DSA4212:
Fast Natural Evolution Strategies vs Particle Swarm Optimization

April 18, 2024

Group Composition:

1. Sven Pontus Lindgren, A0276729L, e1133035@u.nus.edu
2. Calvin Christian Edward Lee, A0290096Y, E1324589@u.nus.edu
3. Javier Tham Jun Long, A0216040Y, E0538141@u.nus.edu
4. Peh Fang Ling, A0221977W, e0559509@u.nus.edu

1 Introduction

Most of Machine Learning seems to be based on gradient descent. But what can be done when the loss function is not differentiable? This study implements and explains a fast and simple version of Natural Evolution Strategies (NES) which is a special version of Evolutionary Strategies (ES). NES was introduced by [WSG+14]. Furthermore, the algorithm is compared with particle swarm optimization (PSO), another popular black-box optimization algorithm created with inspiration from biology. NES is also compared against ES as a validation. Note that the study used Copilot, but in a constructive manner. The PSO algorithm is not the focus of this study, the implementation has for example not been optimized to be fast.

2 Theory

Here the version of NES implemented and its theory is explained. Approximations we have made and ideas with using Natural Gradients are also communicated. It implements the most important part of [WSG+14], the natural gradient, to the best of our understanding, and one idea from [Wen19]. There are many more techniques mentioned in [WSG+14]. To understand NES a wide range of sources was used, mainly [Wen19], [Sur19], and Copilot. The implementation of PSO follows the canonical implementation. PSO does not use a gradient in any way. The best solution found for PSO is saved across iterations. For all the details about the implementations, see the provided GitHub.

2.1 The basics of Evolution Strategies

NES belongs to a larger family of algorithms, Evolutionary Strategies, that tries to optimize a distribution using a search gradient. We have decided to use the multivariate Gaussian distribution, so that sampled points $x \sim N(\mu, \Sigma)$ as that is what seems most canonical and familiar. To make the algorithm fast, more generalizable to high dimensional problems, and conceptually easier it was decided to approximate Σ as being diagonal.

The algorithm may not be so similar to actual evolution (there are no individuals, just a distribution). One might be able to understand the distribution as a species where μ is the average individual and Σ is a measure of the genetic diversity of the species. The algorithm consists of generating N individuals according to the distribution and then calculating new (μ, Σ) based on the individuals' relative fitness (where fitness, in evolution something to be maximized, is related to the function being optimized). These two steps are repeated.

2.2 Math details for Evolution Strategies

The setting is limited to unconstrained optimization of $f(x)$, here assumed to be a minimization problem. The goal of the algorithm is to find the set of parameters θ that optimizes the expected fitness of samples sampled according to the distribution, $F(\theta) = \mathbb{E}_{x \sim p_\theta(x)}[f(x)]$. The end goal is to ideally have $\mu = \text{argmin} f(x)$, this is done by optimizing in θ-space, not sample space. $f(x)$ is not assumed to be differentiable (otherwise just apply gradient descent) but $p_\theta(x)$ is, given that it is modeled as a multi-dimensional Gaussian. By reframing an optimization problem as finding the distribution that minimizes the expected value of the loss function one can get around the loss function not being differentiable! Now gradient descent can be applied (in parallel for (μ, Σ))

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} F(\theta_t). \tag{1}$$

Omitting some algebra it can be shown that [Wen19] $\nabla_\theta F(\theta) = \mathbb{E}_{x \sim p_\theta(x)} \left[\nabla_\theta \log p_\theta(x) f(x))\right]$. We are sampling the expectation value with a finite amount of samples N, therefore

$$\nabla_\theta \hat{F}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(x_i) f(x_i). \tag{2}$$

For a multivariate Gaussian doing quite some vector calculus, it can be shown that

$$\nabla_\mu \log p_\theta(x) = \Sigma^{-1}(x - \mu) \tag{3}$$

and (a matrix derivative)

$$\frac{\partial}{\partial \Sigma} \log p_\theta(x) = -\frac{1}{2} \left(\Sigma^{-1} - \Sigma^{-1}(x - \mu)(x - \mu)^T \Sigma^{-1}\right). \tag{4}$$

