Deep Learning Evolved: Overcoming Sub-Optimal Local Minima with $(\mu/\rho + \lambda)-$Evolution Strategies

Pablo Rivas , *Senior, IEEE*
School of Engineering and Computer Science
Department of Computer Science
Baylor University
Email: Pablo_Rivas@Baylor.edu

***Abstract*—Integrating Evolution Strategies (ES) and Backpropagation (BP) within a deep neural network framework presents a significant challenge, as ES has previously only been shown to perform comparably to BP for smaller problems. In this study, we extend the application of ES to high-dimensional problems, using it to initialize the weights of a Deep Neural Network (DNN). Our experiments demonstrate that this novel ES approach can effectively overcome local minima and converge towards near-optimal global solutions. Following ES initialization, we employ traditional BP gradient methods to further refine the weights based on the initial set provided by ES. A key finding of our research is the potential for ES to reduce the computational time required by traditional gradient-based backpropagation learning methods. This efficiency is achieved by providing an initial set of weights close to the global optimum, enabling the network to converge more rapidly than with random or zero-weight initialization approaches. Our approach offers a promising direction for future research in the efficient training of deep neural networks and opens up new possibilities for tackling high-dimensional problems with these networks.**

***Index Terms*—machine learning, deep learning, evolution strategies, genetic algorithms**

I. Introduction

Deep Backpropagation Neural Networks (DNNs) have emerged as a pivotal tool in the realm of pattern recognition and classification applications. Their inherent ability to approximate any square-integrable function with remarkable precision, coupled with their capability to flawlessly implement any arbitrary finite training set, underscores their potency in these domains [1]. It is noteworthy that a multitude of problems have been efficaciously resolved utilizing deep neural networks [2].

Recently, Larochelle et al. conducted an exhaustive study on various strategies for training deep neural networks [3]. Their findings indicate that deep neural networks underperform compared to their single-layer or two-layer counterparts. They also highlighted the propensity of gradient descent to become ensnared in sub-optimal local minima, thereby hindering the discovery of a globally optimal solution [4], [5].

The pursuit of augmenting the efficacy of the original backpropagation (BP) algorithm has predominantly centered on pinpointing and adopting a more sophisticated energy function. Additionally, determining an optimal variable learning rate and momentum has been a key focus, per the findings in [6]. It would be beneficial to delve into the specific methodologies and techniques employed to bolster the performance of the BP algorithm and to explore the tangible impacts of these enhancements on the algorithm's precision and efficiency in practical applications.

However, these modifications often add to the complexity of the convergence rate of backpropagation, resulting in computationally intensive algorithms. While fast learning algorithms such as Quickprop exist in the literature, an efficient strategy to escape from local minima remains an elusive challenge for DNN architectures [7]. This paper proposes a formal method for designing Evolution Strategies (ES) for weight initialization in deep neural networks. We hypothesize that by leveraging the optimization capabilities of evolutionary algorithms, we can mitigate the risk of becoming trapped in local minima. This can be achieved by initializing weights in proximity to a global solution and subsequently allowing a gradient descent-based method to finalize the training of the neural network.

Despite existing research that employs ES to train neural networks [4]–[6], [8], [9], integrating both schemes within a deep neural framework still needs to be solved. As demonstrated in [7], ES performs comparably to BP only for smaller problems. While other methods attempt to perform vector optimization using ES [10], we recognize that gradient methods are typically more effective for this task. However, we aim to harness the ability of ES to overcome local minima and converge to globally optimal solutions. We aim to devise a robust method to circumvent local minima by combining these approaches.

Our ultimate objective is to address the issue of local minima trapping in deep neural networks. We propose to evolve a set of initial network weights using Evolution Strategies, intending to produce an output closely aligned with an optimal global solution. Subsequently, traditional gradient methods will update the weights based on the initial set provided. This research project will also demonstrate that by using Evolution Strategies, we can reduce the computational time required by traditional gradient-based backpropagation learning methods. This is achieved by providing initial weights close to the global optimum, enabling the network to converge more rapidly than with a conventional random weight initialization approach. We provide code that implements our approach in a publicly available repository.[1]

[1]https://github.com/pablorp80/mu-ro-lambda-es.git

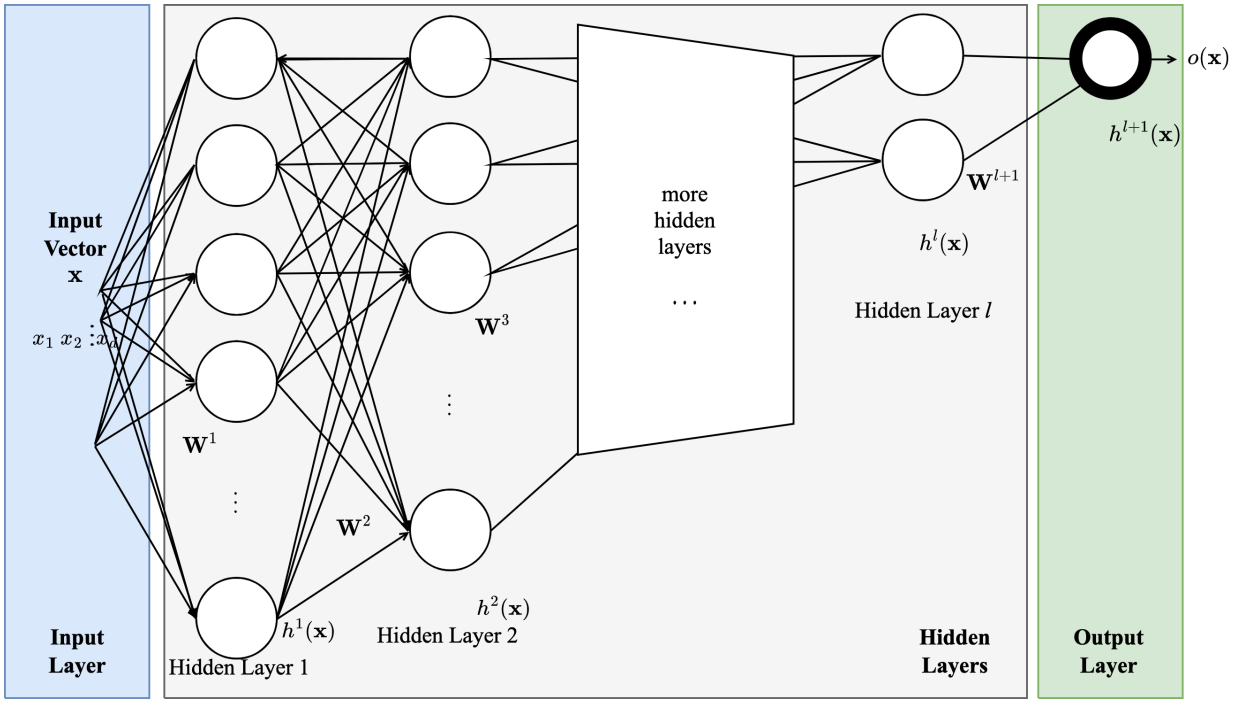


Fig. 1. A deep neural network, also known as a dense network. Network parameters are indicated as $\mathbf{W}$ which directly interact with input $\mathbf{x}$.

