mathematics

Article

Differential Evolution Optimal Parameters Tuning with Artificial Neural Network

Manu Centeno-Telleria [1,*], Ekaitz Zulueta [1,*], Unai Fernandez-Gamiz [2], Daniel Teso-Fz-Betoño [1] and Adrián Teso-Fz-Betoño [1]

[1] System Engineering and Automation Control Department, University of the Basque Country (UPV/EHU), Nieves Cano 12, 01006 Vitoria-Gasteiz, Spain; daniel.teso@ehu.eus (D.T.-F.-B.); ateso001@ehu.eus (A.T.-F.-B.)
[2] Nuclear Engineering and Fluid Mechanics Department, University of the Basque Country (UPV/EHU), Nieves Cano 12, 01006 Vitoria-Gasteiz, Spain; unai.fernandez@ehu.eus
* Correspondence: mcenteno004@ehu.eus (M.C.-T.); ekaitz.zulueta@ehu.eus (E.Z.)

Abstract: Differential evolution (DE) is a simple and efficient population-based stochastic algorithm for solving global numerical optimization problems. DE largely depends on algorithm parameter values and search strategy. Knowledge on how to tune the best values of these parameters is scarce. This paper aims to present a consistent methodology for tuning optimal parameters. At the heart of the methodology is the use of an artificial neural network (ANN) that learns to draw links between the algorithm performance and parameter values. To do so, first, a data-set is generated and normalized, then the ANN approach is performed, and finally, the best parameter values are extracted. The proposed method is evaluated on a set of 24 test problems from the Black-Box Optimization Benchmarking (BBOB) benchmark. Experimental results show that three distinct cases may arise with the application of this method. For each case, specifications about the procedure to follow are given. Finally, a comparison with four tuning rules is performed in order to verify and validate the proposed method's performance. This study provides a thorough insight into optimal parameter tuning, which may be of great use for users.

Keywords: evolutionary algorithm; differential evolution; parameter tuning; artificial neural network

Citation: Centeno-Telleria, M.; Zulueta, E.; Fernandez-Gamiz, U.; Teso-Fz-Betoño, D.; Teso-Fz-Betoño, A. Differential Evolution Optimal Parameters Tuning with Artificial Neural Network. *Mathematics* **2021**, *9*, 427. https://doi.org/10.3390/math9040427

Academic Editor: Fabio Caraffini

Received: 20 January 2021
Accepted: 18 February 2021
Published: 21 February 2021

1. Introduction

Optimization is the act of transforming something as conveniently as possible. From a mathematical perspective, optimization is the process of finding the global maximum or minimum of an objective function. The conventional techniques such as gradient-based methods or deterministic hill climbing are not generally competent to solve nonlinear global optimization problems [1]. In that context, evolutionary algorithms (EAs) have been widely employed. EAs intrinsic stochastic component allows finding global optimal points surpassing the conventional techniques. The field of evolutionary algorithms is mostly constituted by genetic algorithm (GA), evolutionary programming (EP), evolution strategies (ESs) and differential evolution (DE), see Das et al. [2].

The problems that intelligent algorithms such as DE and EAs are capable of solving, all of them belonging to swarm algorithms, are complex. These algorithms do not require gradient calculations of the function to be optimized. Furthermore, it does not require practically any particularly strong characteristics in the definition of optimization. For example, it is possible to perform these algorithms with non-continuous and non-differentiable cost functions. Additionally, it is also possible to use without having an analytical expression that directly links the optimization variables with the cost function. Moreover, they are optimization algorithms that easily avoid local minima by their nature. Complex optimization problems are shown in the following references [3–7]. In differential equations optimization, possible solutions must be defined by a finite parameter set. Once possible solutions are described by a parameter set, this parameter set is optimized in order to minimize a cost function.

Among EAs, DE is a simple and competitive algorithm proposed by Storn and Price [8,9]. In DE, D-dimensional individuals are iteratively mixed to create new individuals and the best of them survive. A standard DE employs three main parameters: population size NP, mutation factor F and crossover rate CR. The setting of these parameters largely influences the algorithm's performance, and the optimal setting depends on the problem to be faced, see Mohamed et al. [10]. Consequently, in real-world problems, the parameters are generally tuned once the objective function is designed [11].

Several studies analyze the relationship between parameters and suggest the rules of tuning them. The most popular four tuning rules are the following ones. Storn and Price [9] proposed the settings NP = 10D, $F \in [0.5, 1]$ and $CR \in [0.8, 1]$. According to Gämperle et al. [12], a reasonable choice for population size is between NP = 3D and NP = 8D, a good initial choice for the mutation factor is F = 0.6 and crossover rate is $CR \in [0.3, 0.9]$. Based on Rönkkönen et al. [13], a reasonable choice for population size is between NP = 2D and NP = 40D, $F \in (0.4, 0.95]$ and when the objective function is separable $CR \in (0, 0.2)$, whereas in non-separable functions $CR \in (0.9, 1)$. Zielinsky et al. [14] reported that, in many cases, the best results are obtained with the settings of $F \geq 0.6$ and $CR \geq 0.6$. As can be seen, the indications of these investigations differ among them. According to Sarker et al. [15], a tedious trial-and-error approach for tuning parameters is commonly used.

Numerous scholars have proposed adaptive parameter control or adjusting the parameter values during evolution to address this situation. According to the study of Al-Dabbagh et al. [16], the adaptive DE algorithms are classified into two main groups: (1) algorithms with a single DE strategy and (2) algorithms with multiple DE strategy. The DE strategy is composed of mutation and crossover strategies. The algorithms with a single DE strategy use a single mutation and crossover operation during the evolution, whereas the algorithms with multiple DE strategy use various mutations and crossover operations.

Within group 1 the following are the most recognized algorithms. jDE adjusted the parameters F and CR using a self-adaptive scheme [17]. MDE_*p*BX, guided by the knowledge of the last generation successful values, updates the values of F and CR [18]. Other scholars adapt F and CR using fuzzy logic controllers [19–21]. JADE adjusts the parameters F and CR based on Cauchy and normal distribution [22]. A modified version of JADE known as JADE_sort assigned smaller CR values to individuals with better fitness values [23]. SHADE is an improved version of JADE in which the updating of parameters is based on the historical memory of successful solutions [24]. LSHADE extends SHADE with a simple deterministic linear reduction in the population size [11]. Various modifications of SHADE and JADE have been developed in recent years, and those are precisely compared and explained in [25].

Within group 2 the following are the most recognized algorithms. SaDE works with adapted mutation strategy and parameters F and CR [26]. In SAKPDE, each individual has its own parameters F and CR, mutation strategy and crossover strategy, and prior knowledge and opposition learning are used to guide the evolution [27]. EFADE is an enhanced fitness-adaptive algorithm that uses two novel adaptation schemes to update the control parameters F and CR [28]. CoDE is a composition of three mutation strategies and three combinations of parameters F and CR [29]. In EPSDE, the mutation strategy and F and CR are randomly selected from two pools [30]. EDEV is a multi-population based framework (MPF) that ensembles JADE, CoDE and EPSDE to cooperate during the evolution [31]. MPEDE is also a multi-population based approach, but it uses unequal sub-population sizes [32]. Besides these techniques, there are numerous algorithms with multiple DE strategy, detailed in [16].

Adaptive DE algorithms have achieved high-precision performance; however, their implementation may be complicated, and the number of function evaluations generally increases [18]. The performance of DE (P) is highly dependent on the algorithm parameters (NP, F and CR) and DE strategy (mutation and crossover operations) [15,29]. This relationship can be mathematically expressed by Equation (1). Specifically, adaptive techniques do not directly focus on searching the function f(·), but rather normally test various parameter combinations with distinct DE strategies, and the best of them are applied during the evolution.

$$P = f(NP, F, CR, DEstrategy) \tag{1}$$

In the current study, the utilization of an artificial neural network (ANN) is proposed to learn the function f(·) and afterward to extract the optimal combination of parameters (NP, F and CR). An ANN consists of many interconnected simple functional units (neurons) that perform as parallel information-processors and approximate the function that maps inputs to outputs [33]. ANNs potential to solve problems with high performance and the ability to adapt to different problems have been implemented in numerous fields such as autonomous driving [34,35], solar and wind energy systems [36,37] and financial time series forecasting [38]. The field of ANNs is in full motion, in that way to review its progress and application areas in real-world scenarios, see Abiodun et al. [39].

This work's main contribution is to present a consistent methodology for tuning the optimal parameters of DE. This methodology's heart is the use of ANN to learn the relationship between the algorithm's performance and the parameters. Knowledge on how to tune the optimal values of these parameters is scarce [15,23]. In that context, this paper suggests a manner to fill that knowledge gap. In the proposed methodology, once the objective function is designed, the experimental requisitions are defined and the DE strategy is chosen, four consecutive steps are applied: data-set generation, data-set normalization, ANN approach and best parameters extraction. In order to analyze the distinct cases that may arise, the proposed methodology is evaluated on a set of 24 objective functions from the Black-Box Optimization Benchmarking (BBOB) [40]. Finally, the validation of the proposed method is performed, on the one hand, checking if the neural network predictions are correct, and on the other, making a comparison with the four most common tuning rules: Storn and Price [9], Gämperle et al. [12], Rönkkönen et al. [13] and Zielinsky et al. [14].

The rest of this paper is organized as follows. In Section 2, the experimental set-up is defined. In Section 3, the basic DE algorithm and ANN model are presented. Section 4 introduces the proposed methodology in detail. In Section 5, the results are presented and discussed. Finally, the conclusions are drawn in Section 6.