The formulas were generated with Copilot but verified against [WSG+14]. Equation 4 requires matrix inversion of Σ, a square matrix with the same dimensionality as x. Not great at all for memory and computational complexity meaning it is reasonable to approximate Σ as being diagonal. In the low-dimensional case explored here it is not strictly necessary but for the algorithm to be generalizable to high-dimensional problems it is, unless approximated in some other way. It can with this approximation be derived from equation 3 with $z_i \sim N(0,1)$ that

$$\frac{\partial}{\partial \mu_i} \log p_\theta(x) = \frac{x_i - \mu_i}{\sigma_i^2} = \frac{z_i}{\sigma_i} \tag{5}$$

and derived that

$$\frac{\partial}{\partial \sigma_i} \log p_\theta(x) = -\frac{1}{\sigma_i} + \frac{(x_i - \mu_i)^2}{\sigma_i^3} = -\frac{1}{\sigma_i} + \frac{z_i^2}{\sigma_i}. \tag{6}$$

These equations can easily be formulated in terms of the vectors z, σ.

Now the only thing needed to have the full Evolution Strategy algorithm is to plug in the analytic gradients concerning the different parameters into equation 2 and apply gradient descent (equation 1) to update θ starting from a θ_0.

2.3 The Natural Gradient

The main difference between Natural Evolution Strategies and any Evolution Strategy algorithm is the use of a preconditioning matrix, which in the NES case means the θ update is done using

$$\theta_{t+1} = \theta_t - \eta \mathbf{F^{-1}}_{\theta_t} \nabla_{\theta_t} F(\theta_t). \tag{7}$$

$\mathbf{F}_{\theta_t}$ is the Fisher Information matrix defined by $\mathbf{F}_{\theta_t} = \mathbb{E}_{x \sim p_\theta(x)} \left[(\nabla_\theta \log p_\theta(x))(\nabla_\theta \log p_\theta(x))^T \right]$.

It is easy to find an analytical formula for $\mathbf{F}_{\theta_t}$ for a diagonal Σ. It is possible to prove that $\mathbb{E}_{x \sim p_\theta(x)}[z_i^4] = 3$ and it is well known that for independent z_i, z_j $(i \neq j)$ which we have in this case $\mathbb{E}_{x \sim p_\theta(x)}[z_i^2 z_j^2] = \mathbb{E}_{x \sim p_\theta(x)}[z_i^2]\mathbb{E}_{x \sim p_\theta(x)}[z_i^2] = 1$. Doing some algebra leads to diagonal matrixes where $[\mathbf{F}_\mu^{-1}]_{ii} = \sigma_i^2$ and $[\mathbf{F}_\sigma^{-1}]_{ii} = \frac{\sigma_i^2}{2}$.

Doing this instead of sampling is beneficial in a lot of ways. There is no need to solve a linear system of equations to obtain the direction of change $\mathbf{F}_{\theta_t} d_t = \nabla_{\theta_t} F(\theta_t)$. $\mathbf{F}_\mu^{-1}$ is also diagonal, meaning it can be stored as a vector (less memory requirement) and multiplied elementwise with $\nabla_{\theta_t} F(\theta_t)$. This means there is no part of the algorithm that does not have linear complexity in the number of input dimensions unless the number of individuals that need to be sampled to approximate the gradient scales more than linearly with the dimension. NES will have the same complexity as ES. Note also that the preconditioning matrixes make intuitive sense as high σ_i^2 should signify certainty that a direction is worth exploring, leading to a possibility to have a higher learning rate in that direction (even though the $1/2$ is hard to explain). A final note is that sampling $\mathbf{F}_{\theta_t} = \mathbb{E}_{x \sim p_\theta(x)} \left[(\nabla_\theta \log p_\theta(x))(\nabla_\theta \log p_\theta(x))^T \right]$ where one σ_i is much bigger than the others seems to yield a very much non-diagonal matrix.

2.3.1 The analytical problem of the Fisher Information Matrix for general Σ

In addition to the mentioned practical benefits that follow from approximating Σ as diagonal, is an analytical problem. The simple derivation which leads to

$$\mathbf{F}_{\theta_t} = \mathbb{E}_{x \sim p_\theta(x)} \left[(\nabla_\theta \log p_\theta(x))(\nabla_\theta \log p_\theta(x))^T \right] \tag{8}$$

does not hold for $\theta_t = \Sigma_t$. One step in the derivation given by [Wen19] is to approximate ($d = \theta_{t+1} - \theta_t$, $\theta = \theta_t$)

$$\log p_{\theta+d}(x) \approx \log p_\theta(x) + \nabla_\theta \log p_\theta(x) d + \frac{1}{2} d^\top \nabla_\theta^2 \log p_\theta(x) d. \tag{9}$$

