algorithms

MDPI

Article

Enhancing Performance of Evolutionary Strategies with Symmetric Sampling (Furthermore, Weight Decay)

Paolo Pagliuca

Institute of Cognitive Sciences and Technologies (ISTC), National Research Council (CNR), 00185 Rome, Italy
paolo.pagliuca@istc.cnr.it

Abstract

Evolutionary Strategies (ESs) are optimization metaheuristics largely adopted in Evolutionary Computation (EC). Since their introduction in the early 70s, researchers in the field have attempted to improve the efficacy of these algorithms. The most advanced ESs, such as the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) and Exponential Natural Evolution Strategies (xNESs), make use of covariance matrices storing relationships between parameters to be optimized, which enable the algorithms to fasten the search in the solution spaces. However, the computational cost of calculating covariance matrices linearly scales with the number of parameters. Recently, the OpenAI Evolutionary Strategy (OpenAI-ES) emerged as an effective ES in different domains, thanks to the parameter information stored in two momentum vectors. Furthermore, OpenAI-ES gains an advantage from the usage of symmetric sampling and weight decay techniques. In this work, I delve into the application of symmetric sampling and weight decay on CMA-ES, xNES and Separable Natural Evolution Strategies (sNESs), with the aim to improve their performance in domains in which they get stuck in local minima outcomes. Specifically, I propose three novel variants for each ES and verify their efficacy with respect to the PyBullet halfcheetah and hopper robot locomotion problems, and two collective tasks (i.e., swarm aggregation and swarm foraging). The findings reveal that symmetric sampling produces performance enhancements in all the domains, whereas the effect of weight decay varies across the considered problems. Furthermore, symmetric sampling allows ESs to keep parameter size limited, which is paramount in these scenarios. This research identifies techniques enhancing the success of modern ESs, proposes several ES variants, and discusses the relationship between algorithmic performance and task properties.

Keywords: Evolutionary Strategies; symmetric sampling; weight decay; robot locomotion; swarm robotics

Academic Editor: Thomas Hanne

Received: 19 May 2026
Revised: 17 June 2026
Accepted: 22 June 2026
Published: 23 June 2026

1. Introduction

Evolutionary Strategies (ESs) [1–4] are a subclass of Evolutionary Algorithms (EAs) [5], which are widespread optimization methods in Evolutionary Computation (EC). As originally defined in [3], ESs make use of mutation, recombination (also termed as "crossover") and selection operators as Genetic Algorithms (GAs) [6]. Depending on the selection operator used, a canonical ES can be expressed as in either (a, b) or $(a + b)$ (see Equations (1) and (2)).

$$(a, b) - ES : selectBest(a)\ from\ b \tag{1}$$

$$(a + b) - ES : selectBest(a)\ from\ a \cup b \tag{2}$$

Specifically, given μ parents and λ offspring, two selection mechanisms have been defined:

1. (μ, λ): The best μ offspring are retained for the next iteration of the algorithm;
2. $(\mu + \lambda)$: Selection of the best is made through the union of μ parents and λ offspring.

The original ES is a $(1 + 1)$-ES, i.e., the population is formed by one parent ($\mu = 1$) and one offspring ($\lambda = 1$). If the performance (also called "fitness") of the offspring is at least as good as the parent's fitness, the offspring becomes the new parent. The $(1 + \lambda)$-ES algorithm considers λ offspring generated by the parent through mutation and/or crossover operators. If one of the offspring is at least as good as the parent, it becomes the new parent. On the other hand, in the $(1, \lambda)$-ES algorithm, the best offspring becomes the new parent regardless of the performance, and the parent is always discarded [7].

Based on this theoretical foundation, several variants have been proposed over time. A tightly related method is the Hill-Climbing (HC) algorithm [8], which considers μ parents evolving independently of each other through a $(1 + 1)$-ES procedure. The advantage of considering a pool of independent parents is to increase the chance to discover more effective solutions. For example, in [9] the authors show that HC succeeds in solving the rather complex 5-bit parity problem [10]. However, the convergence speed is generally slow and the resulting performance might be sub-optimal, particularly in domains like pole balancing [9], test function optimization [11] and robot foraging [12]. The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [13] is a rather sophisticated ES, whose main property lies in the computation of a covariance matrix. In particular, the mutual influence between parameters is encoded in the matrix and allows one to direct the search in the solution space. As a consequence, CMA-ES leverages the covariance matrix to minimize the time spent to discover solutions (exploration) and maximize the performance refinement (exploitation). In the original formulation reported in [13], CMA-ES evolves a single solution, referred to as a "centroid", thus representing a particular instance of $(1 + \lambda)$-ES. At each iteration, the algorithm generate samples from a Gaussian distribution $\mathcal{N}(0, 1)$. The samples are modulated by the covariance matrix and added to the centroid to create a set of candidate solutions, which are evaluated and ranked based on their performance. The centroid is then updated based on the recombination of the best $\frac{\lambda}{2}$ samples. CMA-ES has been firstly applied to test function optimization [13]. Thanks to its features, it became a popular tool to address other types of problems, such as pole balancing [14,15] and robotics [12,15,16]. Nonetheless, CMA-ES showed poor performance in complex tasks like legged robot locomotion: as shown in [17], CMA-ES performs worse than OpenAI-ES with regard to halfcheetah, hopper and walker2D tasks (see Figure 1 in [17]). Closely related to CMA-ES is the Exponential Natural Evolution Strategy (xNES) algorithm [18,19]: the method is a particular $(1 + \lambda)$-ES that calculates a covariance matrix storing the relationship between parameters, which is exploited to accelerate and optimize the search for solutions. According to [19,20], both CMA-ES and xNES explore the solution space based on the natural gradient [21]. Similarly to CMA-ES, at each iteration of xNES, a pool of samples is randomly drawn from a Gaussian distribution $\mathcal{N}(0, 1)$. A set of candidate solutions is derived by multiplying samples and covariance matrix, and adding the resulting vectors to the centroid. These candidate solutions are evaluated and ranked based on their performance. However, differently from CMA-ES, xNES uses all the evaluated samples for recombination and centroid updating. A more detailed description is provided in Section 2. The effectiveness of xNES has been proved in test function optimization [18,19], pole balancing [15,19,22], robotics [12,15,23] and chemistry [24]. As for CMA-ES, xNES is uncapable of achieving good performance in robot locomotion tasks [17], in which it obtains remarkably lower performance than OpenAI-ES in the halfcheetah, hopper and walker2D tasks (see Figure 1 in [17]). A notable drawback of CMA-ES and xNES is their

computational complexity, which increases non-linearily with the number of evolvable parameters. The Separable Natural Evolution Strategy (sNES) [19,25] is another instance of Natural Evolution Strategies (NESs) that makes use of separable search distributions. Specifically, instead of calculating the full covariance matrix, each parameter variance is computed and stored independently. This is particularly useful in large search spaces. Therefore, the sNES algorithm scales better with the number of evolvable parameters, and reduces the computational burden of CMA-ES and xNES. In fact, depending on the number of parameters to be optimized, computing the covariance matrix might be expensive or even impossible [17,19,25–27]. Similarly to CMA-ES and xNES, sNES is a variant of $(1+\lambda)$-ES that extracts a set of samples from a Gaussian distribution $\mathcal{N}(0,1)$ at each iteration. The samples are multiplied by the variances and added to the centroid, thus forming a pool of candidate solutions. These solutions are evaluated and ranked based on their performance. Lastly, the centroid is updated through the recombination of all the samples, similarly to xNES. Examples of the application of sNES can be found in [11,15,17,19,25]. The OpenAI Evolutionary Strategy (OpenAI-ES) algorithm [28] is a modern ES introduced as a viable alternative to Reinforcement Learning (RL) approaches [29], thanks to an efficient parallelization procedure. OpenAI-ES is another instance of $(1+\lambda)$-ES: at each iteration of the algorithm, a pool of samples is extracted from a Gaussian distribution $\mathcal{N}(0,\sigma)$. OpenAI-ES uses symmetric sampling [30] to evaluate both positive and negative parameter modifications. Once all the samples have been evaluated and get a performance value (fitness), the algorithm ranks samples and recombines them to estimate the fitness gradient. Lastly, OpenAI-ES updates two momentum vectors (mean and variance) and the centroid through Adam [31]. A main advantage of OpenAI-ES over CMA-ES and xNES is the limited computational burden, since no covariance matrix has to be computed. This also leads to a better scalability when the number of evolvable parameters increases. A schematic of OpenAI-ES is provided in Figure 1. The authors in [28] showed that OpenAI-ES is capable to achieve good performance in MuJoCo [32] humanoid and Atari games [28]. OpenAI-ES demonstrates a wide capability with respect to MuJoCo and PyBullet [33] robot locomotion [17,34], collective problems [17,35] and many other domains [36,37]. Differently from CMA-ES, sNES and xNES, OpenAI-ES stores two momentum vectors keeping track of how performance varied in order to drive the search in the solution space. Moreover, two main properties affect the performance of OpenAI-ES: (i) symmetric sampling [30] and (ii) weight decay [38]. The former allows the algorithm to understand the direction of parameter variations, thus better coping with the task specificity. The latter enables OpenAI-ES to keep the weights limited in size, which is beneficial for solving PyBullet and MuJoCo robot locomotion problems [17]. Nonetheless, recent studies emphasized that OpenAI-ES underperforms based on the problem characteristics [12]. In fact, the tendency to decrease weight size by OpenAI-ES is detrimental when using a sensory–motor neural network controller, as in [12]. Similarly, the performance of OpenAI-ES is strongly dependent on the hyperparameter settings, as illustrated in [35]. Differently from OpenAI-ES, CMA-ES and xNES suffer from uncontrolled weight growths, which limit their performance in MuJoCo and PyBullet robot locomotion domains [17,35]. Furthermore, the parameter variations are typically applied in one direction only. Therefore, I hypothesize that symmetric sampling and weight decay might be advantageous for enhancing the performance of CMA-ES, sNES and xNES in those scenarios in which they get stuck in sub-optimal results. In fact, analyzing parameter variations in two opposite directions could enable a better understanding of the actual parameters' influence on the overall performance. Furthermore, keeping weight size limited might allow for a better exploration of the search space, without skipping specific regions corresponding to potentially enhanced performance.

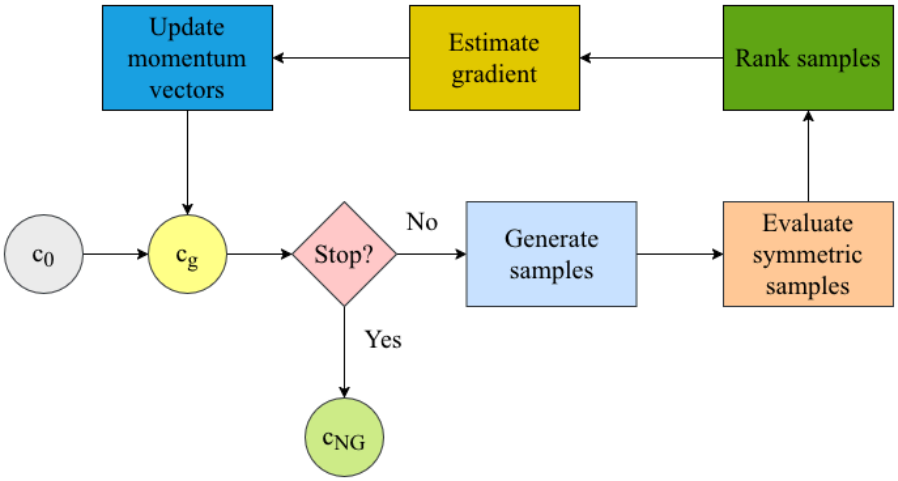


Figure 1. Schematic of OpenAI-ES.

