Evolutionary Higher-Order Extreme Learning Machine

Vasileios Christou[a,*], Alexandros T. Tzallas[a], Markos G. Tsipouras[b], Georgios Tsoumanis[a], Nikolaos Giannakeas[a]

[a] *Department of Informatics and Telecommunications, University of Ioannina, Arta, Greece*
[b] *Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece*

[*] Corresponding Author.

Email addresses: bchristou1@gmail.com (Vasileios Christou), tzallas@uoi.gr (Alexandros T. Tzallas), mtsipouras@uowm.gr (Markos G. Tsipouras), gtsoum@uoi.gr (Georgios Tsoumanis), giannakeas@uoi.gr (Nikolaos Giannakeas)

Abstract

The low-order unit type is the most commonly used neuron architecture, where each input is multiplied by a corresponding weight. These types are restricted to linear separable problems, and overcoming this restriction would require more advanced units. Higher-order units like the multi-cube neuron treat the input vector as a set of multi-dimensional hyper-cubes where each cube's site corresponds to a weight. Multi-cube unit (MCU) single-layer neural networks (SLNNs) can be trained with the extreme learning machine (ELM) algorithm, which has a very fast training speed because it does not use an iterative process like gradient-based methods. An SLNN is trained by randomizing the hidden layer weights and thresholds, and then with the help of the Moore-Penrose pseudo-inverse, it can analytically calculate the output layer(s) weights. The proposed evolutionary higher-order ELM (EHO-ELM) algorithm utilizes a modified self-adaptive genetic algorithm (GA) to create an SLNN containing MCUs in its hidden and output layers. EHO-ELM can automatically determine the optimal number and structure of cubic sub-units for each MCU of the neural network. Also, it can automatically tune the hidden layer weights and thresholds to increase the constructed network's generalization ability. This paper's experimental work section compares EHO-ELM in 18 datasets with 14 existing machine-learning methods. The compared approaches include ten gradient-based methods, support vector machine (SVM), and three ELM-based methods. The experimental results revealed that the proposed method had the best generalization performance. The significance of these results was verified using the Wilcoxon sign-rank test.

Keywords:
extreme learning machine, higher-order neuron, genetic algorithm,
multi-cube neuron, single-layer neural network

1. Introduction

The extreme learning machine (ELM) method created by Huang et al. (2004, 2006b) is a training algorithm for single-layer neural networks (SLNNs). Its main advantage is very fast training speed since it can train an SLNN in a two-step manner. Initially, it randomizes the hidden layer parameters (weights and thresholds), and then, with the help of the Moore-Penrose pseudo-inverse, it analytically calculates the network's output neuron(s) weights. Other advantages include simplicity and efficiency since it doesn't require user-defined parameters like the learning rate in gradient-based methods, and local minima do not affect its performance. Besides the advantages mentioned above, ELM has some disadvantages, one of them being the random initialization of the hidden layer parameters, which can affect its generalization performance (Bhat et al., 2008; Javed et al., 2014).

The earlier work by Christou et al. (2018) utilizing the ELM algorithm for training networks containing the higher-order (cubic) units proposed by Gurney (1989) showed that the use of these neuron types could improve the generalization ability of SLNNs. Cubic neurons treat the input vector as a multi-dimensional hypercube where each site corresponds to a weight. In these units, the number of weights rises exponentially according to the number of inputs (an n input vector will contain 2^n weights), causing performance issues. The solution to this problem was the creation of the multi-cube neuron, where a larger hypercube is replaced with a set of sub-cubes having smaller dimensions. The optimal number and dimensions of these sub-cubes are open problems and can affect the performance of multi-cube neural networks.

The motivation behind creating the proposed evolutionary higher-order ELM (EHO-ELM) algorithm was to create an optimal multi-cube SLNN for each regression or classification problem. The hidden layer weights, the number of sub-cubes in each neuron, and the dimension of each sub-cube are automatically determined without any intervention from the user. EHO-ELM utilizes a genetic algorithm (GA) that creates and evolves a series of multi-cube neural networks. The multi-cube (MCU) SLNNs contain MCUs in both layers. At the end of the evolution process, the best one is selected as the most optimal according to a fitness criterion. The fitness criterion was the mean square error (MSE) for regression problems and the accuracy (acc) for classification problems.

EHO-ELM is self-adaptive and doesn't require any parameter tuning from the user's perspective. The evolution begins by creating the initial population containing a set of randomly created MCU networks. Then, the selection operator trains the population using the ELM algorithm and selects the best ones for the reproduction process using a fitness criterion. The reproduction process uses a custom operator based on the uniform crossover (Syswerda et al., 1989), which adapts the produced offspring for creating valid offspring. A percentage of the reproduced population varying from 10%, 30% to a maximum of 50% is mutated by randomly creating the weights from one of its sub-cubes. The mutation operation is adaptive and gradually increases if the reproduction process at the previous generation has not created a better SLNN. The evolution process continues until no better SLNN is found for five iterations (generations).

The paper is structured into seven main sections starting from the "Introduction", which describes the problem and the proposed EHO-ELM algorithm. It also explains the motivation behind its creation. The "Introduction' is followed by the "Literature review" and "Related work" sections, where the former explores and compares existing works with EHO-ELM. The latter contains five sub-sections explaining the structure of cubic and MCU units, the ELM algorithm, its extension to MCU SLNNs, and the structure of a typical GA. Then, the following section describes the EHO-ELM algorithm. The fifth section presents the experimental results from comparing EHO-ELM with 14 machine-learning methods in 18 datasets. The comparison is supplemented with the Wilcoxon signed-rank test to verify the results' significance. The article finishes with the "Discussion" and "Conclusion" sections.