The remainder of this paper is organized as follows: Section II briefly introduces DNNs. The basic theory of ES is introduced in Section III. The genetic operators used in this research are explained in Section IV. Section V outlines the experiments conducted and the results obtained. Finally, Section VI presents the conclusions drawn.

II. Deep Neural Networks

A deep neural network (DNN) is architecturally defined by the presence of an input layer, an output layer, and l intermediate layers, referred to as hidden layers. Each of these layers is composed of neuron units. Upon presenting an input sample to the input layer, the neuron units in the subsequent layers compute their respective values. This computation is influenced by the activity of the units to which they are connected in the preceding layers [3]. In the context of this study, our attention is specifically drawn to a particular DNN topology. This topology is characterized by a full, dense connection between each layer and its subsequent layer, extending from the input layer to the output layer [11].

In the context of a neural network, each layer, unit, bias, and weight plays a crucial role in the network's functionality. For a given input $\mathbf{x} \in \mathbf{X}$, the $j-$th unit in the $i-$th layer is denoted as $h_j^i(\mathbf{x})$. Here, $i = 0$ signifies the input layer and $i = l + 1$ represents the output layer. The size of a layer is expressed as $\left|\mathbf{h}^i(\mathbf{x})\right|$.

Each unit in the network has a default activation level, which is determined by its internal bias, denoted as b_j^i. The activation of a unit, specifically unit $h_j^i(\mathbf{x})$, is dictated by the weights W_{jk}^i between the unit $h_k^{i-1}(\mathbf{x})$ in the $(i-1)$-th layer and the unit $h_j^{i-1}(\mathbf{x})$ in the i-th layer. This intricate interplay of layers, units, biases, and weights forms the basis of the neural network's operation, enabling it to perform complex computations and make accurate predictions. The activation of a neural unit $h_j^i(\mathbf{x})$ is given by:

$$h_j^i(\mathbf{x}) = \Phi\left(a_j^i(\mathbf{x})\right), \tag{1}$$

where

$$\begin{aligned} a_j^i(\mathbf{x}) &= \sum_k W_{jk}^i h_k^{i-1}(\mathbf{x}) + b_j^i, \\ \forall i &\in \{1, ..., l\}, \end{aligned}$$

with

$$h^0(\mathbf{x}) = \mathbf{x}.$$

The sigmoid activation function, denoted as Φ =sigm$(\cdot)$, is defined as follows:

$$\text{sigm}\,(a) = \frac{1}{1 + e^{-a}},$$

This function can be substituted with any other activation function per the specific task's requirements. The computation of the output layer, given the final hidden layer, follows a similar process. The output function is expressed as:

$$\begin{aligned} \mathbf{o}(\mathbf{x}) &= \mathbf{h}^{l+1}(\mathbf{x}), \\ \mathbf{h}^{l+1}(\mathbf{x}) &= \mathbf{\Phi}\left(\mathbf{a}^{l+1}(\mathbf{x})\right), \end{aligned} \tag{2}$$

where

$$\mathbf{a}^{l+1}(\mathbf{x}) = \mathbf{W}^{l+1}\mathbf{h}^l(\mathbf{x}) + \mathbf{b}^{l+1},$$

The activation function $\mathbf{\Phi}$ is contingent on the supervised task the network is designed to perform [3]. In the context of a

simple binary classification problem, it is typically the *sign function*, defined as

$$\mathbf{\Phi} = \text{sign}\,(\mathbf{a}) = \begin{cases} +1 & \text{if} \quad \mathbf{a} > \mathbf{0} \\ -1 & \text{if} \quad \mathbf{a} \leq \mathbf{0} \end{cases}$$

as per [12].

Upon presenting an input sample $\mathbf{x}$ to the network, the application of (1) across each layer instigates a pattern of activity throughout the various layers of the neural network [11].

Fig. 1 provides a comprehensive visual depiction of a deep network, including its associated parameters. Ideally, the neuronal activity within the first layer should align with the low-level features of the input, such as edge orientations in the context of natural images. In contrast, the last hidden layers should ideally correspond to higher-level abstractions, such as detecting geometric shapes. This ideal mapping of neuronal activity to features of varying complexity is based on the research conducted in [3].

III. Evolution Strategies Theoretical Background

Let $F(\mathbf{w})$, $F \in \mathcal{F}$ be a fitness function representing the error at the output in a real-valued deep neural network, and

$$F(\mathbf{w}) \rightarrow \text{Minimum}$$

where $\mathbf{w}$ are the control parameters, the $N-$dimensional vectorial representation of the weights and biases in the deep neural network weight $\mathbf{W} \in W$ and bias $\mathbf{b} \in b$ spaces, $\mathbf{W}, \mathbf{b} \subset \mathbf{w} \in \mathbb{R}$, (where $W \cup b \in \mathbb{R}^N$ are finite but not fixed) and the location of the minimum is labeled $\widehat{\mathbf{w}}$ such that

$$\mathbf{w} = (W_1, W_2, ..., W_{N-M}, b_1, b_2, ..., b_N)^T$$

and

$$\widehat{\mathbf{w}} = \left(\widehat{W}_1, \widehat{W}_2, ..., \widehat{W}_{N-M}, \widehat{b}_1, \widehat{b}_2, ..., \widehat{b}_N\right)^T.$$

The type of components w_i of $\mathbf{w}$ and the space spanned by them depends on the neural network's architecture. In the context of Evolution Strategies [13], an individual is defined by three components: an object of type $\mathbf{w}$, an evolvable strategy parameter set denoted by $\mathbf{s}$, and a corresponding fitness value $F(\mathbf{w})$. This can be formally represented as:

$$\mathfrak{a} = (\mathbf{w}, \mathbf{s}, F(\mathbf{w}))$$

The set $\mathbf{s}$, which belongs to the space S, comprises self-adaptation parameters of the ES algorithm that are inherited by the offspring.