Table 1 provides a summary of the main works utilizing ESs and the related application domains. Moreover, Table 2 provides a thorough comparison of CMA-ES, OpenAI-ES, sNES and xNES in terms of computational complexity, mechanisms to preserve information about parameter–performance relationships and scalability.

Table 1. List of research works on ESs. For each work, I highlighted the publication year, the authors (and the reference), the algorithm used and the application domain(s).

Year	Authors	Algorithm	Domain
1993	Back & Schwefel [39]	(μ, λ)-ES	Test function optimization
1998	Papadrakakis et al. [40]	(μ, λ)-ES, $(\mu + \lambda)$-ES	Structural optimization
2001	Hansen & Ostermeier [13]	CMA-ES	Test function optimization
2003	Igel [14]	CMA-ES	Pole balancing
2010	Glasmachers et al. [18]	xNES	Test function optimization
2011	Schaul et al. [25]	sNES	Test function optimization
2014	Wierstra et al. [19]	sNES, xNES	Pole balancing, Test function optimization
2017	Salimans et al. [28]	OpenAI-ES	MuJoCo robot locomotion, Atari games
2019	Pagliuca & Nolfi [15]	CMA-ES, sNES, xNES	Pole balancing, Car games, Swarm foraging
2020	Pagliuca, Milano & Nolfi [17]	CMA-ES, OpenAI-ES, sNES, xNES	MuJoCo robot locomotion
2020	Rais Martinez & Aznar Gregori [41]	CMA-ES, OpenAI-ES	Swarm aggregation
2022	Basilio & Goliatt [24]	xNES	Geopolymer self-compacting concrete
2022	Pagliuca & Vitanza [42]	OpenAI-ES	Swarm aggregation
2023	Pagliuca & Vitanza [43]	CMA-ES, xNES	Swarm aggregation
2024	El Saliby et al. [44]	$(\mu + \lambda)$-ES, CMA-ES	Test function optimization, 2D navigation, Voxel-based soft robot control

Table 1. *Cont.*

Year	Authors	Algorithm	Domain
2025	Pagliuca [11]	OpenAI-ES, sNES	Bit-parity, Grid navigation, Pole balancing, Test function optimization
2025	Pagliuca & Vitanza [35]	CMA-ES, OpenAI-ES & xNES	Swarm aggregation

Table 2. Analysis of the characteristics of CMA-ES, OpenAI-ES, sNES and xNES.

	CMA-ES	OpenAI-ES	sNES	xNES
Computational complexity	$O(d^2)$	$O(d)$	$O(d)$	$O(d^3)$
Exploration strategy	Sampling around centroid's parameter distribution	Sampling around centroid's parameter distribution	Sampling around centroid's parameter distribution	Sampling around centroid's parameter distribution
Parameters' importance attribution	Covariance matrix	Momentum vectors	Variance vector	Covariance matrix
Scalability	No	No	Yes	Yes

Motivated by these considerations, this research seeks to fill a gap in the sub-optimality of CMA-ES, sNES and xNES with regard to domains involving hinged robots that have to locomote on flat terrain, or groups of robots that have to cooperate and, possibly, communicate to solve collective tasks. To address this sub-optimality, I propose the incorporation of the techniques positively affecting the performance of OpenAI-ES in such domains. In particular, I delve into the application of symmetric sampling and weight decay into CMA-ES, sNES and xNES, with the aim to verify whether the incorporation of these techniques leads to enhanced performance of the considered ESs in relatively challenging domains. For each ES, I thus propose three variants (denoted for simplicity as "<ES>_s", "<ES>_wd" and "<ES>_swd", with <ES> being either CMA-ES, or sNES, or xNES), and I investigate whether or not they provide advantages over the original algorithms in terms of performance. In particular, my analysis focuses on four problems: (i) the halfcheetah PyBullet robot locomotion problem [45], (ii) the hopper PyBullet robot locomotion task [46], (iii) the swarm aggregation problem [35,43] and (iv) the swarm foraging task [15]. Moreover, as a proof-of-concept, I investigate the effects of applying symmetric sampling on the Rosenbrock function [47], a benchmark task for test function optimization. I choose these tasks because CMA-ES, sNES and xNES have already been tested on them [15,17,35,43], which enables a fair comparison of the outcomes with those reported in the literature. The main goal of this research is to provide a theoretical framework showing that ESs may benefit (in terms of enhanced performance) from the incorporation of mechanisms allowing for the assessment of parameter influence, like symmetric sampling and weight decay (when necessary).

The achieved results confirm the hypothesis and demonstrate that the application of symmetric sampling and weight decay corresponds to performance enhancements in CMA-ES, sNES and xNES. This is particularly evident with regard to the halfcheetah and swarm aggregation problems. Specifically, symmetric sampling improves algorithm performance irrespective of both the considered ES and the specific problem. On the other hand, the application of weight decay produces improved results on CMA-ES and sNES, while the effect on xNES depends on the particular task. Moreover, the analysis of the

parameter size indicates that keeping parameters limited in amplitude is paramount in the considered domains. Surprisingly, symmetric sampling produces a notable drop of the parameter size.

This research provides the following main contributions:

1. I investigate the effectiveness of transferring symmetric sampling and weight decay on three state-of-the-art ESs (i.e., CMA-ES, sNES, xNES), similarly to the operation of OpenAI-ES;
2. I propose three variants for CMA-ES, sNES and xNES, in which I apply either symmetric sampling, weight decay, or both;
3. I show that the new variants enhance algorithm performance in two PyBullet robot locomotion (i.e., halfcheetah and hopper) and two collective (swarm aggregation and swarm foraging) problems;
4. I demonstrate that keeping parameter size limited is paramount to improve performance in the considered domains, aligning this finding with [17,35].

The manuscript is organized as follows: Section 2 illustrates the problems and the ESs. The results are presented in Section 3 and further discussed in Section 4. In Section 5, I, provide my final remarks and future research pathways.

2. Materials and Methods

2.1. Rosenbrock Function Optimization

The first application scenario entails the optimization of the Rosenbrock function [47], which is a non-convex function commonly used to assess algorithmic optimization capabilities [48,49], including ESs [13,19]. Furthermore, it has been recently adopted to assess the efficacy of different algorithms [9,27,50], thus representing a benchmark task for the considered investigation. Given an input vector x of size d, the goal is to minimize the value of function F_{rosen}, which is defined in Equation (3).

$$F_{rosen} = \sum_{i=1}^{d-1} (100(x_{i+1} - {x_i}^2)^2 + (x_i - 1)^2) \tag{3}$$

An illustration of the Rosenbrock function is provided in Figure 2.

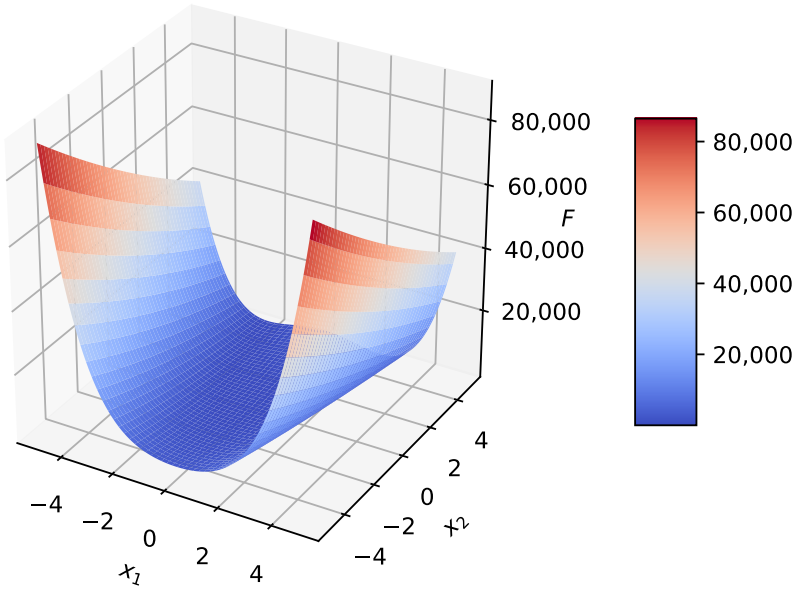
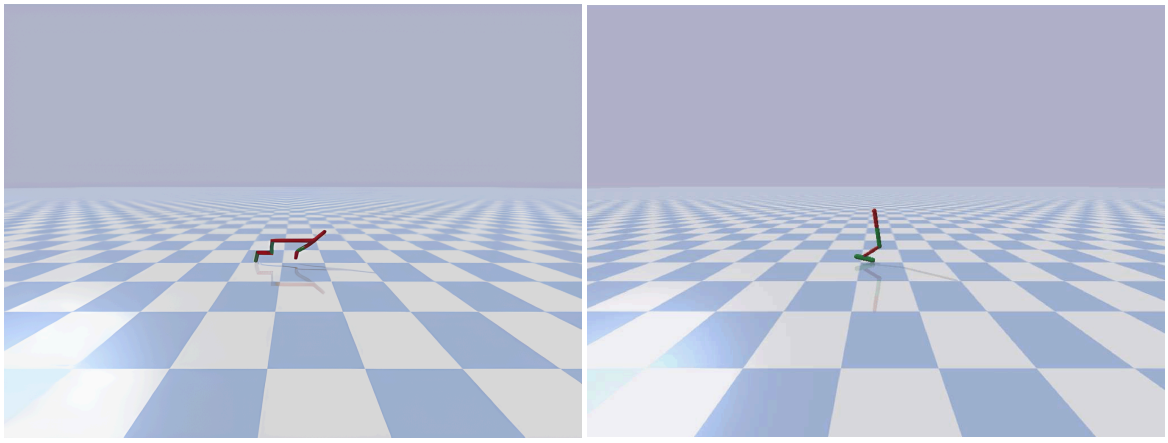


Figure 2. Illustration of the Rosenbrock function for a two-dimensional input $x = [x_1, x_2]$. The optimal value $F(x) = 0.0$ is achieved with $x = [1, 1]$.

In the experiments reported here, I considered an input vector x of size $d = 50$, which was chosen according to [11,27]. Algorithms sought to optimize F_{rosen} in 10^6 time steps. I performed 10 replications of the experiments. To ensure a fair comparison with existing studies, I adopted the same experimental settings employed in [9,11,27,50].

2.2. PyBullet Robot Locomotion Problems

The second and third problems belong to the family of PyBullet robot locomotion tasks [33]. Specifically, I selected the halfcheetah (Figure 3a) [45] and the hopper (Figure 3b) [46] problems. These two tasks are rather challenging for CMA-ES, sNES and xNES, since the behavior is highly affected by the parameter size, which will be shown in Section 3. Therefore, CMA-ES, sNES and xNES might encounter pitfalls in the search for solutions, due to their tendency to grow parameter size (see [12,17]). A similar outcome is reported in [17] with regard to MuJoCo locomotion problems.