2. Literature review

Many ELM variants are focused on improving the generalization performance of the algorithm by tuning the hidden layer weights and thresholds. The evolutionary ELM by Zhu et al. (2005) utilizes a differential evolutionary algorithm for this task. Similarly, the self-adaptive evolutionary ELM by Cao et al. (2012) uses a self-adaptive variant of the same algorithm. Differential evolution is a GA variant for minimizing real-valued multi-modal objective functions (Storn, 1996; Storn & Price, 1997). Alexander & Annamalai (2016) utilized a GA with a novel elitism optimization method, speeding up the evolution process and avoiding local optimums. Han et al. (2013) created the improved particle swarm optimization ELM method, which uses an improved version of the particle swarm optimization algorithm, an artificial life-inspired algorithm closely related to evolutionary computation (Eberhart & Kennedy, 1995), to optimize the hidden nodes' parameters. The optimization process considers the root mean squared error (RMSE) on the validation set and the norm of the output weights. Rathod & Wankhade (2022) combined cuckoo search and invasive weed optimization algorithms with ELM. Cuckoo search (Yang & Deb, 2009) is based on the breeding behavior of cuckoos, while invasive weed optimization (Karimkashi & Kishk, 2010) is based on the colonization phenomenon of invasive weeds. The integration of cuckoo search in invasive weed optimization is done to improve its convergence. Zhang et al. (2016) combined a memetic algorithm with ELM. The memetic algorithm (Moscato et al., 1989) is a hybrid GA that contains an individual learning procedure to perform local refinements. Yang et al. (2022) used the multi-strategy whale optimization algorithm to select the hidden layer parameters of semi-supervised ELM (SS-ELM). The whale optimization algorithm is nature-inspired and simulates the whale behavior during hunting (Mirjalili & Lewis, 2016) while SS-ELM is a variant for semi-supervised tasks (Huang et al., 2014). Cai et al. (2019) utilized an improved version of the grey wolf optimization method to select the kernel ELM (KELM) algorithm parameters. KELM is a method that integrates the kernel function into ELM Huang et al. (2011). The improvement over the traditional grey wolf optimizer, inspired by grey wolves' leadership hierarchy and hunting habits (Mirjalili et al., 2014), involved creating a new hierarchical mechanism for improving its search capabilities. Similarly, Heidari et al. (2019) hybridized another improved version of the grey wolf optimization algorithm with KELM for tuning the latter's parameters. This improved version included adopting exploratory and exploitative techniques for enhancing the search capabilities of the original grey wolf optimizer. Faris et al. (2020) utilized the salp swarm algorithm, which imitates salps' swarming behavior in oceans when navigating and searching for food, to select the hidden layer parameters (Mirjalili et al., 2017). Mengcan et al. (2021) created the constrained voting ELM algorithm for classification problems which calculates the hidden layer weights and thresholds by combining the differences of between-class samples. A voting selection scheme is incorporated to improve its accuracy. Yu et al. (2021) developed an improved version of the grasshopper optimization algorithm, which utilizes the simulated annealing method to enhance its performance. This improved version was applied to the KELM algorithm to optimize its parameters. The grasshopper optimizer Mirjalili et al. (2018) is nature-inspired by the navigation of grasshopper swarms. Cao et al. (2021) hybridized the crow search algorithm with particle swarm optimization to enhance the latter's global search capability. This improved crow search algorithm version was used to estimate the hidden layer weights and thresholds of ELM. Crow search is a method based on the observation that crows store their food sur-plus in hidden areas and retrieve it when they need it Askarzadeh (2016). The above methods focus on tuning the hidden layer parameters of ELM without considering higher-order units.

Existing ELM-based works regarding the higher-order units by Gurney (1989) include the homogeneous hybrid ELM (Christou et al., 2018). The authors divided the neuron into three structural units, namely (dendrite, activation function, and activation-output function). Then, they created a series of custom neuron types with different sub-units used in the hidden layer of SLNNs trained with ELM. Each hidden layer utilized the same custom neuron type. The experiments included low-order and cubic units. The paper revealed that alternative neuron types got better generalization performance than the traditional low-order unit type with the sigmoid transfer function. An extension of this method was the heterogeneous hybrid ELM (Christou et al., 2019), which works with low-order and multi-cube units. The authors used a GA to evolve homogeneous (SLNNs with the same neuron types in the hidden layer) networks into heterogeneous ones (SLNNs with different combinations of hidden units). A self-adaptive version of this algorithm was created by Christou et al. (2020) and was hybridized with the linear-regression method to improve the accuracy of a low-cost temperature sensor (Christou et al., 2022). Although the above methods can work with higher-order neurons, they cannot tune the hidden-layer parameters. Moreover, they are restricted to SLNNs with low-order units in their output layer(s). The heterogeneous variants of the above algorithms work with multi-cube units having a fixed sub-cube structure. Still, they cannot find an optimal combination of sub-cube units for the multi-cube neurons of the created SLNNs.

The self-adaptive version of the heterogeneous hybrid ELM shows some similarities with the proposed EHO-ELM method since both methods are self-adaptive and can create networks with different neuron type combinations in their hidden layer. Existing methods to create ELM-based networks with heterogeneous structures include the optimally pruned ELM (OP-ELM) by Miche et al. (2009), which follows a three-stage approach. Initially, it creates a large hidden layer containing different neuron types. Then, the significance of each neuron is ranked using the multi-response sparse-regression (MRSR) algorithm. MRSR utilizes a linearly parameterized model for selecting the forward regressors in calculating a multi-variate target (Similä & Tikka, 2005). The third step involves selecting the neurons forming the final SLNN using leave-one-out cross-validation. The Tikhonov-regularized OP-

ELM improves the original OP-ELM by utilizing two regularization penalties. The first is the L_1 penalty for ranking the hidden layer neurons, while the second is the L_2 penalty on the regression weights (Miche et al., 2011). These methods can create heterogeneous SLNNs but do not consider higher-order units.

Other higher-order neuron types include the sigma-pi neurons by Rumelhart et al. (1986a), which are similar to the conjunctive units by Feldman & Ballard (1982). The input is a weighted product sum from individual inputs in these neurons. The sigma-pi neurons have the problem of exponentially increasing weights according to the number of inputs. The pi-sigma units (Shin & Ghosh, 1991) were created to circumvent this problem. A pi-sigma network uses product cells as the output units for adopting the capabilities of higher-order units while using fewer weights and processing units. The multi-cube unit has the advantage over the pi-sigma neuron in that a user can have the option to make groups of inputs and assign them a sub-cube unit. This way can increase or reduce the number of weights used for specific inputs and allows more control over the size of the weight vector.

Many ELM variants are used to solve specific problem types. Shariati et al. (2022) hybridized the grey wolf optimizer with ELM for tuning the latter's parameters. The resulting method was applied for predicting hardened concrete compressive strength in cases where cement was partially replaced with alternative pozzolans. Albadr et al. (2022) used particle swarm optimization to optimize the hidden layer weights and thresholds of ELM. The resulting method was used as a classifier for a COVID-19 detection method using voice data from the respiratory system. Wang et al. (2022) tuned ELM's parameters using a GA and used this method to predict future color trends. Dong et al. (2020) hybridized the bat algorithm with ELM for calculating the daily dew point temperature. The bat algorithm is nature-inspired by the echolocation ability of bats (Yang, 2010). Dou et al. (2022) proposed a hybrid approach for assessing the charging state of lithium-ion batteries. The proposed method created an improved version of the salp swarm algorithm for tuning the hidden layer parameters of ELM. This version incorporated chaotic mapping to distribute the initial individuals uniformly and the sine cosine algorithm to improve the salp swarm algorithm's update formula. These methods focus on tuning the hidden layer parameters of ELM without considering higher-order units.

The proposed EHO-ELM algorithm was compared with 14 machine learning methods in total. These methods included three ELM variants and the support vector machine (SVM) algorithm. Besides traditional ELM, the experimental part included online sequential ELM (OS-ELM) and multi-cube unit ELM (MCU-ELM). OS-ELM is an extension to the original ELM, enabling it to work with sequential data (Huang et al., 2005) while MCU-ELM is an ELM extension for SLNNs with MCUs in both their layers Christou (2023). The SVM algorithm creates a maximum margin hyperplane between the data points of the two classes in a binary classification problem which is then used to classify unknown data. In regression problems, support vector regression (SVR) is used instead, an adaptation of SVM for continuous values (Zahir & Mahdi, 2015).