2. Simulation Set-Up

The objective functions are defined and evaluated over $\mathbb{R}^D$. The global point search domain is given as $x_{opt} \in [-5, 5]^D$. In this study, the optimization problem is a minimization bi-dimensional ($D = 2$) problem. BBOB defines 24 noise-free real-parameter single-objective functions, which are detailed and described in [40]. These objective functions are composed of five groups.

The first group functions are separable (Fcn1-Fcn5). The second group functions have low or moderate conditioning (Fcn6-Fcn9). The third group functions have high conditioning and are unimodal (Fcn10-Fcn14). The fourth group functions have adequate global structure and are multimodal (Fcn15-Fcn19). The fifth group functions have weak global structure and are multimodal (Fcn20-Fcn24). Apart from the first group, all other objective functions are non-separable. The termination criteria for the DE is when 20 generations are reached. Therefore, the DE algorithm is forced to converge quickly to the optimal point. The required fitness precision for each objective function is $\Delta f = 10^{-2}$. Consequently, the target value to be achieved is defined as follows $f_t = f_{opt} + \Delta f$, where f_{opt} is the objective function's global optimal point value.

In this paper, the performance (P) expressed in Equation (1) is quantified by the parameter success rate (SR) defined in Equation (2). SR defines the probability of achieving the requirements, so it directly relates to the algorithm's effectiveness. The DE algorithm's intrinsic stochastic component provokes different results with the same parameter setting.

Therefore, the performance is quantified by the mean of satisfactory runs with respect to total independent runs. Satisfactory runs are the executions of DE that achieved the required f_t. Consequently and logically, higher values of SR are more interesting than lower values.

$$SR = \frac{s_{runs}}{t_{runs}} \tag{2}$$

where s_{runs} is the number of satisfactory DE runs, and t_{runs} is the total number of DE independent runs, which in this paper is 40.

All simulations are performed in MATLAB 2020b software with the system configuration of Intel core i7-8550U, 8 GB RAM, 1.80 GHz processor and 64 bit Windows 10 operating system.

3. Procedures

3.1. Differential Evolution Algorithm

In this section, the standard DE algorithm is introduced. DE is a robust population-based meta-heuristic search algorithm in which the population of D-dimensional individuals is used to optimize a problem. Each individual of the population is a candidate solution to the problem and is coded as a vector. The population in the G-generation is defined as $\{X_i^G = (X_{i,1}^G, X_{i,2}^G, \ldots, X_{i,D}^G), i = 1, 2, \ldots, NP\}$, where D is the dimensionality of the problem and NP is the population size. DE is composed of four steps, which are initialization, mutation, crossover and selection.

3.1.1. Initialization

In the initialization step, three main parameters of the algorithm are defined: population size (NP), mutation factor (F) and crossover rate (CR) [41]. Moreover, the initial population of NP individuals is randomly generated according to a uniform distribution within the search space. Once initialized, the DE algorithm performs mutation, crossover and selection operations iteratively until the user-defined stopping criteria are reached.

In this iterative process, new individuals are generated and evolved over generations. In this work, all generated individuals for the search space are feasible solutions since the problems are non-constrained. If it is necessary to handle non-feasible solutions, see Caraffini et al. [42].

3.1.2. Mutation Operation

In each G-generation, following a mutation strategy the mutation vectors $\{V_i^G = (V_{i,1}^G, V_{i,2}^G, \ldots, V_{i,D}^G), i = 1, 2, \ldots, NP\}$ are generated. In this paper, the most popular mutation strategy, 'DE/rand/1', is used. In this strategy, two vectors (individuals) are randomly chosen, their difference is multiplied by a mutation factor F, and the result is added to a third random vector. The difference vector automatically adapts to the scale of the optimized function, and that is the key success factor of the DE algorithm [43]. The 'DE/rand/1' strategy is defined as follows:

$$V_i^G = X_{r1}^G + F \cdot (X_{r2}^G - X_{r3}^G) \tag{3}$$

where $r1$, $r2$ and $r3$ are randomly selected distinct integers within the range $[1, NP]$ and are also different from i.

The most popular mutation strategies are defined in the next list. Each strategy affects population diversity differently; therefore, the search convergence rate might vary [44].

- 'DE/rand/2:'
$$V_i^G = X_{r1}^G + F \cdot (X_{r2}^G - X_{r3}^G) + F \cdot (X_{r4}^G - X_{r5}^G) \tag{4}$$
- 'DE/best/1:'
$$V_i^G = X_{best}^G + F \cdot (X_{r1}^G - X_{r2}^G) \tag{5}$$

- 'DE/best/2:'
$$V_i^G = X_{best}^G + F \cdot (X_{r1}^G - X_{r2}^G) + F \cdot (X_{r3}^G - X_{r4}^G) \tag{6}$$
- 'DE/current-to-best/1:'
$$V_i^G = X_i^G + F \cdot (X_{best}^G - X_{r1}^G) + F \cdot (X_{r2}^G - X_{r3}^G) \tag{7}$$
- 'DE/current-to-pbest/1:'
$$V_i^G = X_i^G + F \cdot (X_p^G - X_i^G) + F \cdot (X_{r1}^G - X_{r2}^G) \tag{8}$$

where $r1$, $r2$, $r3$, $r4$ and $r5$ are randomly selected distinct integers within the range $[1, NP]$ and are also different from i. X_{best}^G is the best individual of the current population, and X_p^G is randomly chosen as one of the top $100p\%$ individuals in the current population with $p \in (0, 1]$ [22].

The mutation factor F defines the search step of the optimization. According to Storn and Price, F range is in $[0, 2]$ [9]. Generally, smaller values of F are better when the population is closer to the global best value. In contrast, larger values of F are preferred when the population is far from the global best value [45].

New mutation strategies are proposed in recent years, such as random neighbor based mutation ('DE/neighbor/1') [46], new triangular mutation [28], mutation vector inspired from the biological phenomenon called Hemostasis [41], an enhanced mutation strategy with time stamp scheme [47] and 'DE/pbad-to-pbest-to-gbest/1' [48].

3.1.3. Crossover Operation

After the mutation operation, the crossover operation is performed to increase the diversity of mutation vectors. The crossover operator generates a trial vector $\{U_i^G = (U_{i,1}^G, U_{i,2}^G, \ldots, U_{i,D}^G), i = 1, 2, \ldots, NP\}$ between the interaction of the mutation vector V_i^G and the target vector X_i^G. In this paper, the classic binary crossover strategy defined in Equation (9) is used ('DE/rand/1/bin'). However, it is possible to use other crossover strategies, such as exponential crossover [49,50] or an eigenvector-based crossover operator [51].

$$U_i^G = \begin{cases} V_i^G, & \text{if } rand(0,1) \leq CR \\ X_i^G, & \text{otherwise} \end{cases} \tag{9}$$

The crossover rate CR reflects the probability in which the trial vector inherits mutation vector's values. According to Storn and Price, CR range is in [0, 1] [9]. CR has a direct influence on the diversity of the population and, therefore, on search convergence capacity [45].

3.1.4. Selection Operation

Following the crossover operation, the fitness value of the trial vector is calculated. Afterward, the trial vector U_i and the target vector X_i compete, and if U_i has a better fitness value than X_i, then U_i will replace X_i in the next generation. The selection operation is defined as follows:

$$X_i^{G+1} = \begin{cases} U_i^G, & \text{if } f(U_i) < f(X_i) \\ X_i^G, & \text{otherwise} \end{cases} \tag{10}$$

where $f(U_i^G)$ and $f(X_i^G)$ are the fitness values of U_i^G and X_i^G, respectively.

3.2. Artificial Neural Network Model

In this section, the mathematical model of an ANN under the DE environment is explained in detail. ANNs are composed of a set of artificial neurons and grouped in interconnected layers that, in their entirety, simulate the behavior of biological neural networks. The connection strength between neurons is quantified by layer weights (W). In this paper, the most popular ANN class called multilayer perceptron (MLP) is designed. The MLP contains at least three layers: an input layer, a hidden layer and an output layer.

As described in Section 2, the DE algorithm's performance (P) is quantified by the success rate (SR). As it is explained in Section 4, the SR value only varies with parameters (NP, F and CR) since the DE strategy is fixed during the data-set generation. The ANN learns the problem of mapping the inputs NP, F and CR to the output SR. Based on this reasoning, in this particular case, the 3-K-1 MLP topology is designed. The hidden layer neuron number (K) is obtained with an experimental approach in the ANN training phase.

Figure 1 illustrates the created MLP in this paper. Inside MLP, additional constant units known as bias are applied to decide if a neuron can be fired or not [52]. The main elements of the designed MLP are as follows:

- Input layer neurons are $\{X_1, X_2$ and $X_3\}$, bias is $\{X_0\}$ and first layer weights are defined in $W^{(1)}$;
- Hidden layer K neurons outputs are $\{a_1^{(2)}, a_2^{(2)}, \ldots, a_K^{(2)}\}$, bias is $\{a_0^{(2)}\}$ and second layer weights are defined in $W^{(2)}$;
- The output layer neuron's output is $a_1^{(3)}$.