(**a**) (**b**)

Figure 3. PyBullet robot locomotion problems: (**a**) Halfcheetah; (**b**) hopper.

Both problems involve a mobile legged robot that has to locomote toward a target location placed approximately at 1 km away from the robot's starting position. The robot can move only horizontally along the x-axis. The objective is to travel as far as possible. The performance is measured through the fitness functions defined in Equations (4) and (5):

$$F_{ha} = \sum_{i=1}^{N_{steps}} \left(\Delta p(i) - 0.1 \times \sum_{j=1}^{N_J} l_j(i) \right) \tag{4}$$

$$F_{ho} = \sum_{i=1}^{N_{steps}} \Delta p(i) \tag{5}$$

where $\Delta p(i)$ is the displacement of the robot at step i, N_{steps} denotes the number of steps and $l_j(i)$ is a flag that activates (i.e., $l_j(i) = 1$) when the joint j reaches its limits at step i. The latter component is adopted to discourage unnatural behaviors that might circumvent the physical limitations of the PyBullet engine.

The robots are controlled by a Multi-Layer-Perceptron (MLP) neural network [51] with one internal layer containing 50 neurons. I used this value since it has been largely employed in previous studies on PyBullet robot locomotion tasks [17,34]. The network has 26 inputs and 6 outputs in the case of the halfcheetah problem (see Table 3), whereas the number of inputs and outputs are, respectively, 15 and 3 for the hopper task (see Table 3).

Table 3. List of sensors (denoted with $s_{<num>}$) and motors (indicated with $m_{<num>}$) of the halfcheetah and hopper robots. The symbols represent the following variables: z is the robot height; z_{init} is the robot height when the episode begins; $angle_{to\_target}$ is the relative robot–target angle; v_x, v_y, v_z are the velocity components along the three axes; ϕ and θ represent the roll and pitch orientation angles of the robot; (p_j, v_j) denotes the position and velocity of the joint j; c_f is set to 1 when the foot f is on the ground, otherwise it is set to 0; and T_j indicates the torque applied to joint j. The number of joints N_J and the number of segments N_S is 6 for the halfcheetah and 3 for the hopper.

Parameter	Value
s_1	$z - z_{init}$
s_2	$\sin(angle_{to\_target})$
s_3	$\cos(angle_{to\_target})$
s_4	$0.3 \times v_x$
s_5	$0.3 \times v_y$
s_6	$0.3 \times v_z$
s_7	ϕ
s_8	θ
$s_9 - s_{9+N_J \times 2}$	$(p_j, v_j) \quad for j \in N_J$
$s_{9+N_J \times 2+1} - s_{9+N_J \times 2+1+N_S}$	$c_i \quad for\ i \in N_S$
$m_1 - m_{N_M}$	$T_j \quad for\ j \in N_J$

2.3. Collective Problems

The fourth and fifth problems belong to the swarm robotics domain [52–54], that is, the scenario in which a group of robots (defined a "swarm") cooperates with the aim to solve a problem that cannot be tackled by a single individual. More specifically, I delve into the swarm aggregation problem introduced in [35,43], and the swarm foraging task proposed in [15].

The swarm aggregation problem involves a swarm of 10 simulated MarXbot robots [55] living in a square walled arena of size 5 m $\times$ 5 m (see Figure 4a). The arena contains a nest (grey circle in Figure 4a), which represents a possible aggregation site. The robots' initial positions and orientations are randomly set at each episode. Similarly, the location of the nest is randomly initialized. The goal for the swarm is to reach a good level of aggregation in the environment. According to [35,43], I used the fitness function shown in Equation (6).

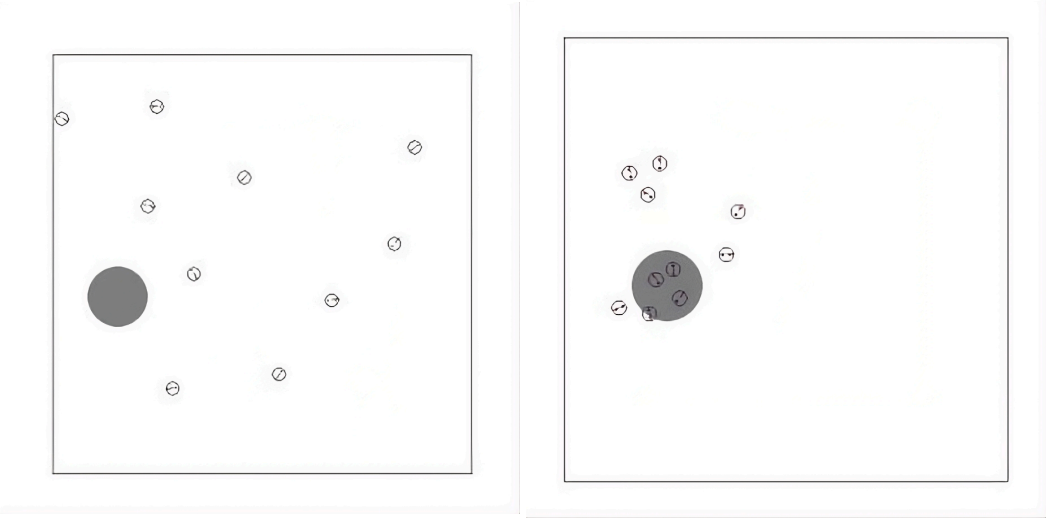
$$F_{sa} = \sum_{r=1}^{N_R} F_r \tag{6}$$

The fitness F_{sa} of the swarm is the cumulative sum of the fitness F_r achieved by each robot r, which is defined in Equation (7).

$$F_r = nest(r) + \left(1 - \frac{1}{N_R - 1} \sum_{o=1}^{N_R - 1} \frac{d_{ro}}{D}\right) \tag{7}$$

In Equation (7), $nest(r)$ is a bonus activation if the robot r is inside the nest ($nest(r) = 1$), d_{ro} is the distance between the robots r and o, and D is the diagonal of the arena (i.e., the theoretical maximum distance in the environment). I chose this fitness function since it represents the most complex version of the problem described in [35]. In fact, with this function, there is no explicit pressure towards reaching the nest or minimizing the distance with the mates, and ESs are free to discover any solution maximizing the fitness function F_{sa}. Clearly, different experimental settings (e.g., different values for $nest(r)$ and/or using a coefficient ≥ 1 for the distance component $\left(1 - \frac{1}{N_R - 1} \sum_{o=1}^{N_R - 1} \frac{d_{ro}}{D}\right)$) might affect the algorithmic performance, as illustrared in [35].

A fully recurrent neural network (RNN) [56,57] determines the robot behavior. In particular, the network has one internal layer of 10 neurons, a value derived from the works in [35,42,43,58]. Sensory information is retrieved by 8 InfraRed (IR) units (each one encoding the average value of 4 adiacent sensors), 4 units encoding the information perceived by a 90° Field-Of-View (FOV) frontal linear camera, and 2 units from the ground sensors. The network outputs encode the left and right wheel speeds, and whether or not the frontal-red Light-Emitting Diode (LED) and the rear-blue LED are active. A summary of the sensors and motors is provided in Table 4. The network is shared between the robots.



(**a**) (**b**)

Figure 4. Collective problems: (**a**) Swarm aggregation; (**b**) swarm foraging.

The swarm foraging problem shares the same environmental features of the previous task (number and type of robots in the swarm, size of the arena, presence of the nest). However, the nest represents the source location for the agents, which are initially located randomly inside it. Moreover, the robots have a battery that reduces through time, which has to be periodically recharged in the nest. As in [15], I consider 400 invisible food items in the arena, and the robots must transport the items to the nest in order to forage them. The objective of the swarm is to collect and forage the the highest number of food items in the environment. To this end, I used the fitness function in Equation (8):

$$F_{sf} = \sum_{r=1}^{N_R} n_F(r) \tag{8}$$

where $n_F(r)$ is the number of food items successfully foraged by the robot r.

Similarly to the former problem, a fully recurrent neural network is used to control the robot behaviors. The network architecture is almost identical to the previous problem, except for the addition of one input encoding the battery level and one bias input (see Table 4). Furthermore, in this case, the neural network is shared among the robots.

Table 4. List of sensors (denoted with $s_{<num>}$) and motors (indicated with $m_{<num>}$) used in the collective problems.

	Swarm Aggregation	**Swarm Foraging**
$s_1 - s_8$	IRs	IRs
$s_9 - s_{12}$	Camera	Camera
$s_{13} - s_{14}$	Ground	Ground
s_{15}	-	Battery
s_{16}	-	Bias
$m_1 - m_2$	Wheel speeds	Wheel speeds
$m_3 - m_4$	LEDs	LEDs

2.4. Evolutionary Strategies

This study focuses on three state-of-the-art ESs: Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [13], Separable Natural Evolution Strategies (sNESs) [19,25] and Exponential Natural Evolution Strategies (xNESs) [18,19]. I selected these methods since they have already been used to discover solutions for the Rosenbrock function [13,19] and the swarm foraging task [15,17]. Furthermore, they showed poor performance in robot locomotion [17] and obtained sub-optimal results in the swarm aggregation problem [35,43]. The algorithms are illustrated in Algorithms A1–A3 in Appendix A. Despite the availability of existing variants, I considered the original algorithms introduced in [13,18,19,25].

Both CMA-ES and xNES use a covariance matrix to identify relationships between parameters enabling to guide the search in the solution space towards the regions that are likely to correspond to enhanced performance. The two methods are tightly linked, as indicated in [19,20]. In particular, both ESs follow the "natural gradient" [21], which is computed by measuring the Kullback–Leibler (KL) divergence [59] between two parameter distributions and penalizes high variances between the distributions (i.e., uncertainty). A mathematical definition of the natural gradient is provided in Equation (9):

$$\tilde{\nabla}_\theta L = F(\theta)^{-1} \times \nabla_\theta L \tag{9}$$

where $\tilde{\nabla}_\theta L$ is the natural gradient of the loss/objective function with respect to parameters θ, $\nabla_\theta L$ denotes the plain gradient, and $F(\theta)^{-1}$ is the inverse of the Fisher Information Matrix (FIM) defined in Equation (10).

$$F = \mathbb{E}\left(\nabla_\theta \log \pi(x \mid \theta) \times \nabla_\theta \log \pi(x \mid \theta)^T\right) \tag{10}$$

In Equation (10), the symbol $\mathbb{E}$ states the expected value, $\pi()$ is the policy to be optimized and x indicates a generic sample.

Compared to the plain gradient, which is based on the Euclidean distance, the natural gradient is independent of the distribution parameterization and represents a more reliable update direction [19]. For further details about the natural gradient, the readers are referred to [19,21,60].

The efficacy and versatily of CMA-ES and xNES have been demonstrated in different domains, such as test function optimization [13,16,19], pole balancing applications [15,16] and robotic scenarios [23,35,61]. However, calculating the covariance matrix can be computationally expensive, which prevents CMA-ES and xNES from being used in high-dimensional parameter spaces [19,25–27]. Specifically, CMA-ES and xNES do not scale with an increasing number of parameters: as shown in [17,26], CMA-ES and xNES lead to memory allocation errors when tested on high-dimensional problems such as the MuJoCo humanoid task. In fact, Pagliuca and colleagues reported that calculating/updating the covariance matrix on 166,673 parameters, as in the MuJoCo humanoid problem, is

computationally intractable [17]. The sNES algorithm allows for the circumvention of these limitations by storing separable parameter distributions [25], hence reducing the computational load of the method. This makes sNES suitable for high-dimensional search space [17]. Moreover, sNES showed good performance with respect to test function optimization [19,25], pole balancing [11,15,19] and car simulation games [15].