EHO-ELM was compared with ten back-propagation (BP) variants. The original idea behind the BP algorithm was developed by Werbos (1974). Still, it was widely accepted a few years later by the works of Rumelhart et al. (1986b) in their entitled paper "Learning representations by back-propagating errors" (Rumelhart et al., 1995). The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is a second-order quasi-Newton approach for numerical optimization which uses the second-order derivative (Hessian matrix) approximation update formula by Broyden (1970a,b), Fletcher (1970), Goldfarb (1970), and Shanno (1970). The Powell-Beale conjugate gradient (PBCG) utilizes the conjugate gradient (CG) method, which minimizes multi-variate functions. This BP variant has a linear convergence rate which can be avoided by periodically restarting its iterative training procedure. The restarting process is done automatically by considering the objective function (Powell, 1977). The Fletcher-Reeves (FRCG) (Fletcher & Reeves, 1964; Scales, 1985) and Polak-Ribiére (PRCG) (Polak & Ribiere, 1969) CG algorithms are generalizations of the CG method for non-linear optimization problems using different conjugate direction update formulas. Scaled conjugate gradient (SCG) (Møller, 1993) utilizes a step size scaling mechanism to speed up the training process. The steepest descent (SD)(Cauchy et al., 1847; Meza, 2010) is one of the oldest and simplest methods for minimizing a non-linear function but faces slow convergence issues. Gradient descent with momentum (GDM) (Polyak, 1964) adds momentum to the gradient descent (GD) optimization process for overcoming noisy gradient oscillations. The momentum allows GD to converge faster. GD with momentum and adaptive learning rate (GDMALR) (Endah et al., 2017) adds an adaptive learning rate and momentum to GD. One-step secant (OSS) (Ginantra et al., 2021) BP is a method that tries to bridge the gap between the quasi-Newton and CG algorithms. Although it is more computationally intensive and needs higher memory requirements per epoch than the CG algorithm, its advantage lies in determining the next search direction. Resilient BP (RPROP) (Riedmiller & Braun, 1993) performs a local adaptation of the weight updates by considering the error function to circumvent the disadvantages of GD. The above 14 machine-learning methods do not take into consideration higher-order neurons. One exception was the MCU-ELM approach which utilizes MCUs in all neurons of the SLNN, but it is unable to find the proper combination of sub-cubes for each MCU.

3. Related work

This section describes the cubic and multi-cube units' structure by Gurney (1989). Then, it analyses the ELM training algorithm and presents its extension for MCU SLNNs. The section concludes with a presentation of the generic structure of a GA.

3.1. The cubic unit

The cubic neurons by Gurney (1989) do not assign one weight at each introduced input like traditional low-order units but follow a more complex structure that considers weights as hyper-cube sites. The weight assignment follows the formula $w_{no}^{c} = 2^n, n \in \mathbb{N}$ where the c superscript defines the cubic unit and n are the number of inputs. A visualization of the cubic neuron structure can be seen in Fig. 1

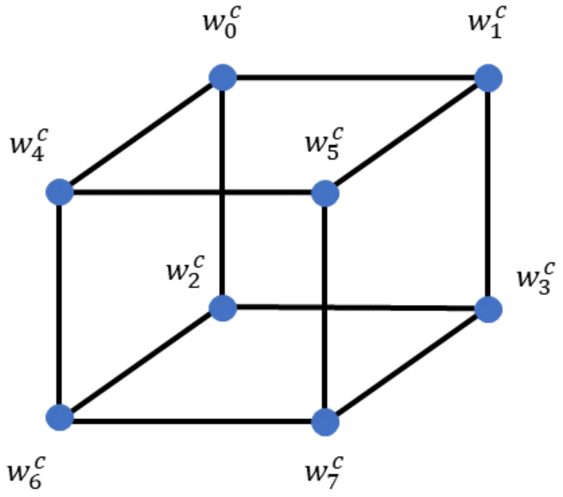

Figure 1: A 3-dimensional hyper-cube. This figure depicts a 3-dimensional hyper-cube with each site assigned to a neuron weight.

The cubic weights follow the probability function (P_μ) depicted in (1), which defines their participation percentage in the neuron's activation. In this equation, n is the number of inputs, and x_i is the current input. The role of $\mu = \mu_1, \mu_2, \ldots, \mu_n$ is to alter the sign of each product term $(1 + \mu_i x_i)$ from plus to minus and vice-versa by converting μ to binary form where each binary symbol corresponds to a plus or minus (1 is defined as a plus while 0 as a minus). The inputs introduced to the cubic neuron must undergo a normalization process, usually dividing each input feature with its highest absolute feature value Gurney (1989).

$$P_\mu = \frac{1}{2^n} \prod_{i=1}^{n} (1 + \mu_i x_i) \tag{1}$$

The cubic activation depicted in equation (2) multiplies each weight with its corresponding probability determined by formula (1). Then, it adds them together and computes their normalized average. The term $\frac{1}{|w_{max^c|}}$ is utilized for normalization purposes, which involve dividing each weight with the highest absolute weight value. The current weight value is defined using the w_μ^c symbol.

$$a^c = \frac{1}{|w^c_{max}|2^n} \sum_{\mu=0}^{2^n-1} w^c_\mu \prod_{i=1}^{n} (1 + \mu_i x_i) \tag{2}$$

In the activation part of the cubic unit depicted in Fig. 2, the input vector $[x_1, \ldots, x_n]$ is introduced to the neuron, which creates the weight probabilities using equation 1.

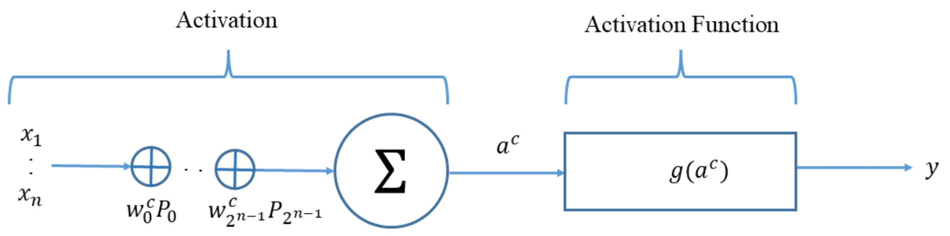

Figure 2: The cubic unit. Creating the neuron output involves introducing the input vector $[x_1, \ldots, x_n]$ to the cubic unit, multiplying each weight with its corresponding probability. The probabilities are calculated using formula (1). After multiplying them by their corresponding weight, they are added together and averaged using a normalized average, forming the neuron's activation. The activation's output is sent to the activation function (g), which computes the neuron's output value y.

Each calculated probability is multiplied by its corresponding weight, they are added all together, and their normalized average is calculated, forming the activation. The outcome from the activation is used as input to the activation function (g), which calculates the neuron's output value y.

3.2. The multi-cube unit

The MCU was created to solve the exponential increase of weights problem in cubic units. Instead of using one high-dimension hyper-cube, the MCU utilizes a series of low-dimension sub-cubes which greatly decreases the number of weights. The sub-cubes forming one MCU can have different dimensions, and the total number of weights in these unit types is defined from the formula $w^{mc}_{no} = p_{no} = \sum_{j=1}^{q} 2^{d_j}, (j, d_j, q) \in \mathbb{N}$. In this equation, the mc superscript defines the MCU type, the number of sub-cubes is indicated by q, and $d = [d_1, d_2, \ldots, d_q]$ contains the dimensions of each sub-cube. Finally, the current sub-cube unit dimension, dj, is derived from vector d.