The population of these individuals, $\mathfrak{a}$, is composed of μ parent individuals, represented as $\mathfrak{a}m$, where m ranges from 1 to μ, and λ descendant individuals, denoted as $\widehat{\mathfrak{a}}l$, where l ranges from 1 to λ. Here, μ and λ are the strategy parameters not changed by the ES algorithm. The population of parents $\mathbf{B}_\mu^{(g)}$ and descendants $\widehat{\mathbf{B}}_\lambda^{(g)}$ at a time g are denoted as

$$\begin{aligned} \mathbf{B}_\mu^{(g)} &= \left\{\mathfrak{a}_m^{(g)}\right\} = \left(\mathfrak{a}_1^{(g)}, \mathfrak{a}_2^{(g)}, ..., \mathfrak{a}_\mu^{(g)}\right), \\ \widehat{\mathbf{B}}_\lambda^{(g)} &= \left\{\widehat{\mathfrak{a}}_l^{(g)}\right\} = \left(\widehat{\mathfrak{a}}_1^{(g)}, \widehat{\mathfrak{a}}_2^{(g)}, ..., \widehat{\mathfrak{a}}_\lambda^{(g)}\right). \end{aligned}$$

To determine the number of parents taking part in the procreation of a *single* individual, a parameter ρ is introduced as part of the strategy parameters along with μ and λ, where $1 \leq \rho \leq \mu$. The parameter ρ dictates the number of parents involved in the recombination process (μ/ρ) [14], [15].

For a more detailed theoretical background on ES, the reader is encouraged to review refs [13], [14], [16].

A. Sigma/Rho + Lambda Evolution Strategy, $(\mu/\rho + \lambda)-ES$

The $(\mu/\rho+\lambda)-$ES is one kind of Evolution Strategies where only ρ out of μ parents participate in the recombination process to produce λ children. This kind of ES also allows the selection of the best $\mu + \lambda$; in other words, both the children λ and the parents μ are evaluated, and the best are kept. This allows the parents to live until new children evolve to be better than any parents.

B. Initializing a Deep Neural Network with $(\mu/\rho + \lambda)-ES$

We propose the usage of Evolution Strategies $(\mu/\rho+\lambda)-$ES for weight and bias initialization of deep real-valued neural networks. We aim to overcome the problem of local minima trapping in deep neural networks. The idea is to evolve a set of initial network weights and biases $\mathfrak{a}$ performing a global search for solutions in the N-dimensional weight space and provide a set of initial weights and biases $\widehat{\mathbf{w}}$ that produce an output very close to an optimal global solution. Finally, we let the traditional gradient methods update the weights and biases based on the provided initial set $\widehat{\mathbf{w}}$. The general scheme is shown in Fig. 2.

The general procedure for the $(\mu/\rho+\lambda)-$ES is characterized by the following pseudo-code [13]:

Algorithm 1 $(\mu/\rho + \lambda)-ES$

```
1   Begin
2     g = 0;
3     initialize(B_μ^(0) = { (w_m^(0), s_m^(0), F_m^(0)(w)) |_{m=1}^{μ} });
4     Repeat
5       For l = 1 to λ Do Begin
6         C_l = reproduction(B_μ^(g), ρ);
7         s_l = s_recombination(C_l);
8         w_l = w_recombination(C_l);
9         ŝ_l = s_mutation(s_l);
10        ŵ_l = w_mutation(w_l, ŝ_l);
11        F̂_l = F(ŵ_l);
12      End
13      B_λ^(g) = { (w_m^(0), s_m^(0), F_m^(0)(w)) |_{m=1}^{μ} };
14      (μ + λ) : B_μ^(g+1) = selection(B_λ^(g), B_μ^(g), μ);
15      g = g + 1;
16    Until stop_criteria(B_μ^(g))
17  End
```

In the initial generation, denoted as $g = 0$, the parental population $\mathbf{B}\mu^{(0)} = \left(\mathfrak{a}1^{(0)}, \mathfrak{a}2^{(0)}, ..., \mathfrak{a}\mu^{(0)}\right)$ is established as per line #3. Following this initialization, a loop is instigated, encompassing lines #4–16. From the parental population $\mathbf{B}\mu^{(g)}$ at generation g, a novel offspring population $\mathbf{B}\lambda^{(g)}$ is generated by executing lines #6–11, repeated λ times. Each iteration

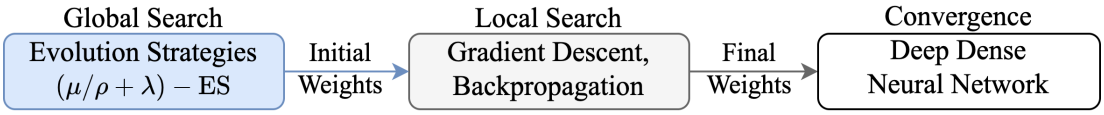


Fig. 2. General description of the proposed scheme.

yields one offspring $\mathfrak{a}$. Initially, during the reproduction phase, a parent family $\mathbb{C}$ of size ρ is selected randomly from the parent pool of size μ via a uniform distribution. This selection process for reproduction is random and does not consider the parental fitness values F, contrasting with conventional selection techniques in genetic algorithms [13], [16]. The recombination of the endogenous strategy parameters (the evolvable $\mathbf{s}$) occurs in line #7, and for the object parameters (the actual candidate solutions $\mathbf{w}$) in line #8. Notably, if $\rho = 1$, the recombinant is merely a copy of the parent, referred to as $(\mu + \lambda)-$ES (not the case here). The mutation of the strategy parameters $\mathbf{s}$ is performed in line #9 and those of the object parameters $\mathbf{w}$ in line #10. The fitness of each offspring is computed in line #11.

Upon the completion of the offspring population $\mathbf{B}\lambda^{(g)}$, selection is executed in line #14, resulting in a new parental population $\mathbf{B}\mu^{(g)}$. Finally, a stop criterion is evaluated in line #16.