Although CMA-ES, sNES and xNES have been applied to a wide range of application domains, previous comparative studies demonstrated that they are bested by the OpenAI Evolutonary Strategy (OpenAI-ES) [28] with respect to MuJoCo robot locomotion tasks [17] and swarm robotics problems [17,35]. As already discussed in Section 1, drawing inspiration from OpenAI-ES, I propose three variants for each considered ES:

- Symmetric variant (termed as "<ES>_s", where <ES> can be CMA-ES, sNES or xNES): It incorporates symmetric sampling [30] to better estimate the importance of each parameter and, thus, the performance gradient;
- Weight decay variant (termed as "<ES>_wd"): It applies weight decay [38] to keep the parameter size constrained;
- Combined symmetric and weight decay variant (termed as "<ES>_swd"): It applies both previous variants to improve gradient estimation and reduce parameter size.

It is worth noting that the symmetric version of sNES has already been used in [37] to evolve solutions for a collective halfcheetah locomotion task [62].

The implementation of the three variants are integrated in Algorithms A1–A3 through two If-statements (Algorithms A1–A3, lines 7–10 and 15–16). In particular, symmetric sampling is applied on the extracted samples by considered both positive and negative directions (Algorithms A1–A3, lines 7–10). Given the samples extracted from the Gaussian distribution (line 4), symmetric sampling acts by also analyzing the negative samples (lines 8–9). This allows the algorithm to better assess the parameters' influence on the performance. Instead, weight decay acts directly on the update of the parameters (Algorithms A1–A3, lines 15–16). Specifically, weight decay is applied after the parameter updates (line 14) and is modulated by the centroid's learning rate (line 15). The decay factor 0.005 was chosen because it does not strongly impact the centroid update and does not interfere with the other mechanisms of the considered ESs. Furthermore, the value was inherited from the OpenAI-ES implementation (see [28,36]).

2.5. Experimental Settings

All the experiments were replicated 10 times by using different seeds. Although this value might theoretically limit the validity of this investigation, it actually represents a good trade-off between computational load, time required to perform simulations and performance evaluation. Furthermore, this value has been used in previous research works [12,17,34,63]. Table 5 reports the experimental settings used. I used the Xavier method [64] to initialize the neural network controller parameters (i.e., connection weights and biases). Moreover, for locomotion problems, I use batch normalization [65], as already done in [17,28].

Concerning CMA-ES, sNES and xNES, the algorithmic parameters are illustrated in Table 6.

Table 5. Summary of the experimental settings used for the halfcheetah, hopper, swarm aggregation and swarm foraging problems. These values were taken from [15,17,35,43] and allow us to perform a fair comparison with existing studies. N_{reps} denotes the number of replications; N_{evals} indicates the number of evaluations performed during evolution; N_R represents the number of robots; N_{eps} is the number of evaluation episodes; N_{steps} is the number of steps of an episode; d is the number of evolvable parameters.

	Halfcheetah	**Hopper**	**Swarm Aggregation**	**Swarm Foraging**
N_{reps}	10	10	10	10
N_{evals}	5×10^7	5×10^7	1.5×10^9	1.5×10^9
N_R	1	1	10	10
N_{eps}	3	3	20	20
N_{steps}	1000	1000	1000	1000
d	1656	953	406	434
WeightInit BatchNorm	Yes	Yes	No	No

Table 6. Parameters used to discover solutions with the CMA-ES, sNES and xNES algorithms. I adopted the default values reported in [13,18,19,25]. The symbols are defined as follows: PopSize defines the population size. $N_{samples}$ is the number of samples. η_θ and η_σ are, respectively, the learning rate for the centroid θ and the covariance/variance σ. n_{evals} indicates the number of evaluations performed. The symbol "-" is used when a parameter is not defined for the corresponding ES.

	CMA-ES	**sNES**	**xNES**
PopSize	1	1	1
$N_{samples}$	$4 + \lfloor 3 \times \ln(d) \rfloor$	$4 + \lfloor 3 \times \ln(d) \rfloor$	$4 + \lfloor 3 \times \ln(d) \rfloor$
η_θ	1.0	1.0	1.0
η_σ	value defined as in [13]	$\min(\frac{1}{d}, 0.25)$	$\min(\frac{1}{d}, 0.25)$
StepSize	0.5	$\frac{1}{\frac{4+\lfloor 3\times\ln(d)\rfloor}{2}}$	-
Termination	$n_{evals} = N_{evals}$	$n_{evals} = N_{evals}$	$n_{evals} = N_{evals}$
Seeds	$1 \rightarrow 10$	$1 \rightarrow 10$	$1 \rightarrow 10$

All the experiments were performed with evorobotpy3 [36], a modern simulation tool that provides a broad list of EAs (including CMA-ES, sNES and xNES) and tasks (including swarm aggregation and swarm foraging). Moreover, it incorporates external libraries such as PyBullet, MuJoCo and Gymnasium [66,67], hence allowing users to run experiments of robot locomotion tasks (including halfcheetah and hopper) and Atari games. Evorobotpy3 has been recently used to create custom environments and run simulations on a wide spectrum of problems [27,63,68]. Furthermore, evorobotpy3 contains the implementation of CMA-ES, sNES and xNES algorithms and their symmetric variants, and I used those versions for the experiments presented in this paper. All the simulations were run on an Intel Xeon-Gold 6252N, 128-gigabyte (GB) Random Access Memory (RAM), and an Ubuntu 20.04 Operating System (OS).

3. Results

This section provides a thorough description of the outcomes achieved in the selected problems. Regarding the statistical analysis, I used the Mann–Whitney U test [69] for the Rosenbroack function optimization, and the Kruskal–Wallis H test [70] combined with the Dunn post hoc test [71] for the other problems. Moreover, to analyze relationships between

variables, I employed the Spearman rank correlation coefficient [72]. Statistical significance is reported at a threshold $p < 0.05$.

3.1. Rosenbrock Function Optimization

As a proof-of-concept of my hypothesis, I analyzed CMA-ES, sNES, xNES and their symmetric variants with respect to the Rosenbrock function. As shown in Figure 5, CMA-ES and CMA-ES_s converged to an almost optimal solution within the first 5000 steps, while xNES and xNES_s required approximately 1.5×10^5 steps to discover the optimum. Finally, sNES and sNES_s got stuck in sub-optimal solutions (Figure 5). In statistical terms, both CMA-ES_s and sNES_s significantly outperformed CMA-ES and sNES ($p < 0.05$). It is worth noting that I did not apply weight decay given the particular task objective (i.e., parameter minimization).

3.2. PyBullet Robot Locomotion Problems

The performance analysis of the different algorithms and their variants is summarized in Table 7. With regard to the PyBullet robot locomotion problems, the symmetric variants CMA-ES_s, sNES_s and xNES_s significantly outperform the original CMA-ES, sNES and xNES algorithms ($p < 0.05$ for each pair). Moreover, CMA-ES_swd is remarkably better than CMA-ES ($p < 0.05$). Therefore, the application of symmetric sampling results in enhanced performance. In addition, concerning the hopper problem, the symmetric variants achieve higher performance than the original algorithms (this is particularly evident for the sNES algorithm and its variants), although there is no statistically significant difference ($p > 0.05$). Interestingly, the CMA-ES_wd strongly outperforms CMA-ES ($p < 0.05$). Overall, these outcomes indicate that symmetric sampling is beneficial for performance enhancement in robot locomotion tasks. The fact that the hopper problem is relatively easier than the halfcheetah task explains why symmetric sampling does not produce significantly better results in the former scenario.

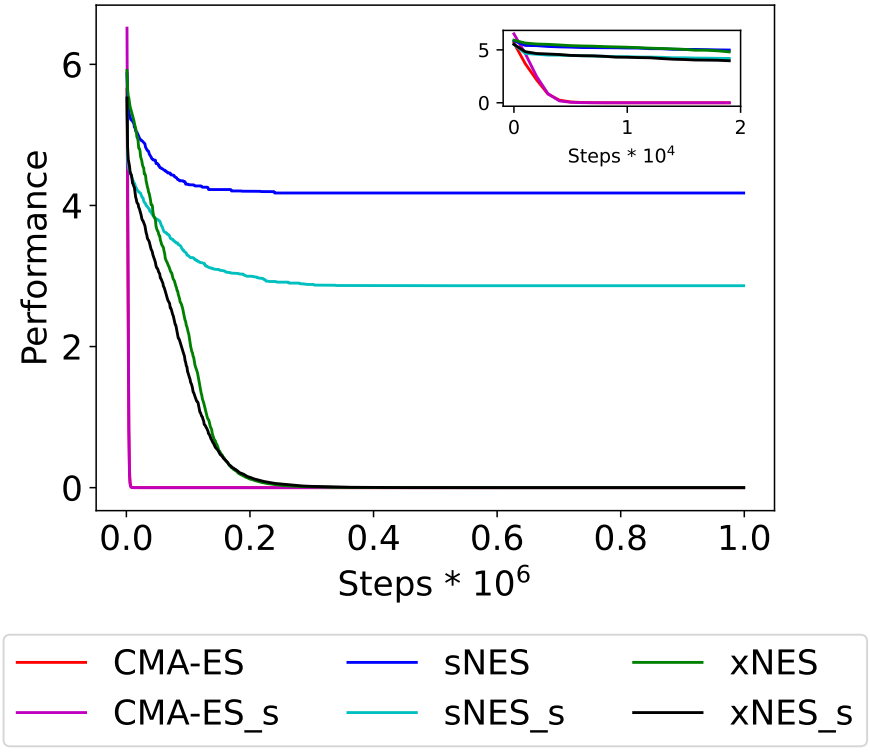


Figure 5. Fitness convergence of CMA-ES, sNES, xNES and their symmetric variants (denoted with the suffix "_s") with respect to the Rosenbrock function. Data are averaged over 10 replications.

Table 7. Average performance achieved by the different ESs in 10 replications of the experiments. Data are illustrated as $\mu \pm \sigma$ (respectively, mean and standard deviation), and bold values identify the best outcomes.