The activation function of the MCU is defined in formula (3) where the term $\frac{1}{|w^{mc}_{max}|}$ is utilized for normalization purposes, which involve dividing each weight with the highest absolute weight value.

$$a^{mc} = \frac{1}{|w^{mc}_{max}|} \sum_{j=1}^{q} \frac{1}{2^{d_j}} \sum_{\mu=0}^{2^{d_j}-1} w^{mc}_{\mu} \prod_{i=1}^{d_j} (1 + \mu_i x_i) \quad (3)$$

In the activation part of the MCU depicted in Fig. 3, the input vector $[x_1, \ldots, x_n]$ is introduced to the neuron, which divides and assigns these inputs into several sub-cubes. The dimension for each sub-cube is taken from vector $d = [d_1, d_2, \ldots, d_q]$ which results to a $\sum_{j=1}^{q} 2^{d_j}$ total number of weights. For each sub-cube, the calculated probability is multiplied by its corresponding weight, then they are added together and averaged using a normalized average. The outcome from the activation is used as input to the activation function (g), which calculates the neuron's output value y.

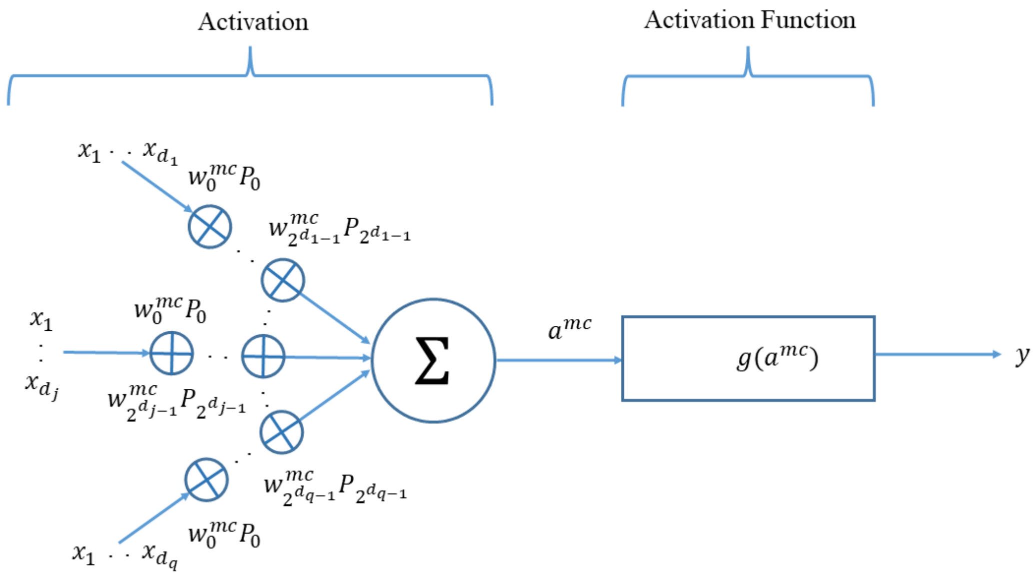
Activation
Activation Function
$x_1 \cdot \cdot x_{d_1}$
$w_0^{mc} P_0$
$w^{mc}_{2^{d_1-1}} P_{2^{d_1-1}}$
x_1
x_{d_j}
$w_0^{mc} P_0$
$w^{mc}_{2^{d_j-1}} P_{2^{d_j-1}}$
Σ
a^{mc}
$g(a^{mc})$
y
$w^{mc}_{2^{d_q-1}} P_{2^{d_q-1}}$
$w_0^{mc} P_0$
$x_1 \cdot \cdot x_{d_q}$

Figure 3: The MCU. Creating the neuron output involves introducing the input vector $[x_1, \ldots, x_n]$ to the MCU, which divides and assigns these inputs into several low-dimension sub-cubes. The weight probabilities of each sub-cube are calculated using formula (1). For each sub-cube, the calculated probability is multiplied by its corresponding weight, then they are added together and averaged using a normalized average. The activation's output is sent to the activation function (g), which computes the neuron's output value y.

3.3. The ELM algorithm

Huang et al. (2006a) utilized an incremental construction method to theoretically prove that SLNNs with randomized hidden weights and thresholds are universal approximators. The mathematical model describing an SLNN which can be trained using ELM is shown in equation 4. The transfer function is denoted with g while $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N \times n} \in \mathbb{R}^{N \times n}$ is the input matrix, $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n \times h} \in \mathbb{R}^{n \times h}$ is the hidden layer weights matrix and $\theta = [\theta_1 \dots \theta_h] \in \mathbb{R}^h$ is the vector containing the hidden layer thresholds. The l superscript defines the low-order unit type, n is the number of neuron inputs, h denotes the number of hidden layer units, and N is the number of input patterns. The matrix $\beta^l = \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h \times m} \in \mathbb{R}^{h \times m}$ contains the output layer weights with m being the number of output units. Finally, the matrix $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$ contains the target values.

$$\begin{bmatrix} g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) & \dots & g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \\ \vdots & \dots & \vdots \\ g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) & \dots & g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \end{bmatrix}_{N \times h} \cdot \qquad (4)$$

$$\begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h \times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$$

The ELM method is described in Algorithm 1. The algorithm begins with the randomization of the hidden layer parameters (weights and thresholds) in steps 1 and 2. The following line involves the construction of the hidden layer output matrix H containing the output values of every training sample introduced to the SLNN. The fourth step is the construction of a matrix carrying the target output values. The final line computes the output layer weights matrix by multiplying the Moore-Penrose pseudo-inverse of H with the target output matrix T.

Algorithm 1 : ELM

```
1: $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n \times h}$
2: $\theta = [\theta_1, \dots, \theta_h]$
3: $H = \begin{bmatrix} g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) \dots g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \\ \vdots \dots \vdots \\ g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) \dots g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \end{bmatrix}_{N \times h}$
4: $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$
5: $\beta^l = H^\dagger T$
```

3.4. The MCU-ELM algorithm

The MCU-ELM method is an extension of the ELM algorithm for SLNNs containing the MCUs defined by Gurney (1989) in all their layers. The mathematical model defining an MCU SLNN, which can be trained with the MCU-ELM algorithm, is defined in formula 5. P denotes the MCU probability function and g defines the transfer function. The matrix containing the MCU activations is defined with $a^{mc} \in \mathbb{R}^{N \times h} =$

$$\begin{bmatrix} a^{mc}_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1}\end{bmatrix}\right) & \dots & a^{mc}_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h}\end{bmatrix}\right) \\ \vdots & \dots & \vdots \\ a^{mc}_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1}\end{bmatrix}\right) & \dots & a^{mc}_{N,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h}\end{bmatrix}\right) \end{bmatrix}_{N\times h}.$$

Inside the a^{mc} matrix, $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n}\end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ is the input matrix and $w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{mc}_{p_h-1,h}\end{bmatrix}_{p_h\times h} \in \mathbb{R}^{p_h\times h}$ is the hidden layer weights matrix. The mc superscript defines the MCU type, n is the number of neuron inputs, h denotes the number of hidden layer units, N is the number of input patterns, p_h is the hidden layer weights number, and p_o denotes the number of output layer weights.