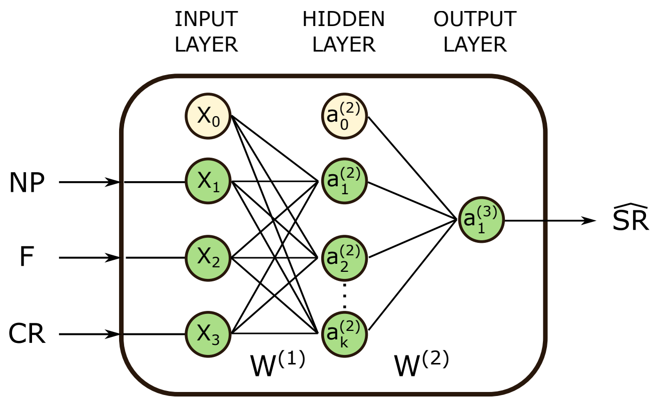


Figure 1. The designed ANN-MLP with dimension 3-K-1. Inputs of the neural network are DE parameters NP, F and CR, and the output is the prediction of SR expressed as $\widehat{SR}$. The hidden layer neuron number (K) is obtained in the ANN training process.

In MLP, the information is propagated from the input layer to the output layer. In that propagation, mathematical operations are performed to achieve a final prediction output value ($\widehat{SR}$). The input layer only transmits the information, whereas the hidden and output layer apart from transmitting also transform it by using an activation function. The parameter values of NP, F and CR are just positive, so in the hidden layer a sigmoid activation function and in the output layer a linear activation function are used. The mathematical model of network propagation is explained below.

Initially, the network takes the inputs and multiplies them by the weights of the first layer. The result generated by the multiplication passes through the sigmoid activation function to the second layer. The outputs of the hidden layer are defined as follows:

$$a_1^{(2)} = g(W_{10}^{(1)} X_0 + W_{11}^{(1)} X_1 + W_{12}^{(1)} X_2 + W_{13}^{(1)} X_3) \tag{11}$$

$$a_2^{(2)} = g(W_{20}^{(1)} X_0 + W_{21}^{(1)} X_1 + W_{22}^{(1)} X_2 + W_{23}^{(1)} X_3) \tag{12}$$

$$\vdots$$

$$a_K^{(2)} = g(W_{K0}^{(1)} X_0 + W_{K1}^{(1)} X_1 + W_{K2}^{(1)} X_2 + W_{K3}^{(1)} X_3) \tag{13}$$

where $W_{ji}^{(1)}$ is the weight value from input neuron i to hidden neuron j.

Abbreviating in vectors and matrices,

$$a^{(2)} = g(W^{(1)}X) \tag{14}$$

where the dimension of the weights of the first layer is $W^{(1)} \in \mathbb{R}^{Kx4}$, the dimension of the input vector is $X \in \mathbb{R}^{4x1}$, the dimension of the hidden layer output vector is $\mathbb{R}^{Kx1}$ and the sigmoid activation function is $g(s) = \frac{1}{1+e^{-s}}$.

Subsequently, the hidden layer outputs are multiplied by the weight of the second layer, and it passes through the linear activation function to the output of the network. The output of the network is the predicted value $\widehat{SR}$ defined as follows:

$$\begin{aligned} \widehat{SR} = a_1^{(3)} &= f(W_{10}^{(2)}a_0^{(2)} + W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + \cdots + W_{1K}^{(2)}a_K^{(2)}) \\ &= f(W^{(2)}(a_0^{(2)} + a^{(2)})) \\ &= f(W^{(2)}z^{(2)}) \end{aligned} \tag{15}$$

where the dimension of the weights of the second layer is $W^{(2)} \in \mathbb{R}^{1x(K+1)}$, the dimension of the input vector is $z^{(2)} \in \mathbb{R}^{(K+1)x1}$, the output predicted value is $\widehat{SR} \in \mathbb{R}^{1x1}$ and the linear activation function is $f(s) = s$.

In conclusion, it is clearly observed that the output prediction value ($\widehat{SR}$) depends directly on input parameters values (NP, F and CR), first and second layer weight values ($W^{(1)}, W^{(2)}$) and their dimension (K hidden neurons). Weights values and hidden layer neuron number (K) are defined in the neural network training phase.

4. Proposed Method

In solving an optimization problem with DE, the objective function and the DE algorithm are designed first (the left sub-block of Figure 2). In this work, as stated in Section 2, 24 single-objective functions are performed to check the proposed method in distinct environments. On the other hand, the DE algorithm is designed as defined in Section 3.1. After this, the DE algorithm parameters (NP, F, and CR) are tuned. A tedious trial-and-error approach for tuning the best DE algorithm parameters is commonly used in real-world problems. To tackle this issue, a consistent methodology for tuning the optimal parameters of DE is presented in this work.

The proposed methodology comprises four consecutive steps: data-set generation, data-set normalization, ANN approach and best parameters extraction. These four steps are graphed in Figure 2. For each objective function, each step is executed just one time. As a result of this process, the optimal DE parameters (NP, F and CR) are obtained for each objective function.

In the data-set generation, the DE algorithm is executed with various NP, F and CR parameter combinations, and the achieved SR is saved. In the data-set normalization, the obtained combinations of NP, F, CR and SR are normalized to help the learning process of the ANN. In the ANN approach, the training and testing of the ANN are performed. Lastly, using the best ANN of the previous step, the optimal parameters of DE (NP, F and CR) are extracted.

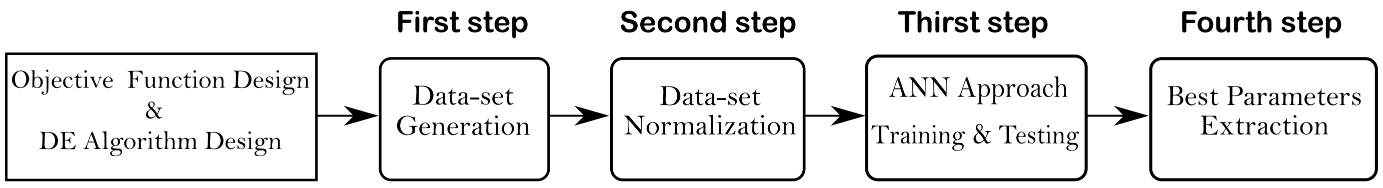

First step
Second step
Thirst step
Fourth step
Objective Function Design & DE Algorithm Design
Data-set Generation
Data-set Normalization
ANN Approach Training & Testing
Best Parameters Extraction

Figure 2. The block diagram of the proposed method. The method is applied once the objective function and DE algorithm are designed. Each step is executed just one time for each objective function. As a result of this method, the optimal DE parameters (NP, F and CR) are obtained for each objective function.

As declared before, the ANN learns the function $f(\cdot)$ defined in Equation (1). Once the objective function and DE strategy are chosen, the performance of DE (P) quantified by SR only varies with parameters NP, F and CR. Therefore, the ANN exactly learns to map the input parameters (NP, F and CR) with the output SR.

As concluded in Section 3.2, the network's prediction ($\widehat{SR}$) depends directly on the weight values ($W^{(1)}$, $W^{(2)}$), and these values are calculated in the training approach of the neural network. The training procedure requires a training set that relates the network's input parameters with the output value. For that reason, initially, a data-set of examples is generated.

4.1. Data-Set Generation

In this step, the DE algorithm with different combinations of input parameters (NP, F and CR) is performed, and their output (SR) is calculated and saved. The parameter NP is taken from [10, 100] by steps of 10. The parameter F is taken from [0.1, 2] by steps of 0.1. The parameter CR is taken from [0.1, 1] by steps of 0.1. As a consequence of this process, a data-set of 2000 combinations of NP, F and CR with their respective SR is generated.

It should be noted that it is possible to take bigger or smaller steps. With smaller steps, more combinations of parameters are achieved, and the neural network's learning will be more precise, although the time of the data-set generation will be longer. In this study, for each objective function, the generation of the data-set required an average of six hours.

4.2. Data-Set Normalization

Once the data-set is generated, the normalization of the NP and F parameters is performed as defined in Equations (16) and (17), respectively. The normalization of CR and SR has not been performed since they are already within the range [0, 1]. After the normalization, the four parameters used in ANN (NP, F, CR and SR) are in the same range, and that helps the ANNs accurate learning.

$$NPnorm = \frac{NP - NPmin}{NPmax - NPmin} \tag{16}$$

where $NPnorm$ is the normalized NP, $NPmax$ is the highest NP value and $NPmin$ is the lowest NP value.

$$Fnorm = \frac{F - Fmin}{Fmax - Fmin} \tag{17}$$

where $Fnorm$ is the normalized F, $Fmax$ is the highest F value and $Fmin$ is the lowest F value.

4.3. ANN Approach

In this third step, firstly, the ANN training is performed in order to learn the relationship between the input parameters ($NPnorm$, $Fnorm$ and CR) with the desired output (SR). The training is resumed in the left scheme of Figure 3. Secondly, the trained ANN is validated in the testing procedure. The testing procedure is resumed in the right scheme of Figure 3.

The data-set obtained after the second step of the method is divided into three sets: training data-set, validation data-set and testing data-set. In the training procedure, the training data-set and validation data-set are used. The training set, which represents 75% of the data-set, is used to choose the best layer weights ($W^{(1)}$ and $W^{(2)}$). The validation set, which represents 12.5% of the data-set, is used to prevent the ANNs from overfitting. In the testing procedure, the test set, which represents the remaining 12.5% of the data-set, is used to evaluate and validate the trained ANNs performance.