	Halfcheetah	Hopper	Swarm Aggregation	Swarm Foraging
CMA-ES	325.245 ± 141.232	1428.111 ± 286.928	0.896 ± 0.199	153.605 ± 13.197
CMA-ES_s	1191.764 ± 434.595	1532.802 ± 373.663	**1.147 ± 0.221**	**179.310 ± 18.482**
CMA-ES_swd	1318.588 ± 492.879	**1862.538 ± 388.316**	1.038 ± 0.120	160.085 ± 15.476
CMA-ES_wd	687.086 ± 375.933	1450.183 ± 320.090	1.131 ± 0.111	169.460 ± 12.858
sNES	296.042 ± 58.318	826.994 ± 610.100	0.726 ± 0.099	86.490 ± 13.375
sNES_s	1327.817 ± 300.128	1158.427 ± 618.993	0.787 ± 0.123	95.180 ± 4.447
sNES_swd	637.021 ± 321.079	1313.770 ± 280.792	0.948 ± 0.111	130.005 ± 8.845
sNES_wd	262.203 ± 123.086	1394.895 ± 259.452	0.823 ± 0.064	122.815 ± 8.166
xNES	351.127 ± 206.058	1618.882 ± 320.384	0.699 ± 0.063	157.745 ± 13.160
xNES_s	**1561.019 ± 312.735**	1793.127 ± 443.307	0.805 ± 0.162	161.340 ± 17.163
xNES_swd	782.082 ± 161.120	1339.842 ± 275.274	0.927 ± 0.103	151.475 ± 7.011
xNES_wd	470.423 ± 309.552	1452.774 ± 275.731	0.830 ± 0.108	156.020 ± 6.542

The performance distributions achieved in the PyBullet robot locomotion tasks are illustrated in Figure 6a,b. A noteworthy feature is the limited variance in the performance of the CMA-ES, sNES and xNES algorithms (red, green and blue violins) with respect to the halfcheetah problem (see Figure 6a). This implies that these ESs converge towards similar solutions in the different experimental replications. Conversely, the variants display performance differences, with higher median values (white horizontal lines within the violins). Concerning the hopper task, all the algorithms show more pronounced performance variances (see Figure 6b). This is especially evident for sNES and sNES_s (Figure 6b, green and neon green violins).

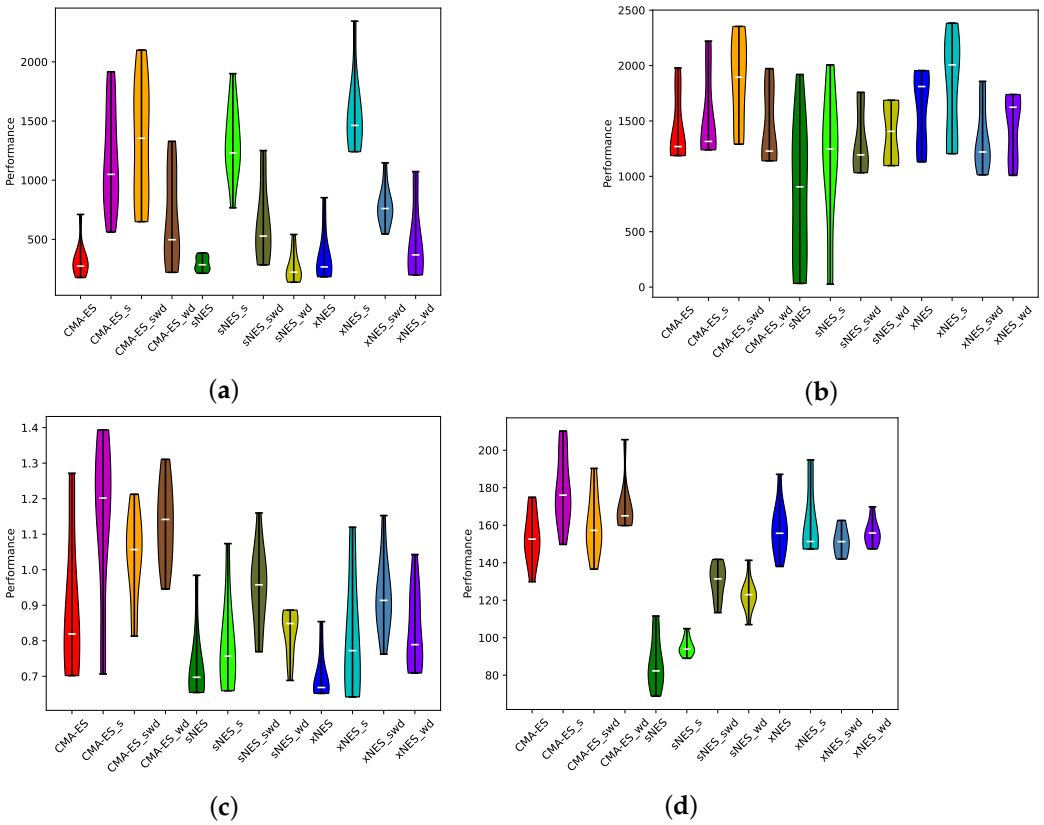


Figure 6. Performance distribution of the original methods (i.e., CMA-ES, sNES and xNES) and their variants in the considered problems: (**a**) Halfcheetah; (**b**) hopper; (**c**) swarm aggregation; (**d**) swarm foraging.

3.3. Collective Problems

By looking at the results achieved in the swarm aggregation problem, all the proposed variants outperform the original CMA-ES, sNES and xNES algorithms. More specifically, CMA-ES_s is remarkably better than CMA-ES ($p < 0.05$). In addition, sNES_swd and xNES_swd are superior to sNES and xNES ($p < 0.05$), respectively. As can be observed in Table 7, both symmetric sampling and weight decay lead to higher fitness values on average. However, the combination of the two techniques generates the best outcomes for sNES and xNES. On the other hand, I observe a significant performance enhancement for CMA-ES with the application of symmetric sampling only.

Concerning the swarm foraging problem, CMA-ES_s is significantly better than CMA-ES ($p < 0.05$). Furthermore, sNES_swd and sNES_wd strongly outperform sNES ($p < 0.05$). Instead, there are no significant differences between xNES and its variants ($p > 0.05$). Therefore, similarly to the former case, I observe that symmetric sampling is beneficial for all the algorithms, particularly for CMA-ES. Moreover, the combination of symmetric sampling and weight decay produces better performance, especially for sNES.

As pointed out in [17,35], a noteworthy feature affecting performance of the original CMA-ES, sNES and xNES algorithms in the considered domains is the parameter size $|w|$. As observed in Table 8, all the variants evolve solutions characterized by remarkably lower values of $|w|$ ($p < 0.05$ for each pair of comparison). Interestingly, CMA-ES_s produces solutions with lower $|w|$ than CMA-ES_wd (see Table 8). This depends on the implementation of the weight decay technique in CMA-ES (see Algorithm A1).

Table 8. Average parameter size of the individuals evolved by the different ESs in 10 replications of the experiments. Data are illustrated as $\mu \pm \sigma$ (respectively, mean and standard deviation).

	Halfcheetah	**Hopper**	**Swarm Aggregation**	**Swarm Foraging**
CMA-ES	8.861 ± 2.124	4.942 ± 2.061	3.359 ± 2.517	5.003 ± 1.173
CMA-ES_s	1.132 ± 0.043	1.031 ± 0.045	0.687 ± 0.161	0.761 ± 0.045
CMA-ES_swd	0.176 ± 0.049	0.216 ± 0.103	0.295 ± 0.134	0.301 ± 0.061
CMA-ES_wd	1.526 ± 0.605	1.096 ± 0.107	0.785 ± 0.268	0.548 ± 0.140
sNES	21.672 ± 4.524	32.675 ± 24.908	4.684 ± 5.916	18.791 ± 7.384
sNES_s	5.843 ± 0.712	3.956 ± 2.190	4.688 ± 3.383	5.785 ± 1.144
sNES_swd	1.602 ± 0.033	1.594 ± 0.094	1.740 ± 0.063	1.404 ± 0.081
sNES_wd	2.478 ± 0.048	2.648 ± 0.049	2.739 ± 0.298	2.851 ± 0.092
xNES	26.672 ± 9.848	12.088 ± 2.747	5.717 ± 4.919	29.654 ± 6.738
xNES_s	7.734 ± 0.623	6.584 ± 1.302	5.030 ± 5.184	11.029 ± 2.250
xNES_swd	2.042 ± 0.083	1.938 ± 0.101	2.112 ± 0.494	2.520 ± 0.236
xNES_wd	3.690 ± 0.533	2.958 ± 0.272	4.136 ± 0.581	4.869 ± 0.893

I also analyzed the runtime execution of the different ESs, which is summarized in Table 9. As can be seen, the usage of symmetric sampling reduces the time required to perform simulations of CMA-ES and xNES. Concerning sNES, I observe similar runtime values for the original algorithm and its variants. Clearly, because sNES does not store any covariance matrix, it is remarkably faster than CMA-ES and xNES (see Table 9).

The analysis of correlation between the parameter size $|w|$ and the performance F is shown in Table 10. Specifically, there exists a significantly negative correlation between the two variables in the halfcheetah, swarm aggregation and swarm foraging problems (Table 10, last row). This means that reducing $|w|$ produces higher values of F. This is in line with the outcomes reported in [35]. However, there are significant differences between the considered ESs. In fact, the performance of CMA-ES and its variants increases when the parameter size $|w|$ decreases regardless of the task. This is particularly evident for the

halfcheetah and hopper problems (see Table 10). Regarding sNES, there is a strong negative correlation between $|w|$ and F in the swarm foraging problem (see Table 10). Instead, the performance of xNES positively correlates with parameter size in the hopper problem (see Table 10).

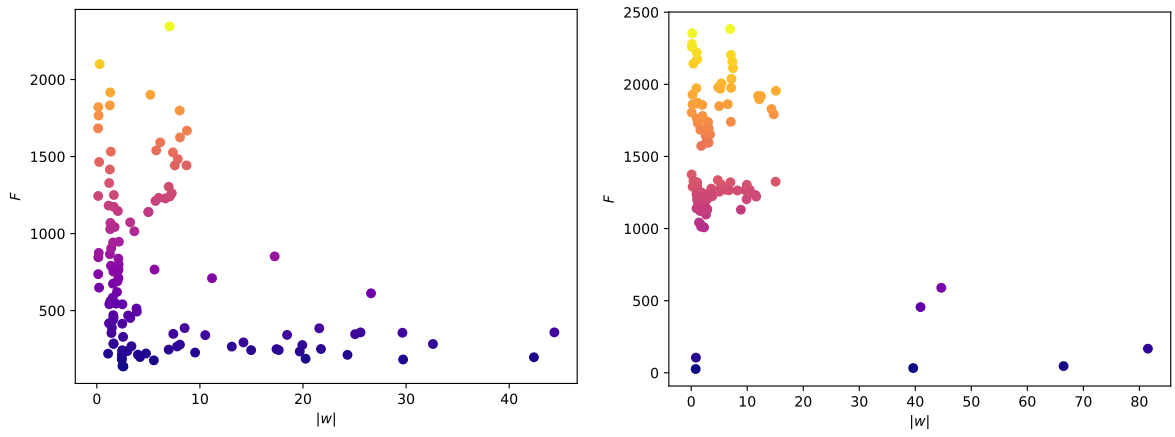
Table 9. Runtime execution (in seconds) of the different ESs in 10 replications of the experiments. Data are shown as $\mu + \sigma$ (respectively, mean and standard deviation).

