ momentum.

Algorithm 2 Natural Evolution Strategy (NES)

1: **Input**: objective function f, population size λ, learning rate η, momentum weight β, initialized parameters $\boldsymbol{\theta}^{(0)} = (\boldsymbol{\mu}^{(0)}, \boldsymbol{A}^{(0)})$, number of generations N and utilities $u_1 \geq \ldots \geq u_\lambda$.
2: **for** each generation $n = 0, \ldots, N-1$ **do**
3: Generate $\boldsymbol{z}_i \sim \mathcal{N}(0, \boldsymbol{I})$ independently and set $\boldsymbol{x}_i = \boldsymbol{\mu}^{(n)} + \boldsymbol{A}^{(n)}\boldsymbol{z}_i$ for each $i = 1, \ldots, \lambda$.
4: Evaluate $f(\boldsymbol{x}_i)$ for each sample $\boldsymbol{x}_i$ and sort $\{\boldsymbol{x}_i, \boldsymbol{z}_i\}_{i=1}^{\lambda}$ based on $f(\boldsymbol{x}_i)$ in descending order.
5: Compute $\nabla \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)})$ for each sample $\boldsymbol{x}_i$, where $\nabla_{\boldsymbol{\mu}^{(n)}} \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)}) = {\boldsymbol{C}^{(n)}}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})$ and $\nabla_{\boldsymbol{A}_{ij}^{(n)}} \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)}) = \boldsymbol{R}_{ij}^{(n)} - \delta(i,j)\frac{1}{\boldsymbol{A}_{ii}^{(n)}}$ for each $i \leq j$.
6: Estimate the gradient from samples with Monte Carlo: $\hat{\nabla} J(\boldsymbol{\theta}^{(n)}) = \frac{1}{\lambda}\sum_{i=1}^{\lambda} \nabla \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)}) u_i$
7: Estimate the Fisher matrix from samples with Monte Carlo: $\hat{\boldsymbol{F}}_{\boldsymbol{\theta}^{(n)}} = \frac{1}{\lambda}\sum_{i=1}^{\lambda} \nabla \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)}) \nabla \log p(\boldsymbol{x}_i|\boldsymbol{\theta}^{(n)})^T$.
8: Update $\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} + \eta \hat{\boldsymbol{F}}_{\boldsymbol{\theta}^{(n)}}^{-1} \hat{\nabla} J(\boldsymbol{\theta}^{(n)}) + \beta(\boldsymbol{\theta}^{(n)} - \boldsymbol{\theta}^{(n-1)})$
9: **end for**

Similar to the drawback of natural gradient mentioned in Section 2.1, vanilla NES requires computing the inverse of the estimated $\boldsymbol{F}_{\boldsymbol{\theta}^{(n)}}$. Specifically, the estimates based on generated samples may not be invertible, and a sufficiently large sample size λ is needed to obtain an accurate estimate. Additionally, the invariance under linear transformations of coordinates is partially lost when parameterizing the covariance matrix $\boldsymbol{C}$ using its Cholesky decomposition $\boldsymbol{A}$.

2.4 Exponential Natural Evolution Strategies

There are several approaches to avoid the computation of the explicit $\hat{\boldsymbol{F}}_{\boldsymbol{\theta}^{(n)}}$ and enhance the vanilla NES algorithm. For instance, [YWSS09] suggests analytically computing the *exact* $\boldsymbol{F}_{\boldsymbol{\theta}^{(n)}}$ instead of estimating it from the samples. However, this approach still requires matrix inversion. In contrast, exponential Natural Evolution Strategies (xNES) [GSY+10] introduces an exponential parameterization of $\boldsymbol{\theta}$. In this subsection, we first explain the key innovations of xNES and subsequently provide a detailed algorithm.

Updating $\boldsymbol{C}$ while ensuring it is both positive semi-definite and invariant under linear transformations of coordinates poses a challenge. One innovation of xNES is the use of the exponential map, which maps a symmetric matrix into a positive semi-definite matrix. We denote this exponential map as $\exp(\boldsymbol{M}) = \boldsymbol{U}\exp(\boldsymbol{D})\boldsymbol{U}^T$, where $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$ represents the eigen decomposition of $\boldsymbol{M}$ and $\exp(\boldsymbol{D})$ is computed by taking the exponential of the diagonal elements of $\boldsymbol{D}$. The positivity of $\exp(\boldsymbol{D})$ implies that $\exp(\boldsymbol{M})$ is positive semi-definite. As a result, $\boldsymbol{C}$ can be represented as $\exp(\boldsymbol{\xi})$, where $\boldsymbol{\xi}$ is a symmetric matrix. This representation ensures that the update of $\boldsymbol{\xi}$ is invariant under linear transformations.