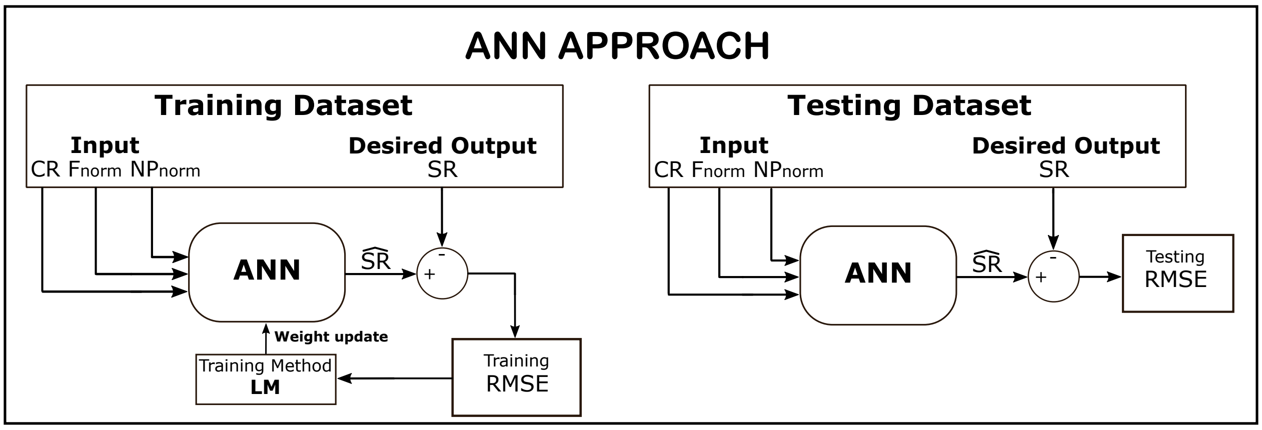


Figure 3. Schematic illustration of the proposed method's third step, called ANN approach. The ANN approach is composed of the training and the testing processes of ANN. The left scheme defines the training of the ANN. The right scheme defines the testing of the ANN.

The training of ANNs has been performed using Neural Network Toolbox from Matlab software. At the beginning of the training procedure, each layer's weights are initialized with the Nguyen–Widrow method [53]. Iteratively, the training set is propagated in the network and the network's prediction error is calculated. To conclude the training procedure, the weights of the layers are updated using the Levenberg–Marquard (LM) algorithm [54]. With this iterative procedure, the neural network has the ability to approximate non-linear functions. The training's stopping condition is the gradient level criteria of 10^{-8} and six validation checks. The metric to evaluate the neural network's performance is the root-mean-squared error (RMSE) in this work.

As stated in Section 3.2, the ANNs performance depends on weight values and the number of hidden neurons (K). In this paper, K is defined by an experimental procedure. Six neural network architectures with a different number of hidden neurons (K) are created, K varies from 5 to 30, with intervals of 5 neurons. For each possible architecture, 20 networks are trained to reduce the effect of randomness in the weights initialization. The network that presents the smallest test error is chosen as the representative of the architecture. Therefore, there will be a total of six neural networks representing six architectures. Lastly, their performance is compared, and the neural network that shows the smallest testing error is selected as the final network to extract the optimal parameters values.

4.4. Best Parameters Extraction

Finally, the extraction of the best combination of parameters is performed. In this way, new combinations are created: the parameter NP is taken from [10, 100] by steps of 5. The parameter F is taken from [0.1, 2] by steps of 0.05. The parameter CR is taken from [0.1, 1] by steps of 0.05. As a consequence of this process, a data-set of 14,709 combinations of NP, F and CR has been generated. Note that the ANN trained in the previous step has learned the relation of normalized inputs parameters with the output SR variable, so new combinations generated in this step need to be also normalized following Equations (16) and (17).

Subsequently, new normalized combinations of parameters are propagated through the neural network, and their respective output $\widehat{SR}$ predictions are saved. In this case, the data-set consists of 14,709 combinations of NP, F, CR and $\widehat{SR}$. These combinations have to be de-normalized following Equations (16) and (17). Finally, the combinations with the highest $\widehat{SR}$ values are extracted from the data-set, and these ones are the optimal parameters to face the problem.

5. Results and Discussion

5.1. ANN Training Results

In this section, the results obtained after applying the third step of the proposed method are summarized. Table 1 presents the training and testing performance of each neural network architecture in BBOB objective functions. The network with the smallest testing error is chosen to extract the optimal parameter values. Table 2 resumes the chosen architecture for each objective function.

From the data given in Table 1, it can be seen that the increase in hidden layer neurons (K) generally leads to a decrease in test error. However, a small network with 15 hidden neurons obtained the best performance in Fcn6 and Fcn11. We can also see that the training and testing errors in Fcn23 and Fcn24 were zero. As it is explained in Section 5.2.3, the designed DE algorithm cannot converge to the optimal point in 20 generations with the required fitness in these two functions. Therefore, the obtained SR value with all parameters is zero, and consequently, it is a straightforward mapping problem for the neural network.

Table 1. The summary of the training and testing RMSE performance of each neural network architecture in BBOB objective functions.

Function Number	Statistical Measure	Hidden Neurons K					
		5	10	15	20	25	30
Fcn 1	Training RMSE	0.0793	0.0651	0.0617	0.0573	0.0554	0.0558
	Testing RMSE	0.0758	0.0616	0.0599	0.0613	0.0578	0.0556
Fcn 2	Training RMSE	0.0433	0.0312	0.0308	0.0297	0.0258	0.0265
	Testing RMSE	0.0448	0.0328	0.0322	0.0307	0.0264	0.0295
Fcn 3	Training RMSE	0.0262	0.0210	0.0215	0.0197	0.0190	0.0192
	Testing RMSE	0.0325	0.0268	0.0238	0.0202	0.0208	0.0208
Fcn 4	Training RMSE	0.0146	0.0131	0.0132	0.0130	0.0133	0.0119
	Testing RMSE	0.0180	0.0149	0.0132	0.0134	0.0132	0.0124
Fcn 5	Training RMSE	0.0343	0.0297	0.0298	0.0213	0.0243	0.0178
	Testing RMSE	0.0383	0.0369	0.0318	0.0274	0.0261	0.0246
Fcn 6	Training RMSE	0.0412	0.0300	0.0313	0.0322	0.0317	0.0302
	Testing RMSE	0.0413	0.0333	0.0284	0.0322	0.0312	0.0290
Fcn 7	Training RMSE	0.0558	0.0511	0.0468	0.0471	0.0495	0.0450
	Testing RMSE	0.0565	0.0523	0.0469	0.0480	0.0509	0.0465
Fcn 8	Training RMSE	0.0406	0.0395	0.0359	0.0349	0.0356	0.0357
	Testing RMSE	0.0414	0.0418	0.0396	0.0379	0.0381	0.0344
Fcn 9	Training RMSE	0.0405	0.0379	0.0375	0.0362	0.0360	0.0344
	Testing RMSE	0.0407	0.0389	0.0381	0.0381	0.0365	0.0395
Fcn 10	Training RMSE	0.0315	0.0236	0.0220	0.0206	0.0198	0.0193
	Testing RMSE	0.0421	0.0304	0.0237	0.0214	0.0223	0.0207
Fcn 11	Training RMSE	0.0330	0.0219	0.0185	0.0287	0.0200	0.0220
	Testing RMSE	0.0355	0.0234	0.0223	0.0317	0.0235	0.0227
Fcn 12	Training RMSE	0.0190	0.0182	0.0164	0.0169	0.0159	0.0137
	Testing RMSE	0.0216	0.0224	0.0178	0.0171	0.0187	0.0159
Fcn 13	Training RMSE	0.0360	0.0288	0.0259	0.0232	0.0228	0.0208
	Testing RMSE	0.0381	0.0330	0.0272	0.0237	0.0233	0.0247
Fcn 14	Training RMSE	0.0116	0.0127	0.0106	0.0115	0.0102	0.0131
	Testing RMSE	0.0165	0.0133	0.0119	0.0133	0.0116	0.0134
Fcn 15	Training RMSE	0.0116	0.0127	0.0106	0.0115	0.0102	0.0131
	Testing RMSE	0.0165	0.0133	0.0119	0.0133	0.0116	0.0134

Table 1. *Cont.*

Function Number	Statistical Measure	Hidden Neurons K					
		5	10	15	20	25	30
Fcn 16	Training RMSE	0.0220	0.0229	0.0202	0.0209	0.0199	0.0199
	Testing RMSE	0.0247	0.0229	0.0221	0.0236	0.0210	0.0206
Fcn 17	Training RMSE	0.0263	0.0208	0.0248	0.0249	0.0193	0.0211
	Testing RMSE	0.0278	0.0236	0.0249	0.0253	0.0208	0.0232
Fcn 18	Training RMSE	0.0322	0.0149	0.0133	0.0127	0.0145	0.0136
	Testing RMSE	0.0332	0.0191	0.0176	0.0191	0.0193	0.0143
Fcn 19	Training RMSE	0.0485	0.0466	0.0468	0.0467	0.0463	0.0475
	Testing RMSE	0.0505	0.0481	0.0478	0.0469	0.0486	0.0476
Fcn 20	Training RMSE	0.0264	0.0265	0.0245	0.0249	0.0264	0.0254
	Testing RMSE	0.0288	0.0258	0.0271	0.0296	0.0289	0.0259
Fcn 21	Training RMSE	0.0744	0.0665	0.0656	0.0651	0.0649	0.0637
	Testing RMSE	0.0780	0.0668	0.0679	0.0654	0.0693	0.0689
Fcn 22	Training RMSE	0.0741	0.0645	0.0638	0.0622	0.0625	0.0620
	Testing RMSE	0.0757	0.0660	0.0639	0.0626	0.0636	0.0626
Fcn 23	Training RMSE	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Testing RMSE	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Fcn 24	Training RMSE	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Testing RMSE	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 2. The chosen ANN architecture for each objective function of BBOB.