Fitness Function: The fitness will be evaluated provided an input vector $\mathbf{x}$ and its associated vector of targets (desired output classification) $\mathbf{T}$. In this project, we will compute the sum of the absolute value of the difference between the desired DNN output $\mathbf{T}$ and the actual output $\mathbf{o}(\mathbf{x})$ [17], derived from (2). The fitness function will be denoted by:

$$F\left(\widehat{\mathbf{w}}\right) = \sum_{k} \left|\mathbf{T}_k - \mathbf{\Phi}_k\left(\mathbf{W}^{l+1}\mathbf{h}^{l}(\mathbf{x}) + \mathbf{b}^{l+1}\right)\right|,$$

where

$$\mathbf{W}, \mathbf{b} \subset \widehat{\mathbf{w}}.$$

For the implementation, see the provided code. The following section will explain the proposed Genetic Operators for the ES.

IV. Genetic Operators

Genetic operators determine how new generations are produced, which we discuss next.

A. Selection

We will use a *determistic* process called *truncation selection* for the selection operator. That is, only the best m best individuals out of γ individuals (denoted as $\mathfrak{a}_{m;\gamma}$) will be chosen as the new population of solutions, this is denoted as

$$\mathbf{B}_{\mu}^{(g+1)} = \left(\mathfrak{a}_{1;\gamma}, \mathfrak{a}_{2;\gamma}, ..., \mathfrak{a}_{\mu;\gamma}\right).$$

The notation $(\mu/\rho + \lambda)$ indicates that both the parents $\mathbf{B}_{\mu}^{(g)}$and their children $\widehat{\mathbf{B}}_{\lambda}^{(g)}$ will be considered in the selection pool of size $\mu + \lambda$.

B. Mutation

Within the set of self-adaptation parameters $\mathbf{s}$, we consider the parameter σ as the strength of the mutation. Since the maximum entropy is denoted as

$$\begin{aligned} \widetilde{\mathbf{w}} &= \mathbf{w} + \mathbf{z}, \\ \mathbf{z} &= \left(z_1, z_2, ..., z_N\right)^T, \end{aligned}$$

then we consider two types of mutations: first, the basic isotropic mutation $\mathbf{s} = \sigma$, and second, a Gaussian non-isotropic mutation $\mathbf{s} = \mathbf{C}$.

1) Isotropic Mutations: The basic isotropic mutation considers every element z_i as a standard normal distribution with zero mean and standard deviation σ:

$$\mathbf{z} = \sigma\left(\mathbb{N}_1(0,1), \mathbb{N}_2(0,1), ..., \mathbb{N}_N(0,1)\right)^T,$$

with a probability density function (*pdf*):

$$f_{\mathbf{Z}}\left(\mathbf{z}\right) = \frac{1}{\sqrt{2\pi}^N \sigma^N} e^{-\frac{\mathbf{z}^T\mathbf{z}}{2\sigma^2}}.$$

$\frac{1}{5}$th Rule $\sigma-$Adaptation. This adaptation for isotropic mutations is based on the probability of generation success P_s denoted as:

$$P_s = \frac{G_s}{G},$$

where G_s is the number of successful generations, and G is a fixed number of generations.

The method states that σ must be kept constant during G generations, and after $g > G$ the value of σ must be updated according to the following formula:

$$\sigma = \begin{cases} \sigma/a, & \text{if} \quad P_s > \frac{1}{5} \\ \sigma \cdot a, & \text{if} \quad P_s < \frac{1}{5} \\ \sigma, & \text{if} \quad P_s = \frac{1}{5} \end{cases}$$

then repeat for another G generations. Schwefel recommended using $0.85 \leq a < 1$.

Multiplicative (Log-Normal) $\sigma-$Adaptation. This adaptation method states that the mutation strength should change according to the "log-normal operator" [16]. The concept of a log-normal operator is elucidated by its role in generating a logarithmic normal distribution, symbolized as $\widehat{\sigma}$. This is expressed mathematically as $\widehat{\sigma}_l = \sigma_l e^{\tau\mathbb{N}(0,1)}$. The parameter τ, also known as the learning parameter, plays a pivotal role in determining the rate and precision of self-adaptation. Both theoretical and empirical studies provide guidance on the selection of the τ value. Specifically, it is recommended that τ be chosen in relation to the number of variables N, using the formula $\tau \approx \frac{1}{\sqrt{N}}$.

"Two-Point Rule" $\sigma-$Adaptation. The process of determining the value of σ employs a probabilistic mechanism, akin to a

coin flip. This method involves adjusting σ to either σ/α or $\sigma \cdot \alpha$, contingent on whether a value generated by a uniform random source exceeds or is less than or equal to 0.5. In this context, uniform$(0,1)$ denotes the uniform random source, while α is identified as a learning parameter. Beyer's research [16], posits that an optimal performance is achieved when α is set to 1+τ.

2) Non-isotropic Mutations: For the non-isotropic Gaussian mutation, we consider two methods of adaptation, described next.

Selective Covariance Matrix Adaptation. The selective correlation $\mathbf{C}$ is performed between the m randomly selected elements of $z_i, m \subset i$, such that the *pdf* is denoted as

$$f_{\mathbf{Z}}\left(\mathbf{z}_m\right) = \frac{1}{\sqrt{2\pi}^m \sqrt{|\mathbf{C}|}} e^{-\frac{1}{2}\mathbf{z}^T \mathbf{C}^{-1} \mathbf{z}},$$

where $|\mathbf{C}|$ is the determinant of the matrix $\mathbf{C}$, and $\mathbf{C}^{-1}$ denotes the inverse. Then the σ's are adapted as follows:

$$\mathbf{z}_m = \mathbf{C}\left(\sigma_1 \mathbb{N}_1(0,1), \sigma_2 \mathbb{N}_2(0,1), ..., \sigma_m \mathbb{N}_m(0,1)\right)^T,$$

and the remaining $i - m$ elements of $\mathbf{z}_{i-m}$ are adjusted according the adaptation method introduced next.