The matrix $\beta^{mc} = \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m}\end{bmatrix}_{p_o\times m} \in \mathbb{R}^{p_o\times m}$ contains the output layer weights with m being the number of output units. Finally, the matrix $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m}\end{bmatrix}_{N\times m} \in \mathbb{R}^{N\times m}$ contains the target values.

$$\begin{bmatrix} P_{1,0}\left(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\right) & \dots & P_{1,p_o-1}\left(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\right) & \dots & P_{N,p_o-1}\left(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\right) \end{bmatrix}_{N\times p_o} \cdot \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m}\end{bmatrix}_{p_o\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m}\end{bmatrix}_{N\times m} \quad (5)$$

The MCU-ELM method is described in Algorithm 2. The algorithm begins with the randomization of the hidden layer weights in step 1. The next line involves the construction of the hidden layer output matrix H containing the output values of every training sample introduced to the SLNN. The third step is the construction of a matrix carrying the target output values. The final line computes the output layer weights matrix by multiplying the Moore-Penrose pseudo-inverse of H with the target output matrix T.

Algorithm 2 :MCU-ELM

```
1: $w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{c}_{p_h-1,h} \end{bmatrix}_{n\times h}$
2: $H = \begin{bmatrix} P_{1,0}\left(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\right)\dots P_{1,p_o-1}\left(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\right) \\ \vdots\dots\vdots \\ P_{N,0}\left(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\right)\dots P_{N,p_o-1}\left(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\right) \end{bmatrix}_{N\times p_o}$
3: $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m}$
4: $\beta^{mc} = H^{\dagger}T$
```

3.5. The GA

The GA is a method inspired by the Darwinian theory of biological evolution developed by Holland (1975) and his colleagues. The main idea behind the creation of GAs is natural selection, where stronger species have more chances to pass their genetic material (genes) to their offspring through reproduction. In succeeding generations, stronger (fitter) species dominate the population and eliminate the weaker ones. During this evolution process, alternation in a few genes may occur, giving additional advantage to the produced offspring (Konak et al., 2006).

GAs are used for solving optimization problems. The structure of a generic GA can be seen in Algorithm 3. In a GA, one possible solution is represented by a chromosome (individual) constituted by genes. Each gene may control more than one chromosome feature, and each possible solution is mapped to an individual using an encoding procedure. The original GA implementation utilized the binary encoding mechanism, where a binary digit represented each gene. Other encoding mechanisms include real value encoding, where each gene is mapped as a real number, and Gray encoding. The latter has a binary representation, and adjacent integers differ only by one binary digit (Whitley, 2001). The GA begins in step 1 by creating the initial population, usually a set of randomly created possible solutions with Pop_{no} size.

Algorithm 3 : GA

```
1:  $Population \leftarrow create(Pop_{no})$
2:  loop
3:      $Population_{evaluate} = evaluate(Population)$
4:      if $(criterion_{stop} = true)$ or $(solution_{best} = true)$ then
5:          $solution_{best} \leftarrow best(Population_{evaluate})$
6:          return $solution_{best}$
7:      end if
8:      $Population_{select} \leftarrow select(Population_{evaluate})$
9:      $Population_{crossover} \leftarrow crossover(Population_{select})$
10:     $Population \leftarrow mutation(Population_{crossover})$
11: end loop
```

After creating the initial population, the evolutionary process begins. It is an iterative process where each iteration (termed generation) utilizes the evaluation, selection, crossover (recombination), and mutation operators. The evaluation operator assesses the individuals according to a fitness function (step 3). Step 4 checks if the stopping criteria have been met or the best solution has been found. If the condition is satisfied, the best solution from the current population is selected (step 5) and returned to the user (step 6). Otherwise, the evolution process continues for another cycle (Konak et al., 2006).

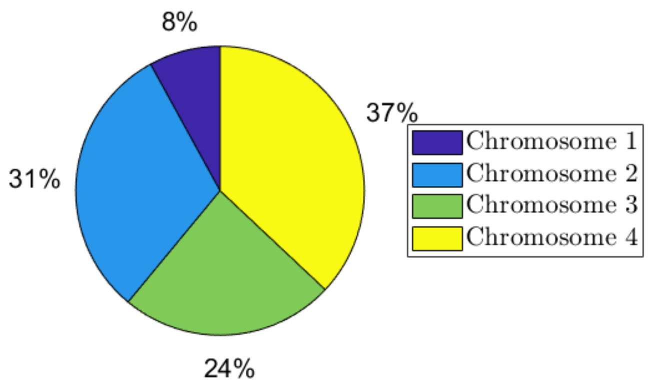

Figure 4: The roulette wheel selection. Each chromosome has a selection percentage assigned proportionally to its fitness value in the roulette wheel selection process. This graph displays four chromosomes, with 8%, 31%, 24%, and 37% selection probabilities assigned accordingly. It is shown that the fourth chromosome has the highest probability (37%) for selection.

The selection operator (step 8) utilizes a roulette wheel mechanism that assigns selection probabilities to the chromosomes according to their fitness. Individuals with higher fitness values are assigned a larger percentage than those with low-fitness values, making them more likely to be selected for creating the next generation. Low-fitness chromosomes are not excluded from the selection process with the purpose of keeping the population diverse. The roulette wheel mechanism with four individuals is visualized in Fig. 4. Other selection mechanisms include rank selection, and local selection Mirjalili (2019).

The reproduction process (step 9) uses the one-point crossover operator, which utilizes the mutual exchange of genetic information between two selected parents at a random crossover point. The reproduction process is visualized in Fig. 5 and produces two offspring. The crossover operator searches the solution locally in the search space. Other recombination mechanisms include multi-point and masked crossover operators Mirjalili (2019).

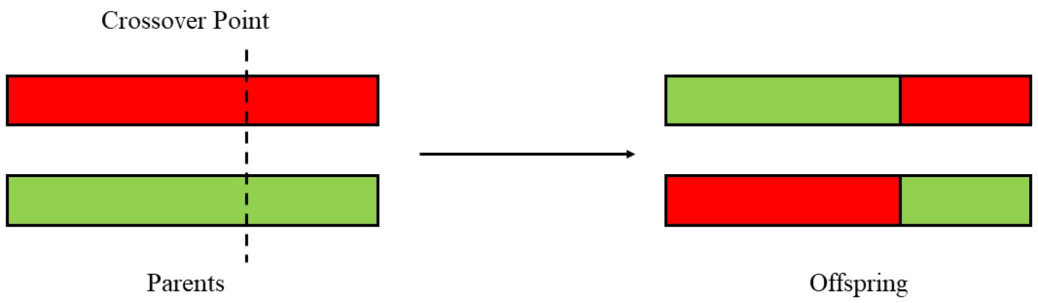

Figure 5: The one-point crossover operator. Using a random crossover point, the one-point crossover operator produces two offspring by mutually exchanging genetic information between two parent chromosomes.

The mutation operator is used for exploring unexplored areas of the search space. It randomly changes the information in one or a few genes in offspring chromosomes. The reason for keeping the mutation rate low is that a high number of mutations would convert the GA to a random search algorithm, a primitive optimization method. The bit mutation operator (used in binary encoding) is depicted in Fig. 6 where a randomly chosen gene has its value changed from 1 to 0 Mirjalili (2019).