Function Number	Hidden Neurons K	Function Number	Hidden Neurons K
Fcn1	30	Fcn13	25
Fcn2	25	Fcn14	30
Fcn3	20	Fcn15	25
Fcn4	30	Fcn16	30
Fcn5	30	Fcn17	25
Fcn6	15	Fcn18	30
Fcn7	30	Fcn19	20
Fcn8	30	Fcn20	30
Fcn9	25	Fcn21	20
Fcn10	30	Fcn22	30
Fcn11	15	Fcn23	5
Fcn12	30	Fcn24	5

5.2. Optimal Parameters Results

In this section, the results obtained after applying the fourth step of the proposed methodology are summarized. As stated in Section 4.4, a data-set of 14,709 combinations of NP, F, CR and $\widehat{SR}$ is obtained in each objective function. From this last data-set, the combination of parameters with the highest $\widehat{SR}$ value is extracted. In this extraction, three behaviors are observed.

5.2.1. First Case: Multiple Combinations with High Effectiveness

The DE algorithm has the ability to converge quickly in 20 generations to the optimal point with the required fitness precision of $\Delta f = 10^{-2}$. Furthermore, in this first case, the required fitness is achieved with multiple parameter combinations and high effectiveness defined as $\widehat{SR} \geq 0.95$. That means, if the DE algorithm is executed 100 times, there are several combinations of parameters that will achieve the required fitness a minimum of 95 times.

In Table 3, the objective functions with this behavior and their respective number of combinations that achieved the required fitness are summarized. From these multiple parameter combinations, any of them is competent to solve the problem modeled by the objective function.

Table 3. Objective functions from the first case and their respective number of combinations that achieved the required fitness with high effectiveness $\widehat{SR} \geq 0.95$ are summarized.

Function Number	Number of Combinations
Fcn1	4855
Fcn2	424
Fcn5	12,994
Fcn6	101
Fcn7	1137
Fcn8	171
Fcn9	99
Fcn10	131
Fcn11	103
Fcn13	69
Fcn14	2367
Fcn17	114
Fcn21	1316
Fcn22	2325

There are numerous combinations of parameters for each objective function, so it is unmanageable to show all of them in this paper analytically. For that reason, they are published as a MATLAB file in Supplementary Materials. However, in this paper, as an example, all 14,709 parameter combinations of Fcn1 and Fcn2 are graphed in Figures 4 and 5. It is helpful to illustrate the association of parameters and performance to analyze the DEs behavior in the objective function. Each point reflects a combination of NP, F and CR. The point color depends on $\widehat{SR}$ value. If the color is dark blue, that combination has a high precision; whereas if the color is dark red, that combination has a small precision. From the following two figures, it can be seen that in Fcn1 the density of blue points was higher than in Fcn2. That is logical because, as defined in Table 3, the number of combinations with high effectiveness was bigger in Fcn1 than in Fcn2.

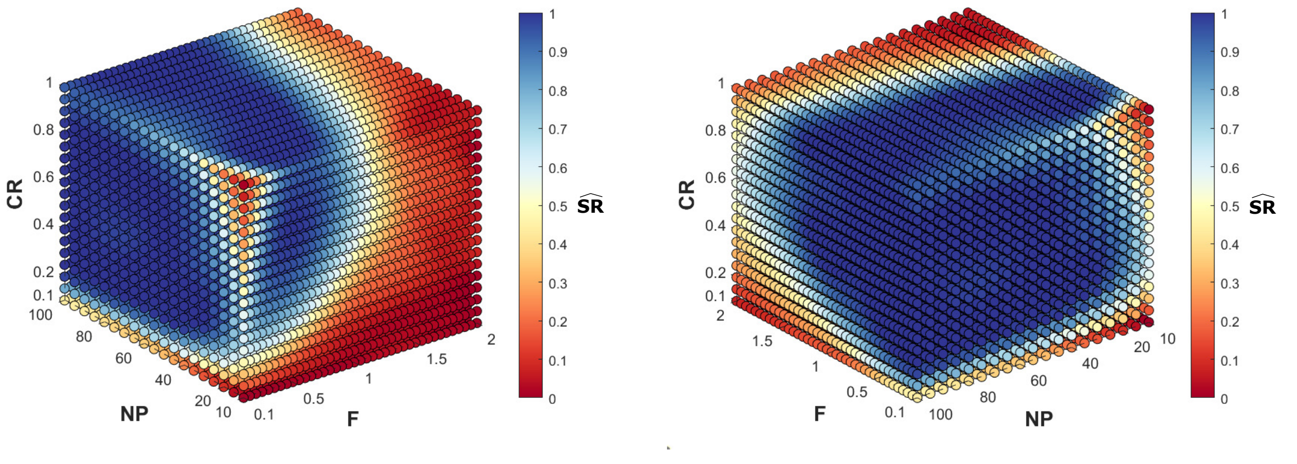
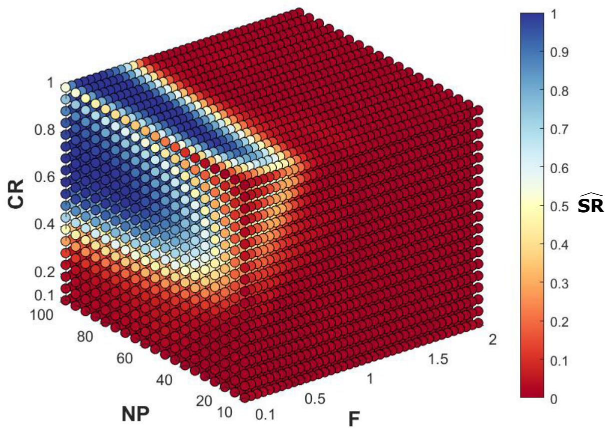
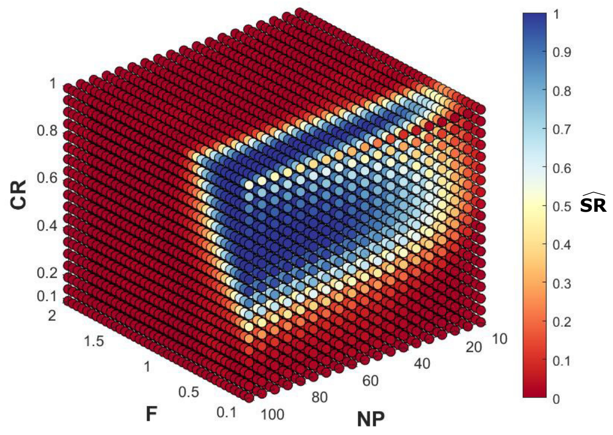


Figure 4. The illustration of 14,709 parameter combinations of Fcn1. Each point reflects a combination of NP, F and CR. The point color depends on $\widehat{SR}$ precision value. The dark blue color means a high DE precision to achieve the requirements. The dark red color means a small DE precision to achieve the requirements: (**a**) first perspective, (**b**) second perspective.



(**a**)



(**b**)

Figure 5. The illustration of 14,709 parameter combinations of Fcn2. Each point reflects a combination of NP, F and CR. The point color depends on $\widehat{SR}$ precision value. The dark blue color means a high DE precision to achieve the requirements. The dark red color means a small DE precision to achieve the requirements: (**a**) first perspective, (**b**) second perspective.

Finally, by analyzing each objective function's extracted data-set, the best ranges are defined and summarized in Table 4. It is necessary to remark that these ranges are a generalization of the extracted analytical data-set. There are also combinations of parameters out of these ranges, which achieve high performance. One can notice that the best ranges were generally represented by high NP and CR values and low F values. We can see this effect visually in Figure 5.

Table 4. The first case objective functions and their respective best parameters ranges.

Function Number	**Best Parameters Ranges**		
Fcn1	$NP > 60$	$0.65 > F > 0.10$	$CR > 0.40$
Fcn2	$NP > 75$	$0.20 > F > 0.10$	$CR > 0.75$
Fcn5	$NP > 50$	$F > 0.40$	$CR > 0.10$
Fcn6	$NP > 80$	$0.45 \geq F \geq 0.40$	$CR > 0.90$
Fcn7	$NP > 75$	$0.50 \geq F \geq 0.30$	$CR > 0.75$
Fcn8	$NP > 80$	$0.35 > F > 0.20$	$CR \geq 0.85$
Fcn9	$NP > 85$	$0.35 > F > 0.20$	$CR > 0.90$
Fcn10	$NP > 65$	$0.35 > F > 0.20$	$CR \geq 0.90$
Fcn11	$NP > 75$	$0.35 > F > 0.20$	$CR > 0.90$
Fcn13	$NP > 75$	$0.35 > F > 0.20$	$CR \geq 0.95$
Fcn14	$NP > 60$	$0.55 > F > 0.10$	$CR > 0.55$
Fcn17	$NP > 80$	$0.30 > F \geq 0.20$	$CR \geq 0.90$
Fcn21	$NP > 70$	$0.45 > F > 0.10$	$CR > 0.55$
Fcn22	$NP > 75$	$0.65 > F > 0.20$	$CR > 0.60$

5.2.2. Second Case: Medium Effectiveness

The DE algorithm can converge quickly in 20 generations to the optimal point with the required fitness precision of $\Delta f = 10^{-2}$ and medium $\widehat{SR}$ effectiveness. In this study, medium effectiveness is expressed as $\widehat{SR} < 0.95$. In this second case, the combination with the highest effectiveness value is extracted.

Table 5 shows the second case's objective functions, their best parameters combination, and the achieved $\widehat{SR}$. In all objective functions the best combination was the maximum NP, F tended to be the minimum and CR tended to the maximum. However, the $\widehat{SR}$ value was distinct in each objective function. Note that depending on the user's specific problem, the effectiveness achieved may be sufficient or not. If greater effectiveness is needed, the user should increment the number of DE generations or try other DE strategies or DE variants.