Another innovation of xNES is the linear transformation of the coordinate system at each iteration to a *natural* coordinate system, in which the current search distribution $p(\cdot|\boldsymbol{\theta})$ is a standard Gaussian.

Denoting $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{A})$ as in NES, we introduce the following coordinate system for $\boldsymbol{\theta}$:

$$(\boldsymbol{\delta}, \boldsymbol{M}) \mapsto (\boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{\delta}, \boldsymbol{A}\exp(\boldsymbol{M}/2))$$

By doing so, the current search distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{A}\boldsymbol{A}^T)$ is encoded as a standard Gaussian with $(\boldsymbol{\delta}, \boldsymbol{M}) = (0, \boldsymbol{I})$. Importantly, the gradients and natural gradients of $(\boldsymbol{\delta}, \boldsymbol{M})$ coincide. Consequently, the computation of the inverse Fisher Information matrix or even the Fisher Information matrix itself can be avoided.

The detailed procedure is given in Algorithm 3. For recommended parameter settings of the learning rates and utilities, please refer to [GSY+10].

Algorithm 3 Exponential Natural Evolution Strategies (xNES)

1: **Input**: objective function f, population size λ, learning rates $\eta_{\boldsymbol{\mu}}, \eta_\sigma$ and $\eta_{\boldsymbol{B}}$, initialized parameters $\boldsymbol{\theta}^{(0)} = (\boldsymbol{\mu}^{(0)}, \boldsymbol{A}^{(0)})$, number of generations N and utilities $u_1 \geq \ldots \geq u_\lambda$
2: Set $\sigma^{(0)} \leftarrow \sqrt[p]{\det(\boldsymbol{A})}$ and $\boldsymbol{B}^{(0)} \leftarrow \boldsymbol{A}^{(0)}/\sigma^{(0)}$
3: **for** each generation $n = 0, \ldots, N-1$ **do**
4: Generate $\boldsymbol{z}_i \sim \mathcal{N}(0, \boldsymbol{I})$ independently and set $\boldsymbol{x}_i = \boldsymbol{\mu}^{(n)} + \boldsymbol{A}^{(n)}\boldsymbol{z}_i$ for each $i = 1, \ldots, \lambda$
5: Evaluate $f(\boldsymbol{x}_i)$ for each sample $\boldsymbol{x}_i$ and sort $\{\boldsymbol{x}_i, \boldsymbol{z}_i\}_{i=1}^{\lambda}$ based on $f(\boldsymbol{x}_i)$ in descending order.
6: Compute $\boldsymbol{G}_{\boldsymbol{\delta}} \leftarrow \sum_{i=1}^{\lambda} u_i \boldsymbol{z}_i$
7: Compute $\boldsymbol{G}_{\boldsymbol{M}} \leftarrow \sum_{i=1}^{\lambda} u_i(\boldsymbol{z}_i\boldsymbol{z}_i^T - \boldsymbol{I})$
8: Compute gradient of step size $\boldsymbol{G}_\sigma \leftarrow \mathrm{tr}(G_{\boldsymbol{M}})/p$
9: $\boldsymbol{G}_{\boldsymbol{B}} \leftarrow \boldsymbol{G}_{\boldsymbol{M}} - \boldsymbol{G}_\sigma \boldsymbol{I}$
10: Update $\boldsymbol{\mu}^{(n+1)} \leftarrow \boldsymbol{\mu}^{(n)} + \eta_{\boldsymbol{\mu}}\sigma^{(n)}\boldsymbol{B}^{(n)}\boldsymbol{G}_{\boldsymbol{\delta}}$
11: Update $\sigma^{(n+1)} \leftarrow \sigma^{(n)}\exp(\eta_\sigma/2\boldsymbol{G}_\sigma)$
12: Update $\boldsymbol{B}^{(n+1)} \leftarrow \boldsymbol{B}^{(n)}\exp(\eta_{\boldsymbol{B}}/2\boldsymbol{G}_{\boldsymbol{B}})$
13: **end for**

2.5 Covariance Matrix Adaptation Evolution Strategy

xNES exhibits a strong connection to the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [HO01, Han16]. In this subsection, we will establish the link between xNES and CMA-ES, and subsequently provide an introduction to CMA-ES.