Extended Multiplicative (Log-Normal) $\sigma-$Adaptation. The previously discussed log-normal operator can be expanded as follows:

$$\widehat{\sigma} = e^{\tau 0 \mathbb{N} 0(0,1)} \left(\sigma 1 e^{\tau \mathbb{N} 1(0,1)}, ..., \sigma N e^{\tau \mathbb{N} N(0,1)}\right).$$

This extension introduces a general mutative multiplier with a learning parameter of τ_0, and individual mutations for each coordinate with a learning parameter of τ. Each component of σ undergoes independent mutation, and subsequently, the entire vector is mutatively scaled by the random factor $e^{\tau 0 \mathbb{N} 0(0,1)}$. This method facilitates the learning of axes-parallel mutation ellipsoids. The recommended learning parameters are:

$$\tau_0 = \frac{c}{\sqrt{2N}},$$

and

$$\tau = \frac{c}{\sqrt{2\sqrt{N}}},$$

where $c = 1$ is a suitable choice, as suggested in [13].

C. Reproduction

For the reproduction operator, we consider $2 < \rho \leq \mu$; thus, we will choose ρ parents for random reproduction with a uniform distribution. The parents taking part in the production of the offspring are

$$\Psi = \left(\mathfrak{a}_{i_1}, ..., \mathfrak{a}_{i_r}, ..., \mathfrak{a}_{i_\rho}\right),$$

for $r \in \{1, ..., \rho\}$, and i_r =uniform$(1, \mu)$.

D. Recombination

We will base the recombination operator on the *genetic repair* (GR) hypothesis introduced by Bayer [13]. Having that $\rho > 2$, we will have *multirecombination*, and we will be using *intermediate $\rho-$recombination*, and *dominant $\rho-$recombination*.

1) Genetic Repair (GR): In the 1990s, research into ES [13], [16], unveiled a recombination mechanism known as the genetic repair (GR) effect. This theory suggests that recombination identifies and extracts similarities from the parent genomes. It is a reasonable assumption that components of the parent genomes that exhibit similarity are likely to have a higher probability of contributing positively to the fitness of the offspring, given that these components are present in the fittest, or selected, individuals. From the standpoint of maximum entropy, the optimal strategy for a variation operator is to conserve these beneficial components. Other components, in contrast, may be less pertinent or even detrimental. This theoretical framework gives rise to the intermediate $\rho-$recombination concept.

2) Intermediate $\rho-$Recombination and Dominant $\rho-$Recombination: The process of intermediate $\rho-$recombination and dominant $\rho-$recombination, both integral to the generation of a descendant $\mathbf{r}$, can be elucidated as follows:

In the case of intermediate $\rho-$recombination, a descendant 'r' is produced based on the centroid of rho randomly chosen parent entities. This process can be mathematically represented as:

$$\mathbf{r} = \langle \mathbf{w} \rangle_\rho = \frac{1}{\rho} \sum_{\nu=1}^{\rho} \mathbf{w}_\nu.$$

On the other hand, dominant $\rho-$recombination, which also adheres to the Genetic Repair (GR) hypothesis, formulates a new descendant $\mathbf{r}$ by randomly selecting the $k-$th component of an individual m_k, chosen uniformly at random. This can be mathematically depicted as:

$$\mathbf{r} = \langle \mathbf{w}_{m_k} \rangle_k,$$

where m_k = uniform $(1, \rho)$.

V. Experiments and Results

The process of experimentation and the results obtained are discussed in detail next.

A. Design of Experiments

In our study, we conducted a comprehensive set of experiments comprising eight basic combinations, each varying in several parameters. The parameters included the number of parents (μ), ranging from 10 to 100 in increments of 10, yielding ten distinct combinations. Similarly, the number of parents for recombination (ρ) varied from 10 to 100 in increments of 10, resulting in ten combinations.

The number of children (μ) was set to 10, 100, or 1000, providing three unique combinations. The initial σ value varied from 1 to 10, creating ten combinations. The number of generations to stop (g) was also varied from 10 to 100 in increments of 10, resulting in ten different combinations.

The mutation operator was varied in five distinct ways, including isotropic and non-isotropic variations. The isotropic variations included the $\frac{1}{5}$th Rule σ-Adaptation, Multiplicative (Log-Normal) σ-Adaptation, and the "Two-Point Rule" σ-Adaptation. The non-isotropic variations included the Selective

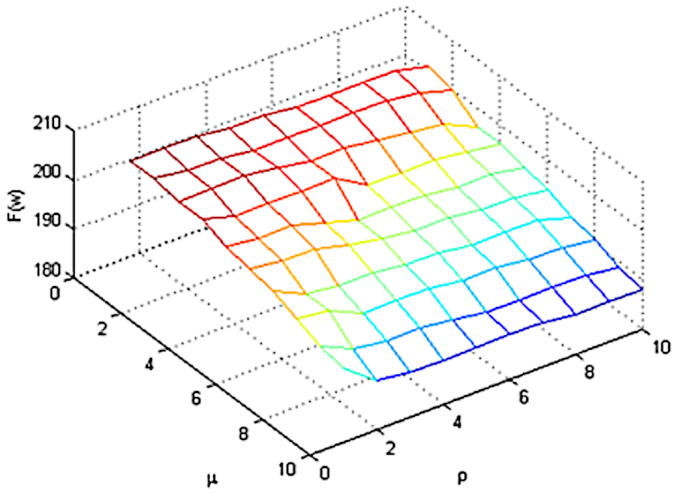


Fig. 3. Fitness behavior varying μ and ρ.

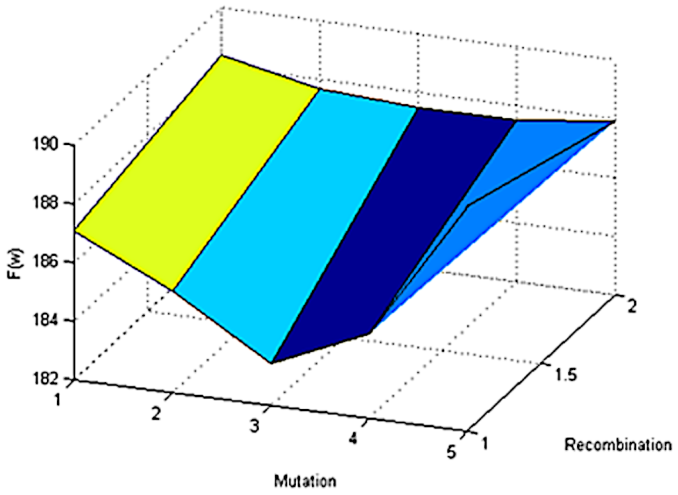


Fig. 4. Fitness behavior varying μ and the type of recombination (1=intermediate, 2=dominant).