Table 5. The second case objective functions, their best parameter combination and the achieved $\widehat{SR}$ value.

Function Number	Best Parameters Combination			$\widehat{SR}$ Value
Fcn3	$NP = 100$	$F = 0.10$	$CR = 0.80$	0.77
Fcn4	$NP = 100$	$F = 0.10$	$CR = 0.80$	0.53
Fcn12	$NP = 100$	$F = 0.25$	$CR = 1.0$	0.69
Fcn15	$NP = 100$	$F = 0.10$	$CR = 1.0$	0.46
Fcn16	$NP = 100$	$F = 0.10$	$CR = 1.0$	0.46
Fcn18	$NP = 100$	$F = 0.20$	$CR = 1.0$	0.87
Fcn19	$NP = 100$	$F = 0.10$	$CR = 0.90$	0.82
Fcn20	$NP = 100$	$F = 0.10$	$CR = 0.80$	0.72

Lastly, as an example, all 14,709 parameter combinations of Fcn3 are graphed in Figure 6. It can be seen that the density of blue points was much lower compared to red ones and also to blue ones of Figures 4 and 5. Furthermore, as in the first case, the best combinations were obtained with high NP and CR values and low F values.

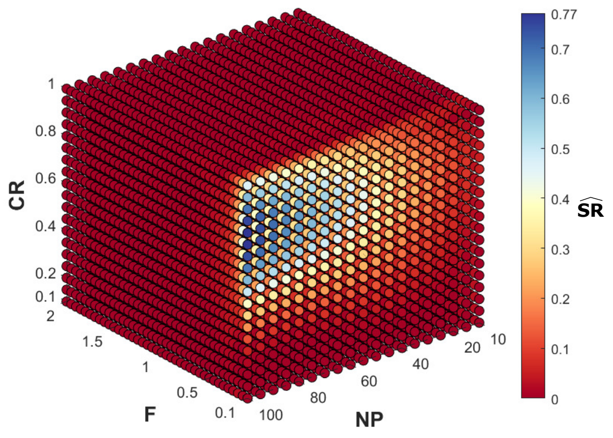


Figure 6. The illustration of 14,709 parameter combinations of Fcn3. Each point reflects a combination of NP, F and CR. The point color depends on $\widehat{SR}$ precision value. The dark blue color means a high DE precision to achieve the requirements. The dark red color means a small DE precision to achieve the requirements.

5.2.3. Third Case: Zero Effectiveness

The DE algorithm could not converge quickly in 20 generations to the optimal point with the required fitness precision of $\Delta f = 10^{-2}$. All 14,709 combinations of parameters had zero $\widehat{SR}$ value. That means there is no combination of parameters that achieved the required fitness. To observe how far real fitness is from the required one, the DE algorithm was executed t_{runs} times with a priory competent parameter combination, and the mean fitness precision was calculated (Δf_{mean}) as follows:

$$\Delta f_{mean} = \frac{\sum_{i=1}^{t_{runs}} \Delta f_i}{t_{runs}} \tag{18}$$

where Δf_i is the difference between the target value f_t and the objective function's global optimal point value f_{opt}, and t_{runs} is the total number of DE independent runs, which in this case was 40.

Table 6 shows the third case's objective functions, the used combination of parameters, and the achieved Δf_{mean}. It is observed that the mean fitness precision was much higher than the required one, so the obtained fitness value was far from the optimal one. In this circumstance, the user has two main possibilities: the definition of more flexible requisitions or the use of another DE strategy or variant.

Table 6. The third case objective functions, the used combination of parameters and the mean fitness value Δf_{mean}.

Function Number	Combination of Parameters			Δf_{mean} Value
Fcn23	$NP = 100$	$F = 0.10$	$CR = 1.00$	0.80
Fcn24	$NP = 100$	$F = 0.10$	$CR = 1.00$	1.25

5.3. Validation of the Method

In the previous section, the best combinations of parameters for each objective function have been shown. As explained, these combinations are obtained from the prediction ($\widehat{SR}$) of the ANN. In this section, these predictions defined in Supplementary Materials, Tables 5 and 6 are checked by running the DE algorithm with the extracted optimal parameters and calculating their respective SR value.

The obtained results are presented in Table 7. In this table, the used combination of parameters for each objective function, the mean value and standard deviation value of SR averaged over 10 independent runs are given. In the first case functions, from multiple optimal combinations extracted, any one is chosen. In the second and third case functions, the combinations defined in Tables 5 and 6 are chosen.

Next, a comparison with prevailing tuning rules is performed in order to verify and analyze the proposed method's performance. In the introduction section, the four most popular tuning rules are defined: Storn and Price [9], Gämperle et al. [12], Rönkkönen et al. [13] and Zielinsky et al. [14]. Following these rules, a parameter combination is designed and evaluated with ten independent DE algorithm runs for each objective function. The designed parameter combinations are as follows:

- Storn & Zielinsky: from Storn and Price rule [9], and Zielinsky et al. rule [14], a combination is designed that meets both rules. NP = 10D = 20, F = 0.7 and CR = 0.85.
- Gämperle et al. [12]: NP = 5D = 10, F = 0.6 and CR = 0.4;
- Rönkkönen et al. [13]: in separable functions NP = 30D = 60, F = 0.6 and CR = 0.1, and in non-separable functions NP = 30D = 60, F = 0.6 and CR = 0.95.

The statistical results achieved with the combinations described above are summarized in Table 8. It includes the mean and standard deviation value of SR calculated over ten DE independent runs.

Several interesting observations can be obtained from the data reported in Tables 7 and 8 and among the results presented in this paper. First, we can conclude that ANN predictions are accurate. In the first and third case functions, the $\widehat{SR}$ predictions and the calculated SR values are practically the same. In the second case functions, there is a mean difference of 2.3% between these two values. This difference is not related to ANN accuracy. Still, it is provoked by the algorithm's intrinsic stochastic component since it is tough to obtain an identical result in two DE independent runs. Even so, assuming this characteristic of the algorithm, the present results confirm that the ANN predictions are precise.

Second, the proposed method outperforms Storn and Price [9], Gämperle et al. [12], Rönkkönen et al. [13] and Zielinsky et al. [14] tuning rules. Compared to Storn & Zielinsky SR values, the achieved SR values are equal in Fcn1, Fcn5, Fcn23 and Fcn24 and significantly better in all other functions. Compared with Gämperle et al. [12] SR values, the achieved SR values are equal in Fcn23 and Fcn24 functions, similar in Fcn5 and significantly better in all other functions. Finally, in comparison with Rönkkönen et al. [13], the achieved

SR values are equal in Fcn14, Fcn22, Fcn23 and Fcn24, similar in Fcn21 and significantly superior in all other functions. Furthermore, the calculated values are equal and zero in Fcn23 and Fcn24 because all tuning rules, including our method, are used in the DE conventional algorithm under the same requirements, and the algorithm cannot converge with the required fitness precision.

Third, in this study, the typical pattern among the optimal parameters values has been a high population size (NP), low mutation factor (F) and high crossover factor (CR). Consequently, the best performance is achieved with many individuals taking small steps and maintaining high diversity. Furthermore, the statement realized in [55] is corroborated, which exclaims that in low-dimensional problems (D < 30), the population size of 100 individuals is a competent choice to solve the problem, whereas a population size lower than 50 individuals is rarely recommended.

Fourth, the biggest problem of the tuning rules defined in Table 8 is that they extrapolate the same choice law for all problems. That is, they do not adapt to the characteristics of the problem. In this case, it has been proven that this way of acting does not produce good results. Between the tuning rules, Storn & Zielinsky obtain the best performance in separable functions (Fcn1-Fcn5) and Rönkkönen et al. [13] in non-separable functions (Fcn6-Fcn24). The main reason for this is that in separable functions, Storn & Zielinsky use a significantly larger CR value than Rönkkönen et al. [13], while in non-separable functions, Rönkkönen et al. [13] employ larger NP and CR value than Storn & Zielinsky. On the other hand, the tuning rules generally obtain their best results in Fcn1, Fcn5, Fcn7, Fcn14, Fcn21 and Fcn22. Within the first case group, these last functions are where the DE algorithm has the greatest facilities to converge to the optimal point with the required fitness. In the results of our method, in Table 3, it is proven that precisely in these functions, the number of combinations that achieved the required fitness is the highest. The most significant advantage of these tuning rules is that it is unnecessary to design or code new approaches.

Finally, the proposed method can be extrapolated to different problems and that is mostly the differentiating key factor compared to other tuning rules. In the DE algorithm, there is a strong interaction between the parameters NP, F and CR, the DE strategy, the objective function to optimize, the global point search domain and the requisitions such as maximum generations and required fitness. In the designed method, once the objective function is designed, the DE strategy is chosen, and the requirements are defined, the values of the optimal parameters are calculated adjusting to these characteristics. In summary, this allows the user not to waste time thinking about what parameter values are competent enough to solve his problem or avoid using the tedious trial-and-error approach.

Table 7. Experimental results by DE algorithm for 24 BBOB objective functions with their respective parameter values. The mean (SR mean) and standard deviation (SR std) values of SR are calculated over 10 DE algorithm independent runs.