If $\eta_\sigma = \eta_{\boldsymbol{B}}$, then step 11 and 12 of Algorithm 3 are reduced to $\boldsymbol{A}^{(n+1)} \leftarrow \boldsymbol{A}^{(n)}\exp(\eta_{\boldsymbol{A}}/2\boldsymbol{G}_{\boldsymbol{M}})$ where $\eta_{\boldsymbol{A}} = \eta_\sigma$. We have

$$\boldsymbol{A}^{(n+1)} \leftarrow \boldsymbol{A}^{(n)}\exp(\eta_{\boldsymbol{A}}/2\sum_{i=1}^{\lambda} u_i(\boldsymbol{z}_i\boldsymbol{z}_i^T - \boldsymbol{I})), \boldsymbol{C}^{(n+1)} \leftarrow \boldsymbol{A}^{(n)}\exp(\eta_{\boldsymbol{C}}\sum_{i=1}^{\lambda} u_i(\boldsymbol{z}_i\boldsymbol{z}_i^T - \boldsymbol{I}))(\boldsymbol{A}^{(n)})^T$$

where $\eta_{\boldsymbol{A}} = \eta_{\boldsymbol{C}}$. By the first order approximate of the exponential map, we could update $\boldsymbol{C}^{(n+1)}$ with

$$\boldsymbol{A}^{(n)}(\boldsymbol{I} + \eta_{\boldsymbol{C}}\sum_{i=1}^{\lambda} u_i(\boldsymbol{z}_i\boldsymbol{z}_i^T - \boldsymbol{I}))(\boldsymbol{A}^{(n)})^T = (1 - \eta_{\boldsymbol{C}}\sum_{i=1}^{\lambda} u_i)\boldsymbol{C}^{(n)} + \eta_{\boldsymbol{C}}\sum_{i=1}^{\lambda} u_i(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})^T \quad (2)$$

which coincides with the rank-μ update of covariance matrix in CMA-ES. However, the update rules in xNES follows from the principle of natural gradient ascent, but CMA-ES is based on heuristics.

We now introduce CMA-ES. In the basic setting of the evolution strategy with an isotropic Gaussian, updating the mean $\boldsymbol{\mu}$ and variance σ^2 is straightforward. We update them using the sample mean and

variance of the selected samples with high fitness in the current generation. It's important to note that this basic strategy is heuristic-based and does not involve gradients or natural gradients. However, the successive generations' variances σ^2 tend to be highly correlated, restricting the exploration space. CMA-ES addresses this issue by introducing a covariance matrix, denoted as $\boldsymbol{C}$, and sampling from $\mathcal{N}(\boldsymbol{\mu}, \sigma^2\boldsymbol{C})$ in each generation. σ controls the scale of the distribution, often referred to as the "step size".

CMA-ES implements the rank-μ update, which accumulates information from previous generations in the update of $\boldsymbol{C}$. Since the estimation of covariance from a small sample is often unreliable due to the usual small population size λ, leveraging information from previous quantities is crucial.

While CMA-ES and xNES share similarities in the rank-μ update, a key difference is the utilization of the "evolution path" during the update of $\boldsymbol{C}$ in CMA-ES. If we invert the sign of $\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)}$ for every i, the update of $\boldsymbol{C}$ in Eq. (1) will remain the same, but the update of $\boldsymbol{\mu}$ will result in the opposite direction. We refer to $\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)}$ as a "step" since it is involved in the update of both $\boldsymbol{\mu}$ and $\boldsymbol{C}$. To preserve the information in the sign of the steps, we accumulate the update of $\boldsymbol{\mu}$ in each generation n into the evolution path, and update $\boldsymbol{C}$ with the evolution path.

To facilitate the implementation of the algorithm from scratch, a fixed step size of $\sigma = 1$ is assumed for each generation, without controlling its value. The details are provided in Algorithm 1 (see 4). It is important to note that this version is a simplified adaptation of the full version presented in the tutorial [Han16] and the paper [NS24] of the CMA-ES python package, with certain parameters simplified.