	Halfcheetah	Hopper	Swarm Aggregation	Swarm Foraging
CMA-ES	$2,403,262.600 \pm 346,527.141$	$587,685.000 \pm 64,178.968$	$227,857.700 \pm 16,119.858$	$602,310.400 \pm 30,751.172$
CMA-ES_s	$895,595.200 \pm 138,957.561$	$431,028.900 \pm 170,173.171$	$221,279.400 \pm 23,526.027$	$495,266.300 \pm 55,198.447$
CMA-ES_swd	$1,165,787.800 \pm 447,464.025$	$486,938.000 \pm 178,398.972$	$191,814.900 \pm 12,953.183$	$443,185.100 \pm 12,211.343$
CMA-ES_wd	$2,605,455.600 \pm 847,680.235$	$585,831.000 \pm 36,673.750$	$245,264.000 \pm 6957.128$	$661,917.000 \pm 15,266.188$
sNES	$403,481.300 \pm 5340.709$	$117,324.200 \pm 4455.388$	$75,741.600 \pm 1742.606$	$693,070.300 \pm 5420.562$
sNES_s	$392,439.600 \pm 4915.888$	$116,163.700 \pm 2784.170$	$80,960.600 \pm 2036.227$	$686,855.700 \pm 10,435.205$
sNES_swd	$399,412.100 \pm 4441.495$	$114,529.900 \pm 1072.164$	$89,258.500 \pm 1991.245$	$685,831.600 \pm 8647.254$
sNES_wd	$402,771.300 \pm 3047.876$	$114,956.200 \pm 668.527$	$85,327.100 \pm 2094.511$	$695,024.800 \pm 7660.745$
xNES	$3,085,422.600 \pm 704,472.062$	$761,742.000 \pm 8755.846$	$447,330.400 \pm 2642.274$	$707,339.500 \pm 7728.604$
xNES_s	$1,014,073.000 \pm 47,048.866$	$640,952.100 \pm 12,914.478$	$337,717.800 \pm 5760.611$	$662,463.200 \pm 14,796.472$
xNES_swd	$1,578,066.100 \pm 292,822.148$	$646,775.100 \pm 7294.765$	$348,927.700 \pm 9513.006$	$677,261.000 \pm 6785.183$
xNES_wd	$3,377,565.200 \pm 616,647.901$	$777,373.3 \pm 17,405.381$	$452,076.000 \pm 1642.544$	$709,013.400 \pm 3205.601$

Table 10. Spearman correlation coefficient ρ between parameter size $|w|$ and performance F for each evolutionary problem. Statistical significance is highlighted in bold.

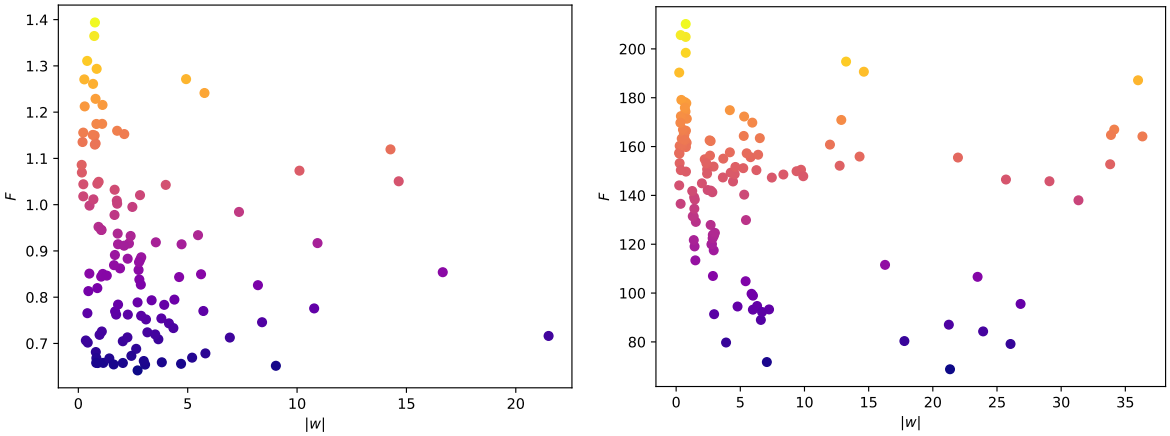
	Halfcheetah	Hopper	Swarm Aggregation	Swarm Foraging
CMA-ES	$-$**0.607**	$-$**0.444**	-0.094	-0.135
sNES	-0.031	0.052	-0.108	$-$**0.825**
xNES	-0.160	**0.525**	0.281	0.303
Overall	$-$**0.347**	-0.034	$-$**0.310**	$-$**0.327**

Figure 7 shows how performance F varies based on $|w|$ with respect to the considered problems, where I take into account all the solutions discovered by CMA-ES, sNES, xNES and their variants. As can be observed, the evolved solutions are generally characterized by a broad range of $|w|$, and the best performance is obtained by solutions with limited $|w|$ (see Figure 7, left side of each subplot). A worthwhile exception is represented by the hopper task, for which all the algorithms typically evolve solutions with moderate values of $|w|$ (Figure 7b, see also Table 10).



(**a**) (**b**)

Figure 7. *Cont.*



(**c**) (**d**)

Figure 7. Correlation between the average absolute value of weights $|w|$ and the performance F: (**a**) Halfcheetah; (**b**) hopper; (**c**) swarm aggregation; (**d**) swarm foraging.

4. Discussion

Overall, the results illustrated in Section 3 produce useful considerations. The analysis involving the Rosenbrock function, which represents the proof-of-concept of this study, demonstrates that symmetric sampling is beneficial for sNES and xNES in terms of convergence speed. Moreover, it significantly enhances the performance of both CMA-ES and sNES. These outcomes prove that symmetric sampling positively influences the considered ESs even in this relatively simple scenario. Future studies might clarify whether these outcomes hold as the input size d scales.

With respect to the remaining problems, the outcomes emphasize that symmetric sampling is advantageous (in terms of performance enhancements) for CMA-ES, sNES and xNES. In detail, CMA-ES_s is significantly better than CMA-ES in the halfcheetah, swarm aggregation and swarm foraging problems. Furthermore, with regard to halfcheetah, sNES_s and xNES_s are remarkably superior to sNES and xNES, respectively. The addition of weight decay is beneficial to CMA-ES with regard to all the problems and produces a performance enhancement for sNES in three problems (hopper, swarm aggregation and swarm foraging), whereas the benefits are limited to two tasks (halfcheetah and swarm aggregation) for xNES. Finally, the combination of symmetric sampling and weight decay is advantageous for CMA-ES and sNES with respect to all the problems: CMA-ES_swd significantly outperforms CMA-ES in the halfcheetah task, whereas sNES_swd is better than sNES in the two collective problems. In addition, xNES_swd exhibits a performance enhancement over xNES in two problems (halfcheetah and swarm aggregation), with a significant difference between the methods in the swarm aggregation task.

To provide a more comprehensive analysis, I report the results of an additional series of experiments on the HC algorithm, by using the parameters provided in Table 11. As can be seen in Table 12, HC poorly performs on the halfcheetah and hopper tasks compared to the other considered ESs. Interestingly, HC is better than other methods in the swarm aggregation task, and is also the second-best performing algorithm in the swarm foraging problem. Lastly, it achieves better performance than sNES and sNES_s in the Rosenbrock function. Therefore, these outcomes indicate that the complexification of ESs through techniques storing the parameter importance, like covariance matrix or variance vector, works depending on the considered problem and is not a general feature. These findings align with the results presented in recent literature studies [11,12,27,44].

Table 11. Parameters used to run experiments with the HC algorithm. I adopted the same settings as in [12].

Parameter	Value
N_{reps}	10
$PopSize$	50
$MutRate$	0.01

Table 12. Performance of the HC algorithm on the considered tasks in 10 replications of the experiments. Data are shown as $\mu + \sigma$ (respectively, mean and standard deviation).

Rosenbrock	Halfcheetah	Hopper	Swarm Aggregation	Swarm Foraging
0.240 ± 0.021	67.337 ± 22.855	301.997 ± 138.418	1.399 ± 0.130	176.422 ± 64.989

Another relevant outcome of the presented analysis lies in the effect of symmetric sampling on parameter size $|w|$, whose impact on the final performance is remarkable in the presented domains (see Table 10). Indeed, as shown in Table 8, considering both positive and negative samples result in a remarkable drop of $|w|$. A similar trend has been observed for OpenAI-ES in [11]. Surprisingly, CMA-ES_s discovers solutions that have a smaller $|w|$ value than CMA-ES_wd in three problems (halfcheetah, hopper and swarm aggregation, see Table 8).

Given that the application of weight decay facilitates the reduction of $|w|$ by definition, the combination of symmetric sampling and weight decay leads to a notable drop in parameter size (see Table 8). In particular, the decrease of $|w|$ ranges from 11.386 (swarm aggregation) to 50.347 (halfcheetah) for CMA-ES, from 2.692 (swarm aggregation) to 20.499 (hopper) for sNES, and from 2.707 (swarm aggregation) to 13.062 for xNES (halfcheetah). Therefore, the swarm aggregation problem is less affected by the reduction of $|w|$. This is confirmed by the fact that the solutions discovered by CMA-ES, sNES and xNES for this task are characterized by smaller $|w|$ than for the other problems (see Table 8). In particular, concerning sNES, there is a significant difference in the parameter size between the solutions evolved for the swarm aggregation task and those discovered for the other problems ($p < 0.05$). A similar trend is observed for xNES with regard to halfcheetah and swarm foraging ($p < 0.05$), and for CMA-ES with respect to halfcheetah ($p < 0.05$). Furthermore, the combined effect of symmetric sampling and weight decay is more pronounced for the halfcheetah and hopper problems. In fact, in these tasks, keeping $|w|$ small is paramount to finding effective solutions.

In general, the parameter size of the solutions discovered by CMA-ES is smaller than sNES and xNES, especially with regard to halfcheetah, hopper and swarm foraging tasks ($p < 0.05$, see Table 8). The same outcome is observed by looking at the CMA-ES, sNES and xNES variants, irrespective of the specific problem ($p < 0.05$). A worthwhile difference between CMA-ES and the other ESs lies in the usage of the best half of samples to update the solution (Algorithm A1, line 12), which narrows the search for influential parameter relationships. Instead, sNES and xNES attribute importance to all samples, albeit proportional to their efficacy. Another dissimilarity regards the covariance matrix update: CMA-ES exploits cumulative paths (Algorithm A1, line 13) and keeps the matrix update linear (Algorithm A1, line 14). Conversely, xNES updates the covariance matrix through a matrix exponential operator (Algorithm A3, line 14), which produces bigger $|w|$ values in the long term.

It is worth underscoring that these observations hold for the PyBullet robot locomotion and collective problems considered here. In fact, there is a tight relationship between the efficacy of symmetric sampling and weight decay and the properties of the problem.

For example, when considering a robot controlled by a sensory–motor neural network, keeping the parameter size limited is likely to produce poor performance. Similarly, the analysis in [12] demonstrates that there is no guarantee that symmetric sampling and weight decay are advantageous in collective problems involving heterogeneous agents. Future studies should clarify the domains in which symmetric sampling and weight decay might play a role, hence providing guidelines for researchers.

5. Conclusions

Evolutionary Strategies (ESs) are optimization metaheuristics that are inspired by evolutionary principles and apply evolutionary operators (i.e., mutation, recombination and selection). Since their introduction in the early 70s, several modifications of the original formulation have been proposed, and the most advanced versions apply rather sophisticated techniques. For example, ESs like the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) and Exponential Natural Evolution Strategies (xNESs) calculate a covariance matrix storing the mutual relationships between the parameters to be optimized, which is computationally demanding and scales linearly with the number of parameters. On the other hand, the Separable Natural Evolution Strategy algorithm (sNES) reduces this burden by storing separate parameter variances, which is suitable for large search spaces. Moreover, the OpenAI Evolutionary Strategy (OpenAI-ES) keeps historical information of how performance varies by means of two momentum vectors. Furthermore, it applies symmetric sampling and weight decay, which allow the algorithm to both have a better understanding of how parameter variations affect performance and keep parameter size limited. The latter property is pivotal in domains such as PyBullet and MuJoCo robot locomotion problems. Because CMA-ES, sNES and xNES achieve sub-optimal performance in these domains, applying mechanisms like symmetric sampling and weight decay to these ESs might positively impact their efficacy.