Function Number	Combination of Parameters			SR Mean	SR Std
Fcn1	$NP = 60$	$F = 0.40$	$CR = 0.45$	1.0000	0.0000
Fcn2	$NP = 80$	$F = 0.15$	$CR = 0.80$	0.9975	0.0079
Fcn3	$NP = 100$	$F = 0.10$	$CR = 0.80$	0.7520	0.0691
Fcn4	$NP = 100$	$F = 0.10$	$CR = 0.80$	0.5075	0.0736
Fcn5	$NP = 55$	$F = 1.65$	$CR = 0.40$	1.0000	0.0000
Fcn6	$NP = 85$	$F = 0.40$	$CR = 0.95$	0.9800	0.0284
Fcn7	$NP = 80$	$F = 0.30$	$CR = 0.85$	1.0000	0.0000
Fcn8	$NP = 85$	$F = 0.25$	$CR = 0.90$	0.9975	0.0079
Fcn9	$NP = 90$	$F = 0.30$	$CR = 0.95$	0.9800	0.0230
Fcn10	$NP = 75$	$F = 0.25$	$CR = 0.90$	0.9875	0.0212
Fcn11	$NP = 90$	$F = 0.25$	$CR = 0.95$	1.0000	0.0000
Fcn12	$NP = 100$	$F = 0.25$	$CR = 1.00$	0.7075	0.0898
Fcn13	$NP = 85$	$F = 0.25$	$CR = 0.95$	0.9950	0.0105
Fcn14	$NP = 65$	$F = 0.45$	$CR = 0.70$	1.0000	0.0000

Table 7. *Cont.*

Function Number	Combination of Parameters			SR Mean	SR Std
Fcn15	$NP = 100$	$F = 0.10$	$CR = 1.00$	0.4300	0.0832
Fcn16	$NP = 100$	$F = 0.10$	$CR = 1.00$	0.4500	0.0870
Fcn17	$NP = 90$	$F = 0.20$	$CR = 0.90$	0.9950	0.0158
Fcn18	$NP = 100$	$F = 0.20$	$CR = 1.00$	0.9175	0.0528
Fcn19	$NP = 100$	$F = 0.10$	$CR = 0.90$	0.7900	0.0444
Fcn20	$NP = 80$	$F = 0.10$	$CR = 0.80$	0.7400	0.0876
Fcn21	$NP = 85$	$F = 0.30$	$CR = 0.70$	1.0000	0.0000
Fcn22	$NP = 100$	$F = 0.25$	$CR = 0.75$	1.0000	0.0000
Fcn23	$NP = 100$	$F = 0.10$	$CR = 1.00$	0.0000	0.0000
Fcn24	$NP = 100$	$F = 0.10$	$CR = 1.00$	0.0000	0.0000

Table 8. Experimental results by DE algorithm for 24 BBOB objective functions following four distinct parameter tuning rules: Storn and Price, Gämperle et al., Rönkkönen et al. and Zielinsky et al. From Storn and Price rule, and Zielinsky et al. rule, a combination called Storn & Zielinsky is designed that meets both rules. The mean (SR mean) and standard deviation (SR std) values of SR are calculated over 10 DE algorithm independent runs.

Function Number	Storn & Zielinsky		Gämperle		Rönkkönen	
	SR Mean	SR Std	SR Mean	SR Std	SR Mean	SR Std
Fcn1	1.0000	0.0000	0.7025	0.0606	0.1900	0.0648
Fcn2	0.0500	0.0264	0.0000	0.0000	0.0025	0.0079
Fcn3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Fcn4	0.0050	0.0105	0.0000	0.0000	0.0000	0.0000
Fcn5	1.0000	0.0000	0.9825	0.0265	1.0000	0.0000
Fcn6	0.0775	0.0322	0.0000	0.0000	0.7075	0.0635
Fcn7	0.5975	0.0862	0.0525	0.0381	1.0000	0.0000
Fcn8	0.0725	0.0362	0.0050	0.0158	0.4175	0.0717
Fcn9	0.1125	0.0710	0.0100	0.0175	0.4925	0.0528
Fcn10	0.0025	0.0079	0.0000	0.0000	0.1700	0.0537
Fcn11	0.0025	0.0079	0.0000	0.0000	0.1675	0.0501
Fcn12	0.0000	0.0000	0.0000	0.0000	0.0200	0.0197
Fcn13	0.0025	0.0079	0.0000	0.0000	0.1325	0.0457
Fcn14	0.9625	0.0358	0.1275	0.0506	1.0000	0.0000
Fcn15	0.0025	0.0079	0.0000	0.0000	0.0075	0.0169
Fcn16	0.0025	0.0079	0.0025	0.0079	0.0275	0.0275
Fcn17	0.0125	0.0177	0.0000	0.0000	0.1175	0.0541
Fcn18	0.0000	0.0000	0.0000	0.0000	0.0025	0.0079
Fcn19	0.0875	0.0489	0.0325	0.0313	0.3125	0.0922
Fcn20	0.0075	0.0121	0.0025	0.0079	0.0500	0.0204
Fcn21	0.5975	0.0759	0.1850	0.0603	0.9825	0.0169
Fcn22	0.8575	0.0678	0.2525	0.0606	1.0000	0.0000
Fcn23	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Fcn24	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

6. Conclusions

The present study focuses on exposing a method to tune the optimal parameters of the DE algorithm. In the DE algorithm, there is a strong interaction between the parameters NP, F and CR, the DE strategy, the objective function to optimize, the global point search domain and the requisitions such as maximum generations and required fitness. It is in the interplay of these variables where the proposed method is inserted.

The core of the method is using an ANN-MLP to predict the optimal parameter values depending on the objective function, the DE strategy and the user-defined requirements.

Subject to the DE algorithm's ability to solve the problem, three different cases may arise with applying this method. For each case, specifications about the procedure to follow are given in detail.

The final results show that ANNs prediction effectiveness is more accurate in the first and third case than in the second case objective functions. This difference is caused because in the second case objective functions, the stochastic component effect of DE is more significant than in the other two cases. Even so, taking into account this stochastic characteristic of DE, the present results confirm that these predictions are considerably accurate.

Finally, a comparison with four prevailing tuning rules is performed, and it is concluded that the proposed method outperforms these rules. Future research could examine this method's application in high-dimensional problems with different user-defined requirements and DE strategies. Our aim is to apply the proposed method to a robotic manipulator axis interpolation with auto-guided vehicle movement in future work.

Supplementary Materials: The following are available online at https://www.mdpi.com/2227-7390/9/4/427/s1. The optimal parameter combinations of the first case objective functions have been added. For each function a MATLAB file (.mat) is defined. In total, there are 14 files.

Author Contributions: Conceptualization, M.C.-T., E.Z. and D.T.-F.-B.; Data curation, U.F.-G.; Formal analysis, M.C.-T. and E.Z.; Project administration, E.Z. and U.F.-G.; Resources, D.T.-F.-B. and A.T.-F.-B.; Software, M.C.-T.; Supervision, U.F.-G. All authors have read and agreed to the published version of the manuscript.

Funding: The authors appreciate the support to the government of the Basque Country through research programs Grants N. ELKARTEK 20/71 and ELKARTEK: KK-2019/00099.

Informed Consent Statement: Not applicable.

Data Availability Statement: The authors have added 14 MATLAB files. In each file, the optimal parameter combinations of the first case objective functions are defined.

Acknowledgments: The authors are grateful for the support provided by SGIker of UPV/EHU. This research has been developed under the frame of the Joint Research Laboratory on Offshore Renewable Energy (JRL-ORE).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Back, T.; Fogel, D.B.; Michalewicz, Z. *Handbook of Evolutionary Computation*, 1st ed.; IOP Publishing Ltd.: Bristol, UK, 1997.
2. Das, S.; Suganthan, P.N. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Trans. Evol. Comput.* **2011**, *15*, 4–31. [CrossRef]
3. Abderazek, H.; Ferhat, D.; Ivana, A. Adaptive mixed differential evolution algorithm for bi-objective tooth profile spur gear optimization. *Int. J. Adv. Manuf. Technol.* **2017**, *90*, 2063–2073. [CrossRef]
4. Biswas, P.; Suganthan, P.; Wu, G.; Amaratunga, G. Parameter estimation of solar cells using datasheet information with the application of an adaptive differential evolution algorithm. *Renew. Energy* **2019**, *132*, 425–438. [CrossRef]
5. Zamuda, A.; Sosa, J.D.H. Success history applied to expert system for underwater glider path planning using differential evolution. *Expert Syst. Appl.* **2019**, *119*, 155–170. [CrossRef]
6. Zulueta, E.; Kurt, E.; Uzun, Y.; Lopez-Guede, J.M. Power control optimization of a new contactless piezoelectric harvester. *Int. J. Hydrogen Energy* **2017**, *42*, 18134–18144. [CrossRef]
7. Aramendia, I.; Saenz-Aguirre, A.; Boyano, A.; Fernandez-Gamiz, U.; Zulueta, E. Oscillating U-Shaped Body for Underwater Piezoelectric Energy Harvester Power Optimization. *Micromachines* **2019**, *10*, 737. [CrossRef]
8. Storn, R.; Price, K. DE-a simple and efficient adaptive scheme for global optimization over continuous space. *Tech. Rep.* **1995**, *25*, 95–102.
9. Storn, R. On the usage of differential evolution for function optimization. In Proceedings of the North American Fuzzy Information Processing, Berkeley, CA, USA, 19–22 June 1996; pp. 519–523. [CrossRef]
10. Mohamed, A.W.; Mohamed, A.K. Adaptive guided differential evolution algorithm with novel mutation for numerical optimization. *Int. J. Mach. Learn. Cybern.* **2019**, *10*, 253–277. [CrossRef]
11. Tanabe, R.; Fukunaga, A.S. Improving the search performance of SHADE using linear population size reduction. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, 6–11 July 2014; pp. 1658–1665.
12. Gämperle, R.; Müller, S.D.; Koumoutsakos, P. A parameter study for differential evolution. In *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*; WSEAS Press: Zurich, Switzerland, 2002; Volume 10, pp. 293–298.