Algorithm 4 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

1: **Input**: objective function f, population size λ, learning rates $\eta_{\boldsymbol{\mu}}, \eta_{\boldsymbol{C}}, \eta_{1,\boldsymbol{C}}$ and $\eta_{2,\boldsymbol{C}}$, evolution path damping parameter c_c, initialize $\boldsymbol{p}_{\boldsymbol{C}}^{(0)} \leftarrow \boldsymbol{0}$, initialized parameters $(\boldsymbol{\mu}^{(0)}, \boldsymbol{C}^{(0)})$, number of generations N and utilities $u_1 \geq \ldots \geq u_\lambda$
2: **for** each generation $n = 0, \ldots, N-1$ **do**
3: Generate $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}^{(n)}, \boldsymbol{C}^{(n)})$ independently for each $i = 1, \ldots, \lambda$.
4: Evaluate $f(\boldsymbol{x}_i)$ for each sample $\boldsymbol{x}_i$ and sort $\{\boldsymbol{x}_i, \boldsymbol{z}_i\}_{i=1}^{\lambda}$ based on $f(\boldsymbol{x}_i)$ in descending order.
5: Update $\boldsymbol{\mu}^{(n+1)} \leftarrow \boldsymbol{\mu}^{(n)} + \eta_{\boldsymbol{\mu}} \sum_{i=1}^{\lambda} u_i(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})$
6: Update evolution path $\boldsymbol{p}_{\boldsymbol{C}}^{(n+1)} \leftarrow (1-c_c)\boldsymbol{p}_{\boldsymbol{C}}^{(n+1)} + \sqrt{c_c(2-c_c)} \sum_{i=1}^{\lambda} u_i(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})$
7: Update $\boldsymbol{C}^{(n+1)} \leftarrow (1 - \eta_{\boldsymbol{C}})\boldsymbol{C}^{(n)}$
8: Rank-μ update $\boldsymbol{C}^{(n+1)} \leftarrow \boldsymbol{C}^{(n+1)} + \eta_{1,\boldsymbol{C}} \sum_{i=1}^{\lambda} u_i(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})(\boldsymbol{x}_i - \boldsymbol{\mu}^{(n)})^T$
9: Update with evolution path $\boldsymbol{C}^{(n+1)} \leftarrow \boldsymbol{C}^{(n+1)} + \eta_{2,\boldsymbol{C}} \boldsymbol{p}_{\boldsymbol{C}}^{(n+1)} \boldsymbol{p}_{\boldsymbol{C}}^{(n+1)^T}$
10: **end for**

In step 7 of Algorithm 4, if $\sum_{i=1}^{\lambda} u_i = 1$ and we disregard the evolutionary path (set $\eta_{2,\boldsymbol{C}} = 0$), then this update rule is equivalent to Equation 2 in xNES since we simplify $\sigma^{(n)} = 1$ at each generation n. In step 5, if $\sum_{i=1}^{\lambda} u_i = 1$ and $\eta_{\boldsymbol{\mu}} = 1$, then it is equivalent to updating the mean as $\boldsymbol{\mu}^{(n+1)} \leftarrow \sum_{i=1}^{\lambda} u_i \boldsymbol{x}_i$. If we select the top $\lambda/2$ samples with the highest fitness and set $u_i = \frac{1}{\lambda/2}$ if $i < \lambda/2$ and $u_i = 0$ otherwise, then $\boldsymbol{\mu}^{(n+1)}$ is the sample mean of the selected samples. If $\eta_{2,\boldsymbol{C}} = 0, \eta_{\boldsymbol{C}} = 0$, and $\eta_{1,\boldsymbol{C}} = 1$, then $\boldsymbol{C}^{(n+1)}$ is the sample covariance of the selected samples. The simplified CMA-ES under the above settings explains the basic heuristics behind.

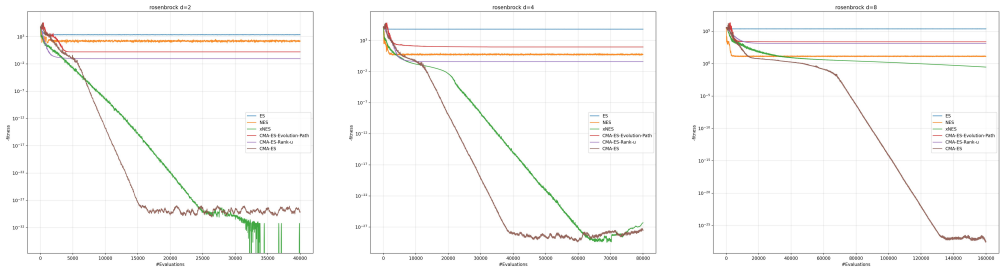

Figure 1: Rosenbrock with $d = 2, 4, 8$. We plot the evaluation of all $R(\boldsymbol{x}_i)$ against the number of evaluations.