This research investigates the effects of transferring symmetric sampling and weight decay to CMA-ES, sNES and xNES. Specifically, I propose three variants for CMA-ES, sNES and xNES, termed as "<ES>_s", "<ES>_wd" and "<ES>_swd" (with <ES> corresponding to CMA-ES, sNES or xNES), which apply, respectively, symmetric sampling, weight decay, or both. To showcase the effects of these variants, I perform a comparison on four problems: (i) halfcheetah robot locomotion; (ii) hopper robot locomotion; (iii) swarm aggregation and (iv) swarm foraging. Furthermore, as a proof-of-concept, I provide evidence of the effects of symmetric sampling on the Rosenbrock function optimization. Experiments in these domains demonstrate that the proposed variants are in general more effective than the original algorithms. In particular, symmetric sampling and weight decay improve the performance of CMA-ES irrespective of the considered problem. The positive impact of symmetric sampling on CMA-ES is statistically significant with regard to halfcheetah, swarm aggregation and swarm foraging tasks. Similarly, symmetric sampling produces a performance enhancement in sNES, especially for the halfcheetah task, while the simultaneous application of symmetric sampling and weight decay to sNES is remarkably advantageous in the collective problems. As concerns xNES, symmetric sampling is particularly beneficial for the halfcheetah problem, whereas the combined symmetric sampling and weight decay variant is more effective in the swarm aggregation task. These outcomes are motivated by the fact that all the proposed variants produce a drop in the parameter size $|w|$, which is crucial to finding effective solutions in these domains [17,35]. However, I observed different trends depending on the specific ES and task: CMA-ES keeps $|w|$ smaller than sNES and xNES, thanks to its operational mechanisms. In addition, the solutions for the swarm aggregation task are typically characterized by smaller $|w|$ than those found for the other problems. Lastly, the application of symmetric sampling to CMA-ES and

sNES is significantly advantageous for optimizing the Rosenbrock function. Overall, the achieved results demonstrate that, in the considered domains, symmetric sampling and weight decay are useful techniques to enhance the performance of ESs.

As discussed in Section 4, the efficacy of symmetric sampling and weight decay is related to the task properties. Therefore, a straightforward pathway for future research is to extend this analysis to a broad spectrum of domains and identify the scenarios in which symmetric sampling and weight decay produce performance enhancements. In particular, analyzing the effectiveness of these techniques on benchmark tasks, like the Traveling Salesman Problem (TSP) or Vehicle Routing Problem (VRP), and on other test functions will be the object of future studies. This in turn will enable researchers to consciously choose the most suitable ESs. Furthermore, future works could investigate alternative techniques, further enhancing the performance of CMA-ES, sNES and xNES. In this respect, the incorporation of local refinement techniques [27,50] could be object of future studies.

Funding: This research received no external funding.

Data Availability Statement: Code to replicate the experiments presented in this work can be found at https://github.com/PaoloP84/evorobotpy3 (accessed on 19 May 2026).

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CMA-ES	Covariance Matrix Adaptation Evolutionary Strategy
EA	Evolutionary Algorithm
EC	Evolutionary Computation
ES	Evolutionary Strategy
FIM	Fisher Information Matrix
FOV	Field Of View
GB	Gigabytes
IR	InfraRed
LED	Light-Emitting Diode
MLP	Multi-Layer Perceptron
OpenAI-ES	OpenAI Evvolutionary Strategy
OS	Operating System
RAM	Random Access Memory
RL	Reinforcement Learning
RNN	Recurrent Neural Network
sNESs	Separable Natural Evolution Strategies
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
xNESs	Exponential Natural Evolution Strategies

Appendix A

Algorithm A1 *CMA-ES*

1: Initialize:

$\lambda \leftarrow 4 + \lfloor 3 \times \ln(n) \rfloor$

$\mu \leftarrow \lfloor \frac{\lambda}{2} \rfloor$

$w_{i=1...\mu} \leftarrow \ln(\frac{\lambda+1}{2}) - \ln(i)$

$\mu_w \leftarrow \frac{1}{\sum_{k=1}^{\mu} w_k{}^2}$

$c_c \leftarrow \frac{4}{n+4}$

$c_{cov} \leftarrow \frac{2}{(n+\sqrt{2})^2}$
$c_\sigma \leftarrow \frac{4}{n+4}$
$d_\sigma \leftarrow \frac{1}{c_\sigma} + 1$
$c_1 \approx \frac{2}{n^2+\mu_w}$
$c_\mu \approx \frac{\mu_w}{n^2+\mu_w}$
$\mathbb{C} \leftarrow \mathbb{I}$
$\sigma \leftarrow 1$
$p_\sigma \leftarrow 0$
$p_c \leftarrow 0$
$\theta \sim \mathcal{N}(0, \mathbb{I})$
$\gamma, f, symm, wd$
for $t \leftarrow 0$ to γ **do**
  for $i \leftarrow 0$ to λ **do**
    draw sample: $s_i \sim \mathcal{N}(0, \mathbb{C})$
    ${z_i}^+ \leftarrow \theta + \sigma \times s_i$
    evaluate the fitness: $f({z_i}^+)$
    if $symm$ **then**
      ${z_i}^- \leftarrow \theta - \sigma \times s_i$
      evaluate the fitness: $f({z_i}^-)$
    end if
  end for
  sort (s_i, z_i) w.r.t. $f(z_i)$ and select best μ samples: $\mathbb{P} \leftarrow selectBest(\mu, (s_i, f(z_i)))$
  update cumulative paths:
    $p_\sigma \leftarrow (1 - c_\sigma) \times p_\sigma + \sqrt{c_\sigma \times (2 - c_\sigma)} \times \sqrt{\mu_w} \times \sum_{s_i \in \mathbb{P}} w_i \times s_i$ ▷ Cumulative path for σ
    $h_\sigma = (\frac{\|p_\sigma\|^2}{n} < 2 \times \frac{4}{n+1})$
    $p_c \leftarrow (1 - c_c) \times p_c + h_\sigma \times \sqrt{c_\sigma \times (2 - c_\sigma)} \times \sqrt{\mu_w} \times \sum_{s_i \in \mathbb{P}} w_i \times \sqrt{\mathbb{C}} \times s_i$ ▷ Cumulative path for $\mathbb{C}$
  update parameters:
    $\sigma \leftarrow \sigma \times \exp^{\frac{c_\sigma}{d_\sigma}} (\frac{\|p_\sigma\|}{\mathbb{E}\|\mathcal{N}(0,\mathbb{I})\|}) - 1)$
    $\mathbb{C} \leftarrow (1 - c_1 - c_\mu) \times \mathbb{C} + c_1 \times (p_c \times {p_c}^\intercal + (1 - h_\sigma) \times c_c \times (2 - c_c) \times \mathbb{C}) + c_\mu \times \sum_{s_i \in \mathbb{P}} w_i \times \sqrt{\mathbb{C}} \times s_i \times (\sqrt{\mathbb{C}} \times {s_i}^\intercal)$
    $\theta \leftarrow \theta + \sigma \times \sqrt{\mathbb{C}} \times \sum_{s_i \in \mathbb{P}} w_i \times s_i$
  if wd **then**
    $\theta \leftarrow \theta - \sigma \times 0.005 \times \theta$
  end if
end for

Algorithm A2 *sNES*

Initialize:
  $\lambda \leftarrow 4 + \lfloor 3 \times \ln(n) \rfloor$
  $\sigma \leftarrow 1.0$
  $\theta \sim \mathcal{N}(0, \mathbb{I})$
  $\gamma, f, symm, wd$
for $t \leftarrow 0$ to γ **do**
  for $i \leftarrow 0$ to λ **do**
    draw sample: $s_i \sim \mathcal{N}(0, \mathbb{I})$
    ${z_i}^+ \leftarrow \theta + \sigma \times s_i$
    evaluate the fitness: $f({z_i}^+)$

```
        if symm then
            z_i^- ← θ − σ × s_i
            evaluate the fitness: f(z_i^-)
        end if
    end for
    sort (s_i, z_i) w.r.t. f(z_i) and compute utilities u_i
    compute gradients:
        ∇_θ J ← Σ_{i=1}^{λ} u_i × s_i
        ∇_σ J ← Σ_{i=1}^{λ} u_i × (s_i^2 − 1)
    update parameters:
        θ ← θ + η_θ × σ × ∇_θ J
        σ ← σ × exp((η_σ / 2) × ∇_σ J)
    if wd then
        θ ← θ − η_θ × 0.005 × θ
    end if
end for
```

Algorithm A3 *xNES*

```
Initialize:
    λ ← 4 + ⌊3 × ln(n)⌋
    Σ_init = A^T × A
    σ ← ∛|det(A)|
    B ← A / σ
    θ ∼ N(0, I)
    γ, f, symm, wd
for t ← 0 to γ do
    for i ← 0 to λ do
        draw sample: s_i ∼ N(0, I)
        z_i^+ ← θ + σ × B^T × s_i
        evaluate the fitness: f(z_i^+)
        if symm then
            z_i^- ← θ − σ × B^T × s_i
            evaluate the fitness: f(z_i^-)
        end if
    end for
    sort (s_i, z_i) w.r.t. f(z_i) and compute utilities u_i
    compute gradients:
        ∇_δ J ← Σ_{i=1}^{λ} u_i × s_i
        ∇_M J ← Σ_{i=1}^{λ} u_i × (s_i × s_i^T − I)
        ∇_σ J ← tr(∇_M J) / d
        ∇_B J ← ∇_M J − ∇_σ J × I
    update parameters:
        θ ← θ + η_δ × (σ × B × ∇_δ J − 0.005 × θ)
        σ ← σ × exp((η_σ / 2) × ∇_σ J)
        B ← B × exp((η_B / 2) × ∇_B J)
    if wd then
        θ ← θ − η_δ × 0.005 × θ
    end if
end for
```