For $\theta_t = \Sigma_t$, $\nabla_\theta \log p_\theta(x) d$ is a matrix while $\log p_\theta(x)$ is a scalar, meaning the identity cannot hold. There seems to be a way to find FIM for multivariate Gaussians using "exponential" local coordinates, see [WSG$^+$14], but we feel that leads to a conceptually much harder algorithm. Considering the already mentioned problem of having to calculate Σ^{-1} it was decided to force Σ to be diagonal. This leads to the description of the genetic diversity of the species being an approximation, something that leads to disadvantages. Intuitively a diagonal multinormal Gaussian implies a sensitivity to the rotation of the coordinate system as it is an untilted ellipse. Hence it would for example not be able to as effectively as the full Σ adjust to a valley shape making a 45-degree angle against the two coordinate axes.

2.3.2 Mathematical motivation of the Fisher Information Matrix

The Fisher Information matrix preconditioning is derived from a much more general idea, the steepest descent. The principle is to take a step in the direction that minimizes the cost function the most. But how long should the step be? It depends on how we measure length. One idea is to minimize the linear approximation of the cost function starting from θ_t regularized so that the distance d between θ_t, θ_{t+1} is small enough for the linear approximation to be reasonable. Note that this does not require the problem to be convex. This means in this case (assuming diagonal Σ) that

$$\theta_{t+1} = \text{argmin}_\theta [F(\theta_t) + \nabla_\theta F(\theta_t)(\theta - \theta_t)^T + D(\theta, \theta_t)]. \tag{10}$$

This leads to the standard gradient descent by using the Euclidean distance as D. For distributions, a reasonable choice is to use the KL divergence. KL divergence can easily be shown to be invariant under any reparametrization, not just under affine transformations as Newton's method [Sur19]! Unfortunately, equation 10 is intractable so approximations must be made meaning that there is to the best of our understanding no guarantee of invariance. By doing a Taylor series expansion to the second order one obtains [Wen19],

$$\text{KL}[p_{\theta_t} \| p_{\theta_{t+1}}] \approx -\mathbb{E}_{x \sim p_{\theta_t}(x)} [\nabla_\theta \log p_\theta(x)] d - \frac{1}{2} d^\top \mathbb{E}_x [\nabla_\theta^2 \log p_\theta(x)] d. \tag{11}$$

The first term can (easily) be shown to be 0 while $\mathbf{F}_{\theta_t} = \mathbb{E}_x[\nabla_\theta^2 \log p_\theta(x)]$. The usual way to proceed is to enforce the constraint that $D(\theta, \theta_t)$ is small and use Lagrangian multipliers [Wen19], [Sur19]. Doing this one obtains that $\mathbf{F}_{\theta_t}^{-1}$ should be used as a precondition matrix.

2.4 Minor Ideas explored

Many things were considered briefly such as $\Sigma \approx \sigma I$, learning rate schedule, momentum and first doing NES and then some ES. One minor idea kept with inspiration from [Wen19] was to update θ_t using elitism for well-conditioned cases (otherwise a high risk of instability). That means that only the sampled top 25 % of individuals concerning fitness were considered when calculating the new species distribution.

3 Method

For the benchmarking [FDG+12] was used. It contains a multitude of well-known functions suitable for benchmarking. To limit the scope of the project single objective functions were used. The benchmarking functions were Sphere (unimodal, well conditioned), Schaffer (multimodal, well conditioned), Rastrigin (multimodal, not well-conditioned), and the Rosenbrock (unimodal, very ill-conditioned) function. All are minimization problems. To visualize and save computation time only 2-dimensional inputs were used.

The benchmark is done with the same initial solution for ES, NES and PSO, obtained by dividing an area into 16 squares of equal size and sampling uniformly one value from each square. $[-1.5, 1.5]^2$ was used. σ was initialized to $[0.25, 0.25]$, something that should be a reasonable prior given the split of $[-1.5, 1.5]^2$. σ should not be sampled unsymmetrically as it might lead to a strong initial preference for one direction, making NES behave very badly!

ES, NES and PSO were run for the same fixed number of iterations and evaluated against the sum of the losses (lower is better) for each problem and initial condition. The time required is also measured but not focused upon. Some qualitative analysis is done by showing loss(iteration) and the optimization landscape. The same amount of particles and ”individuals” was sampled.