Covariance Matrix Adaptation for a fixed size of 100×100 elements and the Extended Multiplicative (Log-Normal) σ-Adaptation.

The recombination operator was varied in two ways: Intermediate ρ-Recombination and Dominant ρ-Recombination.

In total, these variations resulted in 300,000 unique experiments, calculated as $(10)(10)(3)(10)(10)(5)(2)$.

These experiments were performed on a classic benchmark dataset, namely the two spirals dataset [18], to test non-trivial 2D class separation. The architecture used for this dataset was $[2, 128, 64, 32, 16, 8, 4, 1]$.

1) Deep Neural Network Validation: The experiments yield an initialized neural network, which is trained using the resilient backpropagation algorithm, denoted as the *trainrp* method. The Deep Neural Network (DNN) training is conducted ten times to observe two key events. The first event is the time taken to reach the desired minimum error, and the second event is the minimum error reached after the desired number of epochs have been completed.

2) Termination Condition: The termination condition forms a crucial aspect of these experiments. We propose that all experiments should have termination conditions for both the deep neural networks (BP) and the Evolution Strategies (ES). The conditions include achieving an error below 10^{-3} for both BP and ES. In terms of iterations, for BP, the termination condition is set at $10,000$ epochs. In contrast, ES is set at 100 generations, following the experiments in [7] and our previous work [19].

The architectures of the deep neural networks are determined by the computing and memory capabilities of the equipment used to perform these experiments.

B. Results

The behavior of the $(\mu/\rho + \lambda)-$ES for the spiral dataset is illustrated in several figures. The data obtained were averaged over the 300,000 experiments for those variables not depicted in the figures.

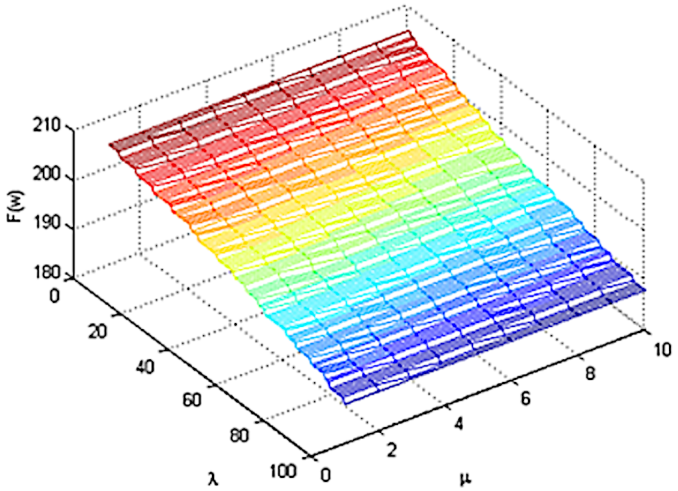


Fig. 5. Fitness behavior varying λ and μ.

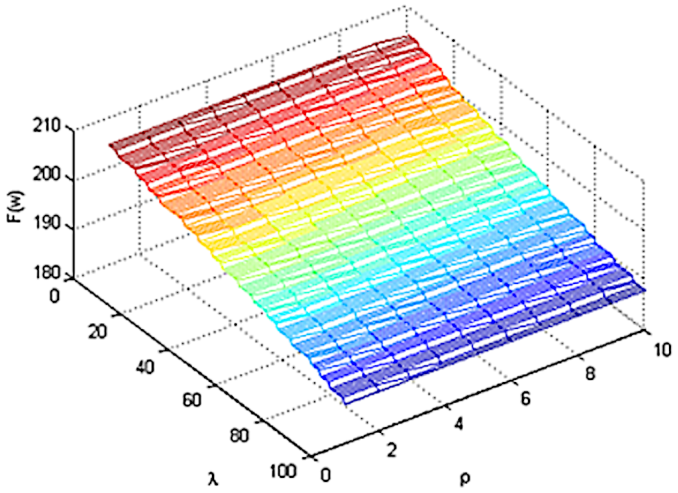


Fig. 6. Fitness behavior varying ρ and λ.

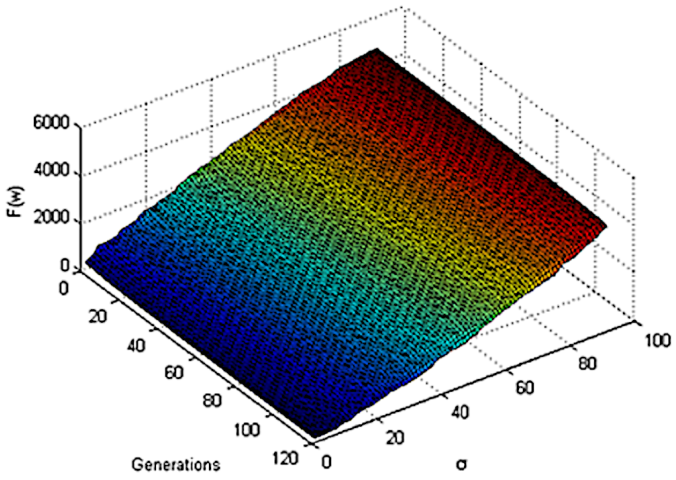


Fig. 7. Fitness behavior varying the mutation strength σ across generations.

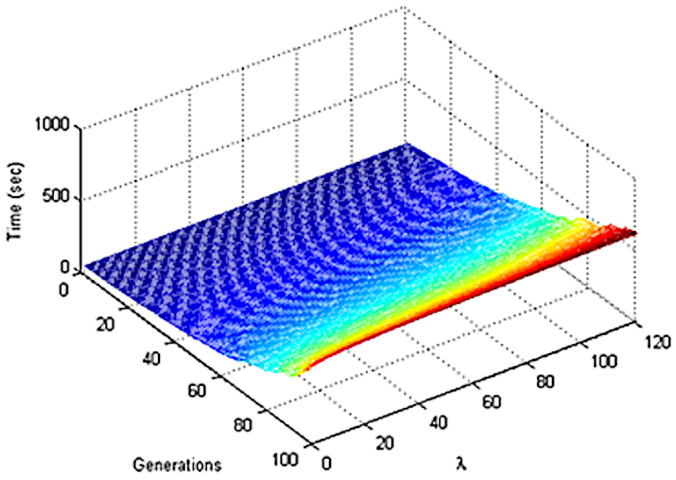


Fig. 8. Time behavior varying the number of generations and the number of children λ.