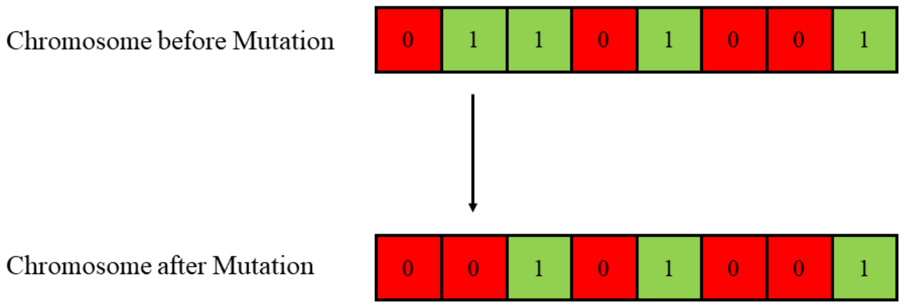

Figure 6: Bit mutation. This figure visualizes bit mutation on the 2^{th} bit where its value changes from 1 to 0.

4. The EHO-ELM algorithm

The EHO-ELM algorithm is a hybrid approach combining the MCU-ELM algorithm presented in section 3.4 with a modified GA, which evolves an initial population of randomly created MCU SLNNs. The evolution process aims to create a more optimal network with a better generalization ability in specific regression and classification problems. The optimization process of the GA is twofold. It optimizes the hidden layer weights of MCUs and can find an optimal combination of sub-cubes for each neuron of the SLNN, including the output layer(s). One of the main goals of EHO-ELM was to retain the simplicity of the original ELM by making all user-defined GA parameters self-adaptive. The structure of EHO-ELM is described in Algorithm 4.

The algorithm begins by creating the initial population (step 1). Each chromosome contains the hidden layer weights and the MCU structure for all neurons of a randomly created SLNN. Each MCU is encoded as a gene, and the size of each randomly created sub-cube unit for each gene varies from 2 to 2^5 weights. The population size (Pop_{no}) is given automatically by using formula 6, which takes into consideration the number of hidden neurons (h).

$$Pop_{no} = \begin{cases} 50, & 2h > 50 \\ 2h, & 0 > 2h \leq 50 \end{cases} \tag{6}$$

Step 2 begins the evolution process. Step 3 trains the networks using MCU-ELM and evaluates them according to a fitness criterion. The classification accuracy (acc) depicted in equation 7 was selected as a fitness criterion for classification problems. In this formula, k is the number of folds, and pat is the total number of input patterns. The err variable denotes the number of erroneously classified patterns for the current fold.

$$acc = \frac{1}{k}\sum_{i=1}^{k}\left(1 - \frac{err}{pat}\right). \tag{7}$$

The mean square error (MSE) defined in equation 8 was selected as a fitness function for regression problems. In this formula, k is the number of folds, and pat is the total number of input patterns. The symbols t_i^j and y_i^j specify the j target network output value and the j network output value for fold i.

$$MSE = \frac{1}{kpat}\sum_{i=1}^{k}\Big(\sum_{j=1}^{pat}(t_i^j - y_i^j)^2\Big) \tag{8}$$

The evaluation process gives a higher score to SLNNs with the highest acc values and SLNNs with the lowest MSE values. In step 4, the best solution is selected.

Algorithm 4 : EHO-ELM

```
1: $Population \leftarrow create(Pop_{no})$
2: loop
3:     $Population_{evaluate} = evaluate(Population)$
4:     $solution_{best} \leftarrow best(Population_{evaluate})$
5:     if $solution_{best}$ is unchanged for 5 generations then
6:         return $solution_{best}$
7:     end if
8:     $Population_{select} \leftarrow select(\frac{Population_{evaluate}}{2})$
9:     $Population_{crossover} \leftarrow crossover(Population_{select})$
10:    $Population \leftarrow mutation(Population_{crossover})$
11: end loop
```

If the best solution remains unchanged for five generations, the algorithm stops and returns the evolved network structure (steps 5-7). If this condition is not satisfied, the evolution process continues in step 8 with the selection mechanism, which selects 50% of the population with the highest fitness scores for the reproduction process (step 9). The reproduction process utilizes a custom crossover operator inspired by the uniform crossover. In the uniform crossover, each gene is selected from either parent using equal probability Syswerda et al. (1989). Similarly, in the EHO-ELM crossover operator, sub-cube units are mutually exchanged between two parent genes using equal probability between each gene's sub-cube. The resulting offspring chromosomes might have different gene input sizes than the parent ones since there is a mutual exchange of sub-cubes that might have different sizes. Due to this reason, an additional procedure adjusting the resulting offspring's chromosomes must be added. If the resulting child gene has more inputs, it randomly removes sub-cubes until it becomes equal or less to the parent gene size. If the child gene has a smaller number of inputs and the input size difference of the child gene from the parent gene is higher than the maximum allowed sub-cube size, it randomly creates a sub-cube unit placed in a random position inside the MCU. This process is repeated until the difference between the number of inputs in the child and the parent gene is equal or less to the maximum allowed size of the sub-cube unit. Then, a sub-cube unit that has input size, the size of this difference, is randomly created and placed in a random position inside the child gene. It should be noted that when sub-cubes for hidden MCUs are created, there is also a random creation of the proper weights. Using this procedure, the input size of the offspring genes stays consistent. An example of the EHO-ELM crossover operator is depicted in Fig 7.

The mutation operator randomly chooses a small number of hidden layer neurons and changes the weight values of one randomly chosen sub-cube from each of these neurons. The mutation operation is self-adaptive (formula 9) and exploits the fitness value of the best SLNN found at each generation. If the evolution process has just begun ($generation = 1$) or a better network has been found ($(generation > 1) \wedge (fitness_{current} > fitness_{previous})$), the mutation rate ($\mu_{rate}$) is set at 10%. Suppose the reproduction process has not created a better child network. In that case, the mutation rate gradually increases by 20% ($\mu_{rate} = \mu_{rate} + 20\%$) until it reaches a maximum value of 50% ($\mu_{rate} = 50\%$).

$$\mu_{rate} = \begin{cases} \mu_{rate} = 10\%, & (generation = 1) \vee ((generation > 1) \\ & \wedge(fitness_{current} > fitness_{previous})) \\ \mu_{rate} = \mu_{rate} + 20\%, & fitness_{current} \leq fitness_{previous} \\ \mu_{rate} = 50\%, & \mu_{rate} \geq 50\% \end{cases} \tag{9}$$