https://doi.org/10.3390/a19070504

References

1. Beyer, H.G.; Arnold, D.V. Theory of evolution strategies—A tutorial. In *Theoretical Aspects of Evolutionary Computing*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 109–133.
2. Hansen, N.; Arnold, D.V.; Auger, A. Evolution strategies. In *Springer Handbook of Computational Intelligence*; Springer: Berlin/ Heidelberg, Germany, 2015; pp. 871–898.
3. Schwefel, H.P. *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie: Mit Einer Vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie*; Springer: Berlin/ Heidelberg, Germany, 1977; Volume 1.
4. Schwefel, H.P. *Evolution and Optimum Seeking: The Sixth Generation*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1993.
5. Back, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*; Oxford University Press: Oxford, UK, 1996.
6. Holland, J.H. *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI, USA, 1975.
7. Slowik, A.; Kwasnicka, H. Evolutionary algorithms and their applications to engineering problems. *Neural Comput. Appl.* **2020**, *32*, 12363–12379.
8. Rödl, V.; Tovey, C.A. Multiple optima in local search. *J. Algorithms* **1987**, *8*, 250–259.
9. Pagliuca, P. Learning and evolution: Factors influencing an effective combination. *AI* **2024**, *5*, 2393–2432.
10. Miller, J.F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, USA, 13–17 July 1999; Volume 2, pp. 1135–1142.
11. Pagliuca, P. Discovering a Single Neural Network Controller for Multiple Tasks with Evolutionary Algorithms. *J. Artif. Intell. Auton. Intell.* **2025**, *2*, 322–348.
12. Pagliuca, P.; Vitanza, A. What Evolutionary Algorithm(s) Should We Use? Towards Preliminary Guidelines from an Exploratory Analysis of Foraging Problems. In Proceedings of the 22st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), Reykjavik, Iceland, 22–24 June 2026.
13. Hansen, N.; Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **2001**, *9*, 159–195.
14. Igel, C. Neuroevolution for reinforcement learning using evolution strategies. In *Proceedings of the 2003 Congress on Evolutionary Computation, (CEC 2003)*; IEEE: New York, NY, USA, 2003; Volume 4, pp. 2588–2595.
15. Pagliuca, P.; Nolfi, S. Robust optimization through neuroevolution. *PLoS ONE* **2019**, *14*, e0213193.
16. Ajani, O.S.; Kumar, A.; Mallipeddi, R. Covariance matrix adaptation evolution strategy based on correlated evolution paths with application to reinforcement learning. *Expert Syst. Appl.* **2024**, *246*, 123289.
17. Pagliuca, P.; Milano, N.; Nolfi, S. Efficacy of modern neuro-evolutionary strategies for continuous control optimization. *Front. Robot. AI* **2020**, *7*, 98.
18. Glasmachers, T.; Schaul, T.; Yi, S.; Wierstra, D.; Schmidhuber, J. Exponential natural evolution strategies. In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, Portland, OR, USA, 7–11 July 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 393–400.
19. Wierstra, D.; Schaul, T.; Glasmachers, T.; Sun, Y.; Peters, J.; Schmidhuber, J. Natural evolution strategies. *J. Mach. Learn. Res.* **2014**, *15*, 949–980.
20. Akimoto, Y.; Nagata, Y.; Ono, I.; Kobayashi, S. Bidirectional relation between CMA evolution strategies and natural evolution strategies. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 154–163.
21. Amari, S.I. Natural gradient works efficiently in learning. *Neural Comput.* **1998**, *10*, 251–276.
22. Pagliuca, P.; Milano, N.; Nolfi, S. Maximizing adaptive power in neuroevolution. *PLoS ONE* **2018**, *13*, e0198788.
23. Pagliuca, P.; Nolfi, S. Integrating learning by experience and demonstration in autonomous robots. *Adapt. Behav.* **2015**, *23*, 300–314.
24. Basilio, S.A.; Goliatt, L. Gradient boosting hybridized with exponential natural evolution strategies for estimating the strength of geopolymer self-compacting concrete. *Knowl.-Based Eng. Sci.* **2022**, *3*, 1–16. https://doi.org/10.51526/kbes.2022.3.1.1-16.
25. Schaul, T.; Glasmachers, T.; Schmidhuber, J. High dimensions and heavy tails for natural evolution strategies. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin Ireland, 12–16 July 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 845–852.
26. Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2016; pp. 1329–1338.
27. Pagliuca, P. Whether, How and When Modern Evolutionary Strategies Can Be Improved. *J. Artif. Intell. Auton. Intell.* **2026**, *2*, 425–449.
28. Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv* **2017**, arXiv:1703.03864.
29. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998; Volume 1.

30. Brockhoff, D.; Auger, A.; Hansen, N.; Arnold, D.V.; Hohm, T. Mirrored sampling and sequential selection for evolution strategies. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 11–21.
31. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
32. Todorov, E.; Erez, T.; Tassa, Y. Mujoco: A physics engine for model-based control. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: New York, NY, USA, 2012; pp. 5026–5033.
33. Coumans, E.; Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016. URL: https://pybullet.org (accessed on 6 December 2017).
34. Pagliuca, P.; Nolfi, S. The dynamic of body and brain co-evolution. *Adapt. Behav.* **2022**, *30*, 245–255.
35. Pagliuca, P.; Vitanza, A. A Comparative Study of Evolutionary Strategies for Aggregation Tasks in Robot Swarms: Macro- and Micro-Level Behavioral Analysis. *IEEE Access* **2025**, *13*, 72721–72735.
36. Pagliuca, P.; Nolfi, S.; Vitanza, A. Evorobotpy3: A flexible and easy-to-use simulation tool for Evolutionary Robotics. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2025 Companion), Malaga Spain, 14–18 July 2025; Association for Computing Machinery: New York, NY, USA, 2025.
37. Pagliuca, P.; Vitanza, A. Evolving effective neural networks in different domains with the OpenAI-ES algorithm. In *Proceedings of the 33rd Italian Workshop on Neural Networks (WIRN 2025)*; Springer: Berlin/Heidelberg, Germany, 2026; *in press.*
38. Krogh, A.; Hertz, J. A simple weight decay can improve generalization. *Adv. Neural Inf. Process. Syst.* **1991**, *4*, 950–957.
39. Bäck, T.; Schwefel, H.P. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.* **1993**, *1*, 1–23. https://doi.org/10.1162/evco.1993.1.1.1.
40. Papadrakakis, M.; Lagaros, N.D.; Thierauf, G.; Cai, J. Advanced solution methods in structural optimization based on evolution strategies. *Eng. Comput.* **1998**, *15*, 12–34.
41. Rais Martínez, J.; Aznar Gregori, F. Comparison of evolutionary strategies for reinforcement learning in a swarm aggregation behaviour. In Proceedings of the 2020 3rd International Conference on Machine Learning and Machine Intelligence, Hangzhou, China, 18–20 September 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 40–45.
42. Pagliuca, P.; Vitanza, A. Self-organized aggregation in group of robots with OpenAI-ES. In *Proceedings of the International Conference on Soft Computing and Pattern Recognition*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 770–780.
43. Pagliuca, P.; Vitanza, A. Evolving aggregation behaviors in swarms from an evolutionary algorithms point of view. In *Applications of Artificial Intelligence and Neural Systems to Data Science*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 317–328.
44. El Saliby, M.; Nadizar, G.; Salvato, E.; Medvet, E. Eventually, all you need is a simple evolutionary algorithm (for neuroevolution of continuous control policies). In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Melbourne, VIC, Australia, 14–18 July 2024; Association for Computing Machinery: New York, NY, USA, 2024; pp. 1904–1913.
45. Wawrzynski, P. Learning to control a 6-degree-of-freedom walking robot. In *Proceedings of the EUROCON 2007-The International Conference on "Computer as a Tool"*; IEEE: New York, NY, USA, 2007; pp. 698–705.
46. Murthy, S.S.; Raibert, M.H. 3D balance in legged locomotion: Modeling and simulation for the one-legged case. *ACM SIGGRAPH Comput. Graph.* **1984**, *18*, 27.
47. Rosenbrock, H. An automatic method for finding the greatest or least value of a function. *Comput. J.* **1960**, *3*, 175–184.
48. Emiola, I.; Adem, R. Comparison of minimization methods for Rosenbrock functions. In *Proceedings of the 2021 29th Mediterranean Conference on Control and Automation (MED)*; IEEE: New York, NY, USA, 2021; pp. 837–842.
49. Ngartera, L.; Diallo, C. A comparative study of optimization techniques on the Rosenbrock function. *Open J. Optim.* **2024**, *13*, 51–63.
50. Pagliuca, P. Analysis of the Exploration-Exploitation Dilemma in Neutral Problems with Evolutionary Algorithms. *J. Artif. Intell. Auton. Intell.* **2024**, *1*, 110–121.
51. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386.
52. Brambilla, M.; Ferrante, E.; Birattari, M.; Dorigo, M. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intell.* **2013**, *7*, 1–41.
53. Hamann, H. *Swarm Robotics: A Formal Approach*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 221.
54. Şahin, E. Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the International Workshop on Swarm Robotics*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 10–20.
55. Bonani, M.; Longchamp, V.; Magnenat, S.; Rétornaz, P.; Burnier, D.; Roulet, G.; Vaussard, F.; Bleuler, H.; Mondada, F. The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: New York, NY, USA, 2010; pp. 4187–4193.
56. Elman, J.L. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211.
57. Elman, J.L. Distributed representations, simple recurrent networks, and grammatical structure. *Mach. Learn.* **1991**, *7*, 195–225.

58. Pagliuca, P. Emerging Trends in Multi-Agent Systems from an Evolutionary Perspective. In Proceedings of the 27th Edition of the Workshop from Object to Agents (WOA26), Salerno, Italy, 15–17 June 2026.
59. Kullback, S.; Leibler, R.A. On information and sufficiency. *Ann. Math. Stat.* **1951**, *22*, 79–86.
60. Martens, J. New insights and perspectives on the natural gradient method. *J. Mach. Learn. Res.* **2020**, *21*, 1–76.
61. Ahn, J.; Im, E.; Lee, Y. Learning impulse-reduced gait for quadruped robot using CMA-ES. In *Proceedings of the 2023 20th International Conference on Ubiquitous Robots (UR)*; IEEE: New York, NY, USA, 2023; pp. 261–266.
62. Iyer, S.; Aydeniz, A.A.; Dixit, G.; Tumer, K. Multiagent Quality-Diversity for Effective Adaptation. In *ECAI 2025*; IOS Press: Amsterdam, The Netherlands, 2025; pp. 3323–3330.
63. Pagliuca, P.; Vitanza, A. Learning Locomotion by Co-Evolution of Morphological and Neural Parameters. In *Proceedings of the 2025 IEEE International Conference on Development and Learning (ICDL)*; IEEE: New York, NY, USA, 2025; pp. 1–6.
64. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*; JMLR Workshop and Conference Proceedings; PMLR: Cambridge, MA, USA, 2010; pp. 249–256.
65. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv* **2015**, arXiv:1502.03167.
66. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. Openai gym. *arXiv* **2016**, arXiv:1606.01540.
67. Towers, M.; Kwiatkowski, A.; Balis, J.; De Cola, G.; Deleu, T.; Goulão, M.; Andreas, K.; Krimmel, M.; Kg, A.; Perez-Vicente, R.; et al. Gymnasium: A standard interface for reinforcement learning environments. *Adv. Neural Inf. Process. Syst.* **2026**, *38*.
68. Pagliuca, P.; Trivisano, G.; Vitanza, A. How to Evolve Aggregation in Robotic Multi-Agent Systems. In Proceedings of the 26th Edition of the Workshop From Object to Agents (WOA25), Trento, Italy, 2–5 July 2025; pp. 140–156.
69. Mann, H.B.; Whitney, D.R. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Stat.* **1947**, *18*, 50–60.
70. Kruskal, W.H.; Wallis, W.A. Use of ranks in one-criterion variance analysis. *J. Am. Stat. Assoc.* **1952**, *47*, 583–621.
71. Dunn, O.J. Multiple comparisons using rank sums. *Technometrics* **1964**, *6*, 241–252.
72. Spearman, C. The proof and measurement of association between two things. *Am. J. Psychol.* **1904**, *15*, 72–101.