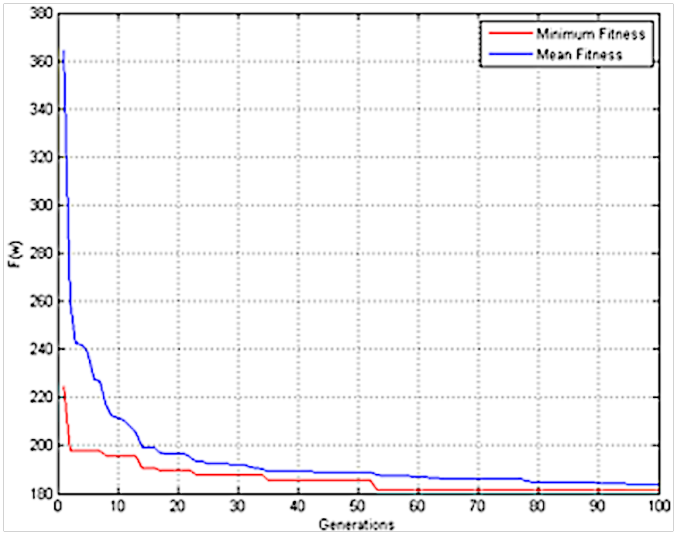


Fig. 9. An example of the best configuration: $(10/10+30)-$ES.

Fig. 3 demonstrates the impact on the fitness $F(\widehat{\mathbf{w}})$ while varying μ and ρ. The influence of μ versus the recombination type is depicted in Fig. 4, while λ versus μ is shown in Fig. 5. Fig. 6 presents the variation of ρ and λ, and Fig. 7 illustrates the variation of generations versus mutation strength σ. The response in time, measured by the most critical factors, the number of generations and the number of children λ, is shown in Fig. 8.

The results indicate that the minimum fitness was achieved using the configuration of $(10/10+30)-$ES, Isotropic Mutation with the $\frac{1}{5}$th rule, initial Mutation Strength $\sigma = 1$, and Intermediate-ρ recombination. Fig. 9 depicts an experiment using this configuration.

The results of several realizations using the configuration $(10/10+30)-$ES and employing cross-validation indicate that initializing the weights with our ES achieves a Mean Squared Error (MSE) of 0.018, for a 100% accuracy in 180 seconds, as shown in Fig. 10.b and Fig. 11.b. Compared to the Nguyen-Widrow method, which achieves an error of 0.009 in 127 epochs and 4.9 seconds (Fig. 10.c and Fig. 11.c), our method performs worse. However, when compared to the traditional zero-initialization method, which achieves an error of 0.068 for a 98% accuracy in 298 seconds (Fig. 10.a and Fig. 11.a), our method performs better.

VI. Related Work

Evolution Strategies (ES) is a robust optimization algorithm that can be applied to various domains. Different versions of ES have been developed, such as $(1+1)$-ES, Higher Order $(\mu/\rho, \lambda)$-ES, and Niching $\kappa(\mu/\rho+\lambda)$-ES [20]. These versions of ES provide different strategies for balancing exploration and exploitation during the optimization process.

In the context of global optimization, researchers have proposed hybrid algorithms that combine ES with other techniques to improve performance. For example, a hybrid algorithm that combines a modified Nelder-Mead method with a self-adaptive evolution strategy has been developed [21]. This algorithm incorporates an adaptive contraction criterion to enhance global exploration ability.

The theoretical analysis of ES has also been explored. The (μ, λ)-Theory analyzes the multimembered ES for real-valued, high-dimensional parameter spaces [22]. This analysis provides insights into the behavior of ES and its application to optimization problems.

Furthermore, ES has been applied to constrained evolutionary optimization. One study focused on (μ, λ)-Differential Evolution, which is an extension of ES, and proposed an improved adaptive trade-off model [23]. This model enhances the algorithm's exploration ability by exploiting the population's feasibility proportion.

However, as far as we know, there are no works on $(\mu/\rho+\lambda)$-ES that have been studied to better position a dense, deep neural network in a parameter space that can aid backpropagation reach a good quality local minima.

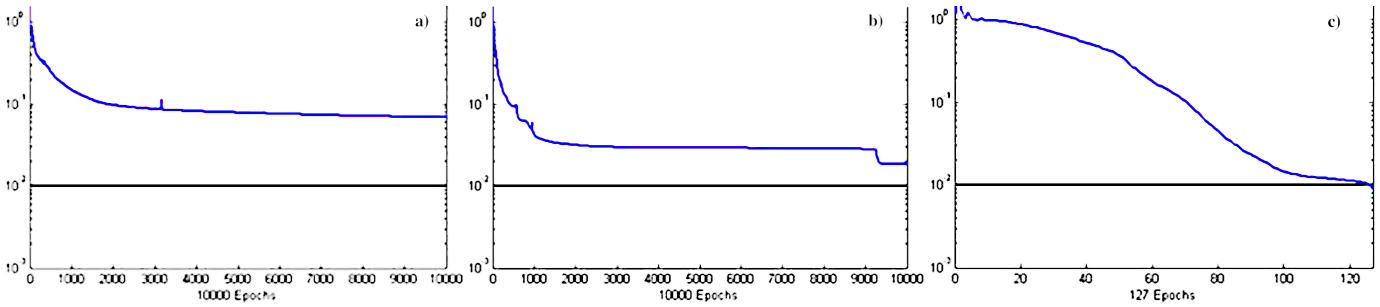
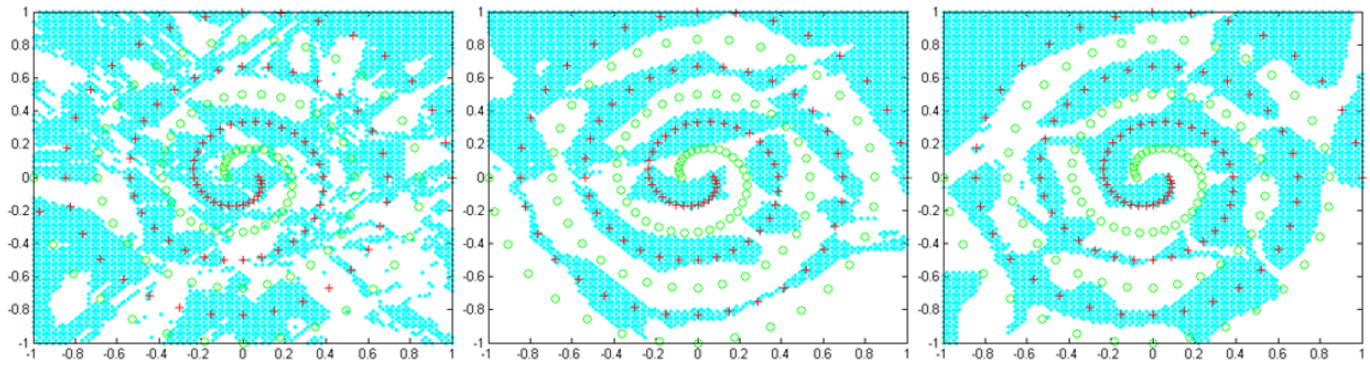