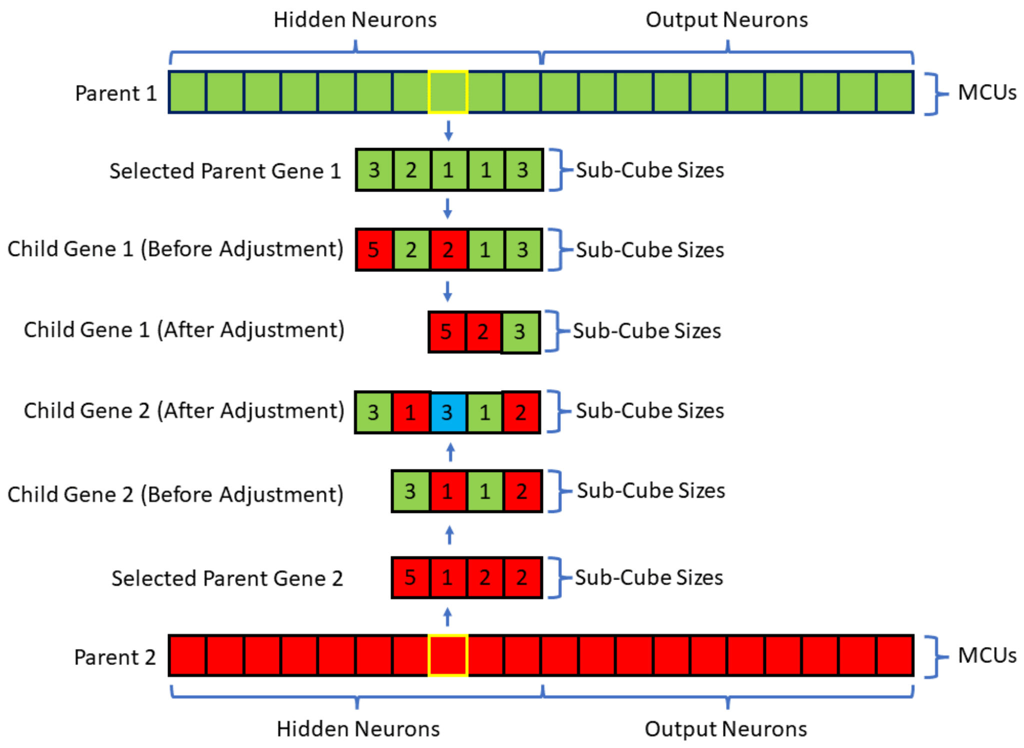

Figure 7: The EHO-ELM crossover operator. This figure visualizes the crossover operation between two parent chromosome genes (green and red). Since the gene contain randomly created MCU structures, where the sub-cube size varies from 1 to 5, they may have different lengths. The recombination is done with a mutual exchange of sub-cube units between two parent genes using equal probability. The reproduction process may result in the creation of offspring with different inputs. Due to this reason, an extra adjustment procedure is added, which removes or creates the necessary sub-cube units for the resulting offspring genes to have the correct input size. The created sub-cubes are depicted with a blue color.

5. Experimental results

The EHO-ELM algorithm was compared with 14 machine learning methods in 12 classification and six regression real-world datasets. All the datasets were taken from the University of California, Irvine (UCI) machine learning repository (Dua & Graff, 2017). In all datasets, the EHO-ELM algorithm achieved the best generalization performance in a test set containing unknown data for both problem types. The statistical significance of these results was verified using the Wilcoxon sign-rank test.

5.1. Dataset characteristics

The characteristics of the classification datasets are summarized in Table 1. The experimental part utilized binary and multi-class classification problems.

Table 1: Attributes of the Classification Datasets

Dataset	**Inputs**	**Outputs**	**Entries**
Balance Scale	4	3	625
Blood Transfusion Service Center	4	2	748
EEG Eye State	14	2	14980
Glass	9	7	214
HIV-1 Protease Cleavage	8	2	746
HTRU2	8	2	17898
Indoor User Movement Prediction From RSS Data	4	2	13197
Leaf	14	36	340
Maternal Health Risk	6	3	1014
Page Blocks	10	5	5473
Vertebral Column	6	2	310
Yeast	8	10	1484

The first data collection is “Balance Scale”, which contains four input features and is created for modeling experimental psychological results. The “Blood Transfusion Service Center” dataset contains four input features received from the Blood Transfusion Service Center in Hsin-Chu city in Taiwan Yeh et al. (2009). The “EEG Eye State” data collection includes 14 input features received from one continuous electroencephalogram (EEG) measurement using the headset by EMOTIV and two outputs corresponding to the open and closed eye states. The “Glass” dataset has six glass types received from USA forensic science service. It contains ten input features, including “ID Number” which has been removed. An octamers list and a binary flag are in “HIV-1 Protease Cleavage”. Whether the HIV-1 protease will cleave in the center determines the value of the flag (Rögnvaldsson et al., 2015). “HTRU2” is a data collection containing entries that describe a sample of pulsar candidates gathered at the high-time resolution universe (HTRU) survey (south) (Lyon et al., 2016). “Indoor User Movement Prediction From RSS Data” includes temporal information gathered from a wireless sensor network set up in physical office settings (Bacciu et al., 2014). “Leaf” contains 36 leaf specimens having 15 input attributes. In the experimental part of the current research, the “specimen number” attribute has been removed (Silva et al., 2013). “Maternal Health Risk” contains data gathered from various maternal health care providers in Bangladesh’s rural areas. “Page Blocks” has entries for categorizing each page layout block of a document that has been identified by segmentation. “Vertebral Column” has six biomechanical features for categorizing orthopedic patients into two classes, and “Yeast” data collection is used for protein cellular localization site prediction tasks.

The characteristics of the regression datasets are summarized in Table 2. The first is “Airfoil Self-Noise” data collection which contains entries from a series of tests on 2D and 3D airfoil blade sections. The tests were performed in an anechoic wind tunnel. ”Auto MPG” has entries for city-cycle fuel usage and contains missing values. These entries were filled in with the average values derived from the available data. “Combined Cycle Power Plant” contains data from a combined cycle power plant working on full load for six years (Tüfekci, 2014; Kaya et al., 2012). “Concrete Compressive Strength” is a data collection containing features that affect concrete compressive strength (Yeh, 1998). “Real Estate Valuation” has historical market data regarding real estate valuation (Yeh & Hsu, 2018), and “Yacht Hydrodynamics” contains features for predicting the hydrodynamic performance of sailing yachts.

Table 2: Attributes of the Regression Datasets

Dataset	**Inputs**	**Outputs**	**Entries**
Airfoil Self-Noise	5	1	1503
Auto MPG	8	1	398
Combined Cycle Power Plant	4	1	9568
Concrete Compressive Strength	8	1	1030
Real Estate Valuation	7	1	414
Yacht Hydrodynamics	6	1	308

5.2. Experimental setup

All data collections were normalized and divided into training/test sets where 80% was used for training, and 20% was retained as a test set. In the EHO-ELM and all BP-based approaches, the training set was further divided into training/validation sets using 10-fold cross-validation. The MCU hidden layer weights were randomly created from the uniform distribution and were restricted inside the $[-1, 1]$ interval. Similarly, the input vector was normalized, and the input values did not exceed the $[-1, 1]$ interval. The sigmoid function was selected as the transfer function for the hidden MCU units, and the number of hidden layer neurons was set to ten for all neural network-based experiments. The neural network-based experiments were repeated ten times with different randomized values for the hidden layer weights and thresholds to avoid any bias due to the initialization of the hidden parameters. Finally, the MCU-ELM method utilized one-dimension subcubes, while the SVM implementation used a linear kernel. The experimental setup parameters for the EHO-ELM are summarized in Table 3.

Table 3: EHO-ELM Experimental Setup Parameters

Parameter Name	**Symbol**	**Values/Types**
Multi-Cube Neuron Weights	w^{mc}	$[-1,1]^{p_{no}}, p_{no} \in \mathbb{N}^*$
Inputs	x	$[-1,1]^n, n \in \mathbb{N}^*$
Activation Function	g	*sigmoid*
Hidden Layer Neurons No	h	10
Folds No	k	10
Experiment Sets	$expNo$	10

5.3. Classification problems

EHO-ELM was compared with ten BP-based methods (BFGS, PBCG, FRCG, PRCG, SCG, GD, GDM, GDMALR, OSS, and RPROP), the SVM algorithm, and three ELM-based approaches (OS-ELM, original ELM, and MCU-ELM) in 12 classification datasets. The experimental results are summarized in Table 4, where it is evident that EHO-ELM managed to achieve the highest classification accuracy in all datasets.