3 Applications

3.1 Experiment settings

Here are some details about the experiment settings. In ES and NES with fitness shaping for Gaussian search distribution outlined in Algorithm 2, we decay the learning rate with $\eta^{(n+1)} \leftarrow 0.9\eta^{(n)}$ in each iteration to facilitate convergence. We set the initial learning rate for $\boldsymbol{\mu}$ as 0.2 and the initial learning rate for $\boldsymbol{A}$ as 0.001. We set the momentum weighted $\beta = 0.8$

Following Algorithm 4, for CMA-ES-rank-μ, we set $\eta_{\boldsymbol{\mu}} = \eta_{\boldsymbol{C}} = 0.2, \eta_{1,\boldsymbol{C}} = 0.2$, and $\eta_{2,\boldsymbol{C}} = 0$. For CMA-ES-Evolution-Path, we set $c_c = 0.3$, $\eta_{\boldsymbol{\mu}} = \eta_{\boldsymbol{C}} = 0.2, \eta_{1,\boldsymbol{C}} = 0$, and $\eta_{2,\boldsymbol{C}} = 0.3$. For CMA-ES, we set $c_c = 0.3$, $\eta_{\boldsymbol{\mu}} = \eta_{\boldsymbol{C}} = 0.2, \eta_{1,\boldsymbol{C}} = 0.2$, and $\eta_{2,\boldsymbol{C}} = 0.3$. For ES, NES, and CMA-ES, we set the utilities $u_i = \frac{1}{\lambda/2}$ if $i < \lambda/2$; otherwise, $u_i = 0$. For xNES, we adopt the implementation in https://github.com/chanshing/xnes/blob/master/xnes.py for Algorithm 3 and set $\eta_{\boldsymbol{\mu}} = 1, \eta_\sigma = 0.1$, and $\eta_{\boldsymbol{B}} = 0.01$. The population size λ is set to be proportional to the dimensionality of the function being optimized.

For the comparison of convergence, we plot the number of times we evaluate the function to maximize f against $-f(\boldsymbol{x}_i)$ evaluated from generated samples.

3.2 Rosenbrock function

Rosenbrock function is a popular black-box optimization benchmark function, defined as

$$R(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right]$$

We want to maximize $-R(\boldsymbol{x})$. We set $\lambda = 10d$ for all algorithms.

The results are compared in Figure 1. It is evident that only xNES and CMA-ES are able to successfully minimize the function, while the other methods fail to do so. Additionally, CMA-ES exhibits faster convergence compared to xNES. When d is large, only CMA-ES is effective in minimizing the Rosenbrock function. Furthermore, in scenarios with large d, NES slightly outperforms the other methods, with the exception of CMA-ES.

3.3 Ackley function

Ackley function is another popular benchmark function, defined as

$$A(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^d x_i^2}{d}}\right) - \exp\left(\frac{\sum_{i=1}^d \cos(2\pi x_i)}{d}\right) + 20 + \exp(1)$$

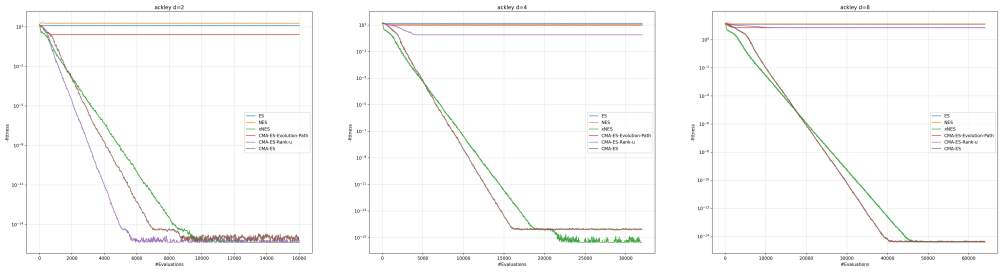

Figure 2: Ackley with $d = 2, 4, 8$. We plot the evaluation of all $A(\boldsymbol{x}_i)$ against the number of evaluations.