Fig. 10. Training performance: a) BP training with zero weight initialization. b) BP training with $(10/10+30)-$ES initialization. c) BP training with Nguyen-Widrow method.



a) After BP with zero initialization. b) After BP and $(10/10+30)-$ES init. c) After BP with Nguyen-Widrow init.

Fig. 11. Mapping dataset and hyperplane decision boundaries across different settings.

VII. Conclusions

While there exists research that employs Evolution Strategies (ES) to train neural networks [4]–[6], [8], [9], the integration of both schemes within a deep neural network framework remains an open problem. This is primarily because, as demonstrated in [7], ES performs comparably to Backpropagation (BP) only for smaller problems.

In this study, we have extended the application of ES to high-dimensional problems by using it to initialize the weights of a Deep Neural Network (DNN). Our experiments have shown that the proposed ES can overcome local minima and converge towards near-optimal global solutions. Following the ES initialization, we employed traditional BP gradient methods to further refine the weights based on the initial set provided by ES.

A significant finding of this research is that using Evolution Strategies can reduce the computational time required by traditional gradient-based backpropagation learning methods. This is achieved by providing an initial set of weights close to the good-quality local minima, enabling the network to converge faster than random or zero-weight initialization approaches. This approach offers a promising direction for future research in the efficient training deep neural networks.

Acknowledgment

The ML model is based upon work supported in part by the National Science Foundation under Grant 2210091.

References

[1] X. Yu, "Can backpropagation error surface not have local minima," *Neural Networks, IEEE Transactions on*, vol. 3, no. 6, pp. 1019–1021, 1992.

[2] E. Chen, X. Yang, H. Zha, R. Zhang, and W. Zhang, "Learning object classes from image thumbnails through deep neural networks," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*. IEEE, 2008, pp. 829–832.

[3] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[4] S.-C. Ng, L. Shu-Hung, and A. Luk, "A hybrid algorithm of weight evolution and generalized back-propagation for finding global minimum," in *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, vol. 6. IEEE, 1999, pp. 4037–4042.

[5] S. Ng, S. Leung, and A. Luk, "Evolution strategies on connection weights into modified gradient function for multi-layer neural networks," in *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 3. IEEE, 2005, pp. 1371–1376.

[6] S.-C. Ng, S. Leung, and A. Luk, "A generalized backpropagation algorithm for faster convergence," in *Neural Networks, 1996., IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 409–413.

[7] M. Mandischer, "A comparison of evolution strategies and backpropagation for neural network training," *Neurocomputing*, vol. 42, no. 1, pp. 87–117, 2002.

[8] J. Hagg, B. Curuklu, B. Akan, and L. Asplund, "Gesture recognition using evolution strategy neural network," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, 2008, pp. 245–248.

[9] A. Berlanga, P. Isasi, A. Sanchis, and J. Molina, "Neural networks robot controller trained with evolution strategies," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 1. IEEE, 1999, p. 419.

[10] F. Kursawe, "A variant of evolution strategies for vector optimization," in *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, ser.

Lecture Notes in Computer Science, vol. 496. Springer, 1991, pp. 193–197.

[11] P. Rivas-Perea, J. Cota-Ruiz, D. G. Chaparro, A. Q. Carreón, F. J. E. Aguilera, and J.-G. Rosiles, "Forecasting the demand of short-term electric power load with large-scale lp-svr," *Smart Grid and Renewable Energy*, vol. 4, pp. 449–457, 2013.

[12] S. Haykin, *Neural Networks and Learning Machines*. Prentice Hall, 2008.

[13] H.-G. Beyer, *The theory of evolution strategies*. Springer Science & Business Media, 2001.

[14] T. Back, F. Hoffmeister, and H. Schwefel, "A survey of evolution strategies," in *Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 2–9.

[15] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.

[16] H. Beyer and H. Schwefel, "Evolution strategies –a comprehensive introduction," *Natural Computing: an international journal*, vol. 1, pp. 3–52, 2002.

[17] E. Cantú-Paz, J. A. Foster, K. Deb, D. Lawrence, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson *et al.*, *Genetic and Evolutionary Computation-GECCO 2003: Genetic and Evolutionary Computation Conference, Chicago, IL, USA, July 12-16, 2003, Proceedings, Part I*. Springer, 2003, vol. 2723.

[18] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in *Proceedings 1988 Connectionist Models Summer School*. Morgan Kaufmann, 1988, pp. 52–59.

[19] M. I. Chacon M. and P. Rivas-Perea, "Performance analysis of the feedforward and som neural networks in the face recognition problem," in *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP 2007)*. IEEE, 2007, pp. 313–318.

[20] M. Jaindl, A. Köstinger, C. Magele, and W. Renhart, "Multi-objective optimization using evolution strategies," *Facta universitatis-series: Electronics and Energetics*, vol. 22, no. 2, pp. 159–174, 2009.

[21] N. Boukhari, F. Debbat, N. Monmarché, and M. Slimane, "An efficient hybrid evolution strategy algorithm with direct search method for global optimization," *International Journal of Organizational and Collective Intelligence (IJOCI)*, vol. 9, no. 3, pp. 63–78, 2019.

[22] H.-G. Beyer, "Toward a theory of evolution strategies: The (μ, λ)-theory," *Evolutionary Computation*, vol. 2, no. 4, pp. 381–407, 1994.

[23] Y. Wang and Z. Cai, "Constrained evolutionary optimization by means of (μ+ λ)-differential evolution and improved adaptive trade-off model," *Evolutionary Computation*, vol. 19, no. 2, pp. 249–285, 2011.