Table 4: Classification Problems Experimental Results

Datasets	**Balance Scale**	**Blood Transfusion Service Center**	**EEG Eye State**	**Glass**	**HIV-1 Protease Cleavage**	**HTRU-2**
BFGS	43.83%	66.48%	51.95%	24.81%	54.47%	74.91%
PBCG	72.38%	59.32%	50%	20.93%	54.46%	68.8%
FRCG	69.77%	55.4%	49.87%	25.67%	54.07%	69.47%
PRCG	73.46%	58.64%	50.2%	24.93%	53.33%	68.1%
SCG	64.68%	65.82%	52.56%	32.28%	55.95%	79.66%
GD	38.39%	58.81%	52.24%	23.63%	50.73%	71.92%
GDM	37.78%	57.73%	52.33%	23.37%	49.85%	68.38%
GDMALR	46.53%	57%	51.58%	22.79%	54.44%	78.41%
OSS	45.54%	68.01%	51.21%	23.35%	56.3%	74.96%
RPROP	87.3%	73.28%	54.65%	52.35%	69.19%	96.49%
SVM	86.4%	74%	54.67%	60.47%	75.17%	97.65%
OS-ELM	88.96%	72.47%	57.94%	66.05%	74.7%	96.84%
ELM	88.96%	74.93%	58.22%	65.81%	74.97%	96.95%
MCU-ELM	90%	75.33%	59%	68.84%	75.3%	97.01%
EHO-ELM	90.62%	75.58%	69.34%	72.44%	77.08%	97.77%

Datasets	**Indoor User Movement Prediction From RSS Data**	**Leaf**	**Maternal Health Risk**	**Page Blocks**	**Vertebral Column**	**Yeast**
BFGS	53.9%	3.21%	38.32%	59.21%	58.81%	17.16%
PBCG	52.89%	6.68%	35.78%	31%	53.35%	18.05%
FRCG	51.92%	5.31%	35.59%	36.93%	50.34%	21.76%
PRCG	52.42%	6.93%	34.61%	38.53%	52.63%	20.59%
SCG	53.42%	5%	40%	46.8%	57.44%	23.48%
GD	52.13%	2.9%	34.82%	40.74%	53.73%	17.47%
GDM	52.89%	3.32%	35.47%	47.81%	53.37%	18.6%
GDMALR	53.03%	3.22%	38.32%	49.19%	55.52%	15.77%
OSS	53.77%	3.13%	37.7%	58.4%	59.08%	17.72%
RPROP	59.42%	26.44%	52.88%	89.04%	68.26%	31.61%
SVM	61.69%	38.24%	59.11%	91.87%	64.52%	56.57%
OS-ELM	64.7%	32.65%	64.73%	91.32%	80.16%	57.17%
ELM	64.47%	33.24%	64.53%	91.21%	78.39%	57.81%
MCU-ELM	64.95%	36.47%	65.47%	91.39%	78.87%	57.88%
EHO-ELM	69.39%	57.57%	66.5%	94.99%	81.89%	59.4%

The significance of these results was evaluated using the Wilcoxon signed-ranked test, a non-parametric statistical hypothesis test used to compare the locations of two populations by utilizing two matched samples or to assess the location of a population based on a sample of data (Wilcoxon; Conover, 1999). The outcome of the Wilcoxon signed-ranked test is a p-value, corresponding to a probability that the two compared populations are the same. The output from this statistical test between EHO-ELM and the 14 alternative methods is summarized in Fig 8. In all cases, the p-value from these comparisons was less than 5%, indicating that the results from EHO-ELM are statistically significant.

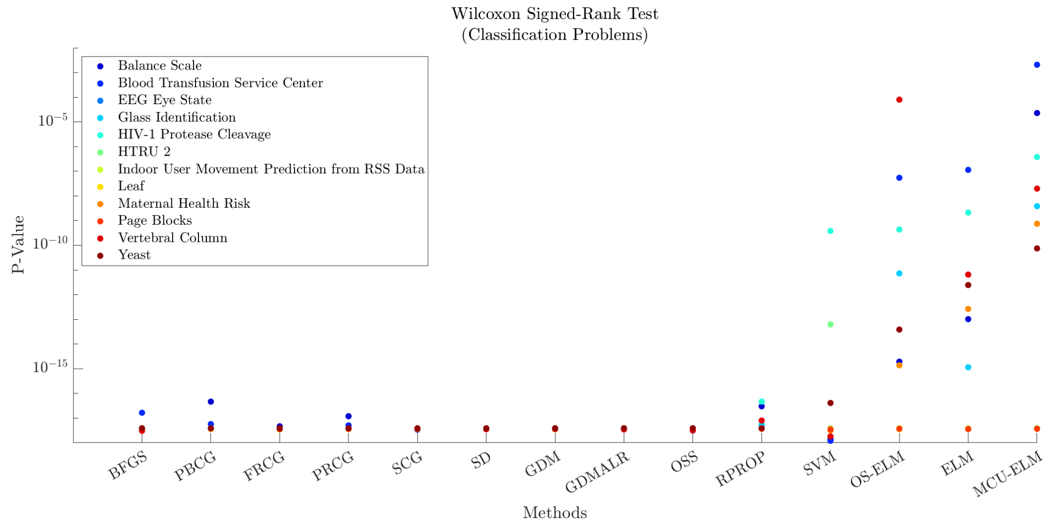

Figure 8: Wilcoxon Signed-Rank Test. This figure visualizes the experimental results from comparing EHO-ELM with 14 machine learning methods (BFGS, PBCG, FRCG, PRCG, SCG, GD, GDM, GDMALR, OSS, RPROP, SVM, OS-ELM, ELM, and MCU-ELM) in 12 classification datasets. It is shown that in all cases, the p-value was below 5%, indicating that the results from the comparison are statistically significant.

5.4. Regression problems

EHO-ELM was also compared with the same ten BP-based methods (BFGS, PBCG, FRCG, PRCG, SCG, GD, GDM, GDMALR, OSS, and RPROP), the SVM algorithm, and three ELM-based approaches (OS-ELM, original ELM, and MCU-ELM) in six regression datasets. The experimental results are summarized in Table 5, showing that EHO-ELM achieved the lowest MSE in all datasets.

Table 5: Regression Problems Experimental Results

Datasets	**Airfoil Self-Noise**	**Auto MPG**	**Combined Cycle Power Plant**	**Concrete Compressive Strength**	**Real Estate Valuation**	**Yacht Hydro-dynamics**
BFGS	0.001009	0.004076	0.000077	0.009913	0.004873	0.005284
PBCG	0.000886	0.004	0.000072	0.009746	0.004634	0.001548
FRCG	0.00091	0.003958	0.000072	0.010472	0.004706	0.00217
PRCG	0.000923	0.00413	0.000073	0.010999	0.004826	0.004055
SCG	0.000924	0.004032	0.000074	0.011631	0.004997	0.002518
GD	0.009