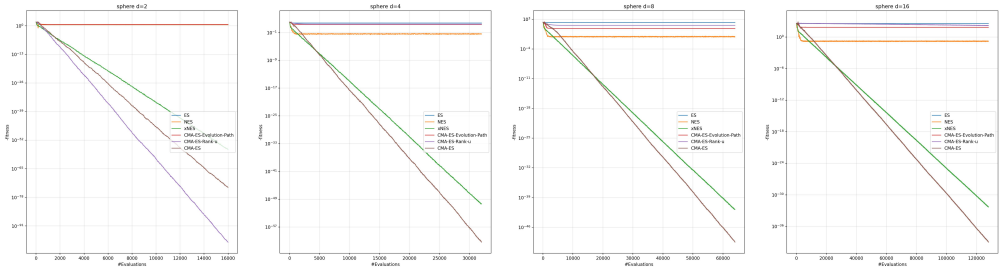

Figure 3: Sphere with $d = 2, 4, 8, 16$. We plot the evaluation of all $S(\boldsymbol{x}_i)$ against the number of evaluations.

We want to maximize $-A(\boldsymbol{x})$. We set $\lambda = 4d$ for all algorithms.

The results are reported in Figure 2. It is interesting to note that when $d = 2$, CMA-ES-rank-μ exhibits the fastest convergence. However, as d increases, the integration of evolution path becomes necessary in CMA-ES-rank-μ to achieve convergence. Once again, only xNES and CMA-ES are capable of minimizing the function, with CMA-ES showing faster convergence compared to xNES.

3.4 Sphere function

Sphere function is the last benchmark function we are going to minimize, defined as $S(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$. We want to maximize $-S(\boldsymbol{x})$. We set $\lambda = 4d$ for all algorithms.

We report the results in Figure 3. Similarly, when $d = 2$, CMA-ES-rank-μ exhibits the fastest convergence. However, as d increases, only xNES and CMA-ES (with both rank-μ updates and evolution path) are capable of minimizing the function, with CMA-ES demonstrating faster convergence compared to xNES. Moreover, unlike the Ackley or Rosenbrock function, xNES and CMA-ES continue to minimize the sphere function. It is possible to increase the learning rate. Additionally, it is once again observed that NES yields relatively higher fitness values, particularly in relatively higher dimensions.

3.5 Hyperparameter tuning

In real-world applications, hyperparameter optimization is a challenge as it involves black box optimization. One possible solution to this is the use of evolution strategies. For our study, we utilized the CMA-ES package [1] and Optuna [2], which is a hyperparameter optimization framework that includes a built-in

[1] https://github.com/CyberAgentAILab/cmaes

[2] https://github.com/optuna/optuna

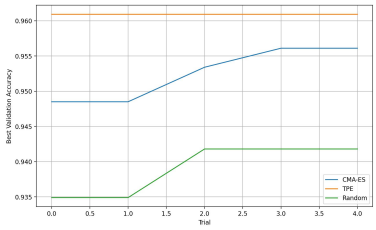

Figure 4: Optimization History. We plot best validation accuracy against number of trials.

CMA-ES sampler. In this report, we applied CMA-ES to tune a simple neural network for a multi-class classification task using the MNIST dataset [3].

The input layer has a size of 28×28, while the output layer has a size of 10. Our goal is to optimize the size of the single hidden layer and the learning rate. We compared the performance of the CMA-ES sampler with that of the RandomSampler and Tree-structured Parzen Estimator (TPE) Sampler. TPE is a Bayesian optimization approach that samples values from the space where better outcomes are more likely.

In Figure 4, we present the comparison of the best accumulated validation accuracy against the number of trials. The results demonstrate that the TPE Sampler is the most efficient, and CMA-ES enables us to efficiently identify the best samplers compared to random search.

4 Conclusion

In this report, we have observed that xNES and CMA-ES are effective solutions for black-box optimization problems. However, it is important to note that xNES is derived from the rigorous principle of natural gradients, whereas CMA-ES is based on heuristics. Furthermore, we have found that the combination of evolution path and rank-μ updates in CMA-ES leads to improved stability and convergence. Additionally, NES outperforms ES and CMA-ES without rank-μ updates by leveraging natural gradient. Finally, we applied CMA-ES to a simple hyperparameter tuning problem and concluded that this approach is valuable in real-world scenarios.

References

[GSY+10] Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 393–400, 2010.

[Han16] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[NS24] Masahiro Nomura and Masashi Shibata. cmaes: A simple yet practical python library for cma-es. *arXiv preprint arXiv:2402.01373*, 2024.

[Wen19] Lilian Weng. Evolution strategies. *lilianweng.github.io*, 2019.

[WSG+14] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

[YWSS09] Sun Yi, Daan Wierstra, Tom Schaul, and Jürgen Schmidhuber. Stochastic search using the natural gradient. In *Proceedings of the 26th annual international conference on machine learning*, pages 1161–1168, 2009.

[3] https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html