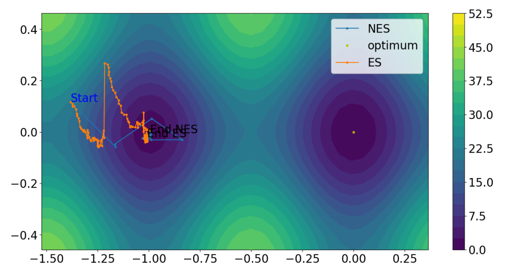
The tuning of hyperparameters (inertia, social, and cognitive weight for PSO; learning rate for NES, ES) was done by finding a case that works reasonably well for the Sphere function and adjusting that for the other functions if it leads to too much divergence. For PSO it was possible to use the same hyperparameters for all functions, for ES, NES the learning rate was adjusted separately for the two not well-conditioned cases of Rosenbrock and Rastrigin. The learning rate of NES and ES was not set as the same. Considerable effort was spent trying to find somewhat optimal parameters for NES and ES, especially to explore if ES could perform as well as NES for the ill-conditioned cases.

4 Results

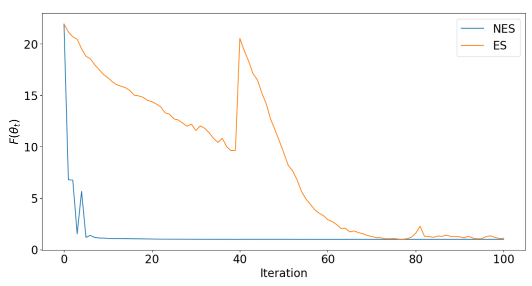
Table 1 shows the mean and standard deviation of the sum of losses over all iterations (a measure that balances speed and performance) for many different initial solutions. The result is for specific seed and learning rates but is in our experience representative given the standard deviation. There are variations depending on the learning rate and seed, especially for the Rosenbrock function. Note that neither ES, NES nor PSO diverged in this case. Note that the best solution found is stored for PSO, as such it makes sense that it does not diverge. The results seem to indicate that

Name	ES	NES	PSO
Sphere	61.11 ± 219.84	7.21 ± 5.27	2.02 ± 1.41
Rosenbrock	1260.89 ± 1662.74	417.25 ± 242.62	242.71 ± 206.30
Schaffer	161.17 ± 72.13	12.65 ± 13.12	13.56 ± 10.72
Rastrigin	882.10 ± 774.35	175.38 ± 93.01	235.22 ± 103.17

Table 1: Mean and standard deviation for the loss function



(a) Optimization landscape



(b) Loss by itertion

Figure 1: Qualitative comparison of NES and ES

NES generates slightly worse results than PSO but a significant improvement against pure ES, especially for the ill-conditioned benchmarks.