13. Ronkkonen, J.; Kukkonen, S.; Price, K. *Real-Parameter Optimization with Differential Evolution*; IEEE: New York, NY, USA, 2005; p. 513.
14. Zielinski, K.; Weitkemper, P.; Laur, R.; Kammeyer, K.D. Parameter study for differential evolution using a power allocation problem including interference cancellation. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 1857–1864.
15. Sarker, R.A.; Elsayed, S.M.; Ray, T. Differential Evolution With Dynamic Parameters Selection for Optimization Problems. *IEEE Trans. Evol. Comput.* **2014**, *18*, 689–707. [CrossRef]
16. Al-Dabbagh, R.D.; Neri, F.; Idris, N.; Baba, M.S. Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy. *Swarm Evol. Comput.* **2018**, *43*, 284–311. [CrossRef]
17. Brest, J.; Greiner, S.; Boskovic, B.; Mernik, M.; Zumer, V. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Trans. Evol. Comput.* **2006**, *10*, 646–657. [CrossRef]
18. Islam, S.M.; Das, S.; Ghosh, S.; Roy, S.; Suganthan, P.N. An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization. *IEEE Trans. Syst. Man Cybern. Part B-Cybern.* **2012**, *42*, 482–500. [CrossRef]
19. Liu, J.; Lampinen, J. A fuzzy adaptive differential evolution algorithm. *Soft Comput.* **2005**, *9*, 448–462. [CrossRef]
20. Ochoa, P.; Castillo, O.; Soria, J. Differential Evolution Using Fuzzy Logic and a Comparative Study with Other Metaheuristics. *Nat.-Inspired Des. Hybrid Intell. Syst.* **2017**, *667*, 257–268. [CrossRef]
21. Tsafarakis, S.; Zervoudakis, K.; Andronikidis, A.; Altsitsiadis, E. Fuzzy self-tuning differential evolution for optimal product line design. *Eur. J. Oper. Res.* **2020**, *287*, 1161–1169. [CrossRef]
22. Zhang, J.; Sanderson, A.C. JADE: Adaptive differential evolution with optional external archive. *IEEE Trans. Evol. Comput.* **2009**, *13*, 945–958. [CrossRef]
23. Zhou, Y.Z.; Yi, W.C.; Gao, L.; Li, X.Y. Adaptive Differential Evolution with Sorting Crossover Rate for Continuous Optimization Problems. *IEEE Trans. Cybern.* **2017**, *47*, 2742–2753. [CrossRef] [PubMed]
24. Tanabe, R.; Fukunaga, A. Success-History Based Parameter Adaptation for Differential Evolution. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; p. 78.
25. Piotrowski, A.P.; Napiorkowski, J.J. Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure? *Swarm Evol. Comput.* **2018**, *43*, 88–108. [CrossRef]
26. Qin, A.K.; Huang, V.L.; Suganthan, P.N. Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. *IEEE Trans. Evol. Comput.* **2009**, *13*, 398–417. [CrossRef]
27. Fan, Q.; Wang, W.; Yan, X. Differential evolution algorithm with strategy adaptation and knowledge-based control parameters. *Artif. Intell. Rev.* **2019**, *51*, 219–253. [CrossRef]
28. Mohamed, A.W.; Suganthan, P.N. Real-parameter unconstrained optimization based on enhanced fitness-adaptive differential evolution algorithm with novel mutation. *Soft Comput.* **2018**, *22*, 3215–3235. [CrossRef]
29. Wang, Y.; Cai, Z.; Zhang, Q. Differential Evolution with Composite Trial Vector Generation Strategies and Control Parameters. *IEEE Trans. Evol. Comput.* **2011**, *15*, 55–66. [CrossRef]
30. Mallipeddi, R.; Suganthan, P.N.; Pan, Q.K.; Tasgetiren, M.F. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl. Soft Comput.* **2011**, *11*, 1679–1696. [CrossRef]
31. Wu, G.; Shen, X.; Li, H.; Chen, H.; Lin, A.; Suganthan, P.N. Ensemble of differential evolution variants. *Inf. Sci.* **2018**, *423*, 172–186. [CrossRef]
32. Wu, G.; Mallipeddi, R.; Suganthan, P.N.; Wang, R.; Chen, H. Differential evolution with multi-population based ensemble of mutation strategies. *Inf. Sci.* **2016**, *329*, 329–345. [CrossRef]
33. Sengupta, S.; Basak, S.; Saikia, P.; Paul, S.; Tsalavoutis, V.; Atiah, F.; Ravi, V.; Peters, A. A review of deep learning with special emphasis on architectures, applications and recent trends. *Knowl.-Based Syst.* **2020**, *194*, 105596. [CrossRef]
34. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **2020**, *37*, 362–386. [CrossRef]
35. Teso-Fz-Betoño, D.; Zulueta, E.; Sánchez-Chica, A.; Fernandez-Gamiz, U.; Saenz-Aguirre, A. Semantic Segmentation to Develop an Indoor Navigation System for an Autonomous Mobile Robot. *Mathematics* **2020**, *8*, 855. [CrossRef]
36. Lopez-Guede, J.; Ramos, J.; Zulueta, E.; Fernandez-Gamiz, U.; Oterino, F. Systematic modeling of photovoltaic modules based on artificial neural networks. *Int. J. Hydrogen Energy* **2016**, *41*. [CrossRef]
37. Saenz-Aguirre, A.; Zulueta, E.; Fernandez-Gamiz, U.; Ulazia, A.; Teso-Fz-Betoño, D. Performance enhancement of the artificial neural network–based reinforcement learning for wind turbine yaw control. *Wind Energy* **2019**, *23*. [CrossRef]
38. Sezer, O.B.; Gudelek, M.U.; Ozbayoglu, A.M. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Appl. Soft Comput.* **2020**, *90*, 106181. [CrossRef]
39. Abiodun, O.I.; Jantan, A.; Omolara, A.E.; Dada, K.V.; Mohamed, N.A.; Arshad, H. State-of-the-art in artificial neural network applications: A survey. *Heliyon* **2018**, *4*, e00938. [CrossRef]
40. Hansen, N.; Finck, S.; Ros, R.; Auger, A. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA. 2009. Available online: https://hal.inria.fr/inria-00362633v2/file/RR-6829v2.pdf (accessed on 20 January 2021).

41. Tong, L.; Dong, M.; Jing, C. An improved multi-population ensemble differential evolution. *Neurocomputing* **2018**, *290*, 130–147. [CrossRef]
42. Caraffini, F.; Kononova, A.; Corne, D. Infeasibility and structural bias in Differential Evolution. *Inf. Sci.* **2019**, *496*, 161–179. [CrossRef]
43. Stanovov, V.; Akhmedova, S.; Semenkin, E. Selective Pressure Strategy in differential evolution: Exploitation improvement in solving global optimization problems. *Swarm Evol. Comput.* **2019**, *50*, 100463. [CrossRef]
44. Brest, J.; Maucec, M.S.; Boskovic, B. *Single Objective Real-Parameter Optimization: Algorithm jSO*; IEEE: New York, NY, USA, 2017; p. 1318.
45. Deng, W.; Xu, J.; Song, Y.; Zhao, H. Differential evolution algorithm with wavelet basis function and optimal mutation strategy for complex optimization problem. *Appl. Soft Comput.* **2020**, 106724. [CrossRef]
46. Peng, H.; Guo, Z.; Deng, C.; Wu, Z. Enhancing differential evolution with random neighbors based strategy. *J. Comput. Sci.* **2018**, *26*, 501–511. [CrossRef]
47. Meng, Z.; Pan, J.S.; Tseng, K.K. PaDE: An enhanced Differential Evolution algorithm with novel control parameter adaptation schemes for numerical optimization. *Knowl.-Based Syst.* **2019**, *168*, 80–99. [CrossRef]
48. Li, X.; Wang, L.; Jiang, Q.; Li, N. Differential evolution algorithm with multi-population cooperation and multi-strategy integration. *Neurocomputing* **2021**, *421*, 285–302. [CrossRef]
49. Zhao, S.Z.; Suganthan, P.N. Empirical investigations into the exponential crossover of differential evolutions. *Swarm Evol. Comput.* **2013**, *9*, 27–36. [CrossRef]
50. Qiu, X.; Tan, K.C.; Xu, J. Multiple Exponential Recombination for Differential Evolution. *IEEE Trans. Cybern.* **2017**, *47*, 995–1006. [CrossRef]
51. Guo, S.; Yang, C. Enhancing Differential Evolution Utilizing Eigenvector-Based Crossover Operator. *IEEE Trans. Evol. Comput.* **2015**, *19*, 31–49. [CrossRef]
52. Nwankpa, C.; Ijomah, W.; Gachagan, A.; Marshall, S. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *arXiv* **2018**, arXiv:1811.03378.
53. Pavelka, A.; Procházka, A. Algorithms for initialization of neural network weights. In Proceedings of the Conference Technical Computing, 4 November 2004; pp. 453–459. Available online: https://www2.humusoft.cz/www/papers/tcp04/pavelka.pdf (accessed on 4 February 2021).
54. Hagan, M.T.; Menhaj, M.B. Training feedforward networks with the Marquardt algorithm. *IEEE Trans. Neural Netw.* **1994**, *5*, 989–993. [CrossRef]
55. Piotrowski, A.P. Review of Differential Evolution population size. *Swarm Evol. Comput.* **2017**, *32*, 1–24. [CrossRef]