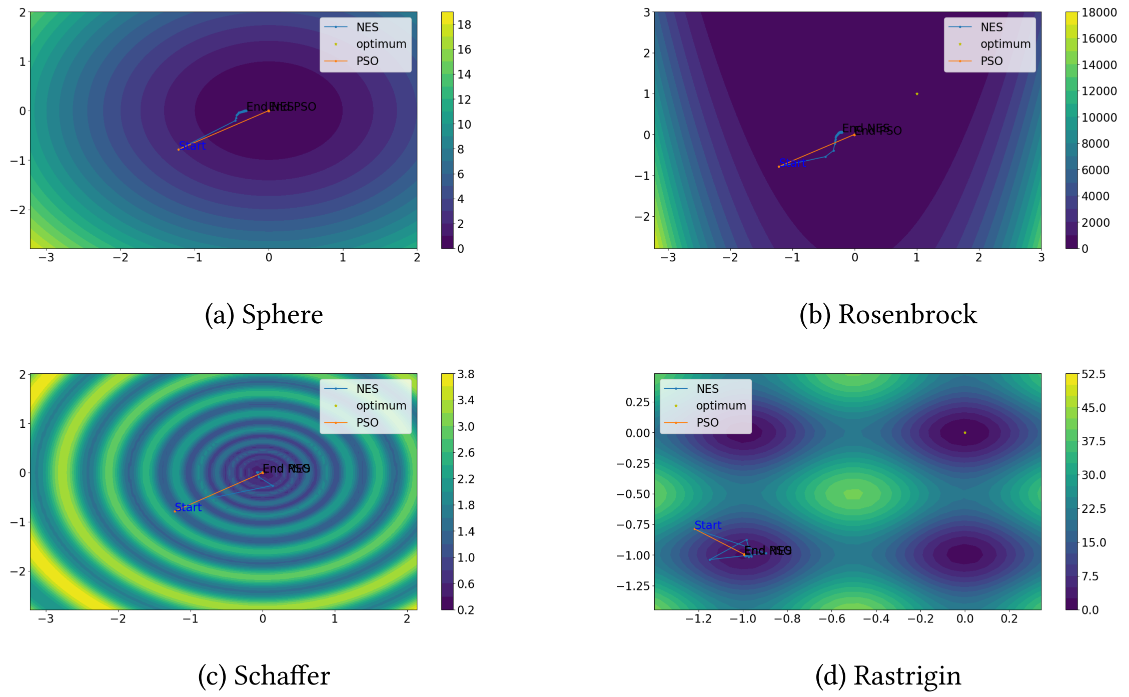
4.1 Qualitative comparasions

Figure 1 compares the optimization process for NES against ES for the Rastrigin function for the same seed as table 1. The figure seems to indicate that NES has a much better feel for the curvature than ES, just as expected. The problem for ES seems to be that it starts to oscillate in the y-direction, something which might be due to the non-circular shape.

Figure 2 shows one initial condition per benchmarking function for the same seed as table 1 of NES against PSO. What should be noted is that PSO seems to beat NES, that NES might have a somewhat too high learning rate for Schaffer and Rastrigin, and that both algorithms seem to be handling the benchmark functions rather well except for the Rastrigin case where they go to the nearby local minima. The curving of NES in the Rosenbrock case seems appropriate.

4.2 Speed

The mean time and standard deviation per initial condition and benchmarking function for the laptop used are given by table 2 for NES and PSO. ES is very similar to NES. The table shows that the implementation of NES seems to be faster than the implementation of PSO given the same amount of iterations and particles or "individuals" sampled. Given that little effort has been spent on optimizing the speed of PSO (nor NES) this makes no guarantee of NES being faster in general.



(a) Sphere

(b) Rosenbrock

(c) Schaffer

(d) Rastrigin

Figure 2: Figure showing a comparison of NES and PSO for the different benchmarking functions.

Algorithm	Speed
NES	0.06 ± 0.01
PSO	0.15 ± 0.02

Table 2: The time in seconds

5 Discussion

Considering the standard deviation and dependency on seed and learning rate it seems like PSO might slightly outperform the fast version of NES implemented with the benchmark used. It seems likely that NES has been implemented correctly as it beats ES by having a better feel for the curvature as expected. On the other hand, PSO saves the best solution seen, something NES does not. Note, however, that considerable effort was spent tuning the learning rate of NES while the learning rate that worked well for the Sphere case for PSO worked well for all cases. That is a significant advantage of PSO, especially that there seems to be little risk of divergence. Tuning the hyperparameters of PSO for the different benchmarking functions might lead to improved results. NES might however be slightly faster, even though there has been no exploration of how the number of "individuals" and particles scale with the dimension of the problem.

A limitation of this approach is that only initial positions rather close to the global minima are considered, starting further away might lead to other results. The study was also limited to a constant learning rate to keep the hyperparameter space for NES and ES manageable, something that might improve the ES and NES results.

There seems to be a dimensionality problem for PSO, meaning that a high dimensional benchmark might lead to superior performance for NES. [Old17] finds that the canonical version of PSO performs very well even for 10-dimensional problems. When changing to 1000-dimensional problems PSO performs many orders of magnitude worse. Furthermore [Old17] investigates approaches to fix this problem and finds none to be satisfactory. Normal gradient descent is always able to find a local minimum no matter the dimension. As such it does not seem likely that we will be using PSO to train neural networks anytime soon. NES might also have somewhat of a scaling problem as more ”individuals” likely must be sampled in high dimensional spaces. Benchmarking against high-dimensional problems would be a natural continuation of this study.

Interestingly, even such a simple thing as scaling each direction of the gradient by the corresponding standard deviation, the only difference between the versions of NES and ES implemented, seems to lead to a significant improvement in performance. Using the full Σ or techniques implemented in [WSG+14] might produce even more performance but potentially for a higher cost in terms of time and memory.

6 Conclusion

This study indicates that Particle Swarm Optimization might be preferable over the fast version of Natural Evolution Strategies implemented for low dimensional problems. There is significant uncertainty due to hyperparameters and randomnes. Interestingly, the approximations made to the full NES algorithm seem to still yield significantly better performance than ES, despite carrying a very small cost in terms of time and memory.

References

[FDG+12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[Old17] Elre Talea Oldewage. *The perils of particle swarm optimisation in high dimensional problem spaces*. University of Pretoria (South Africa), 2017.

[Sur19] Sylesh Suresh. Natural gradient. February 2019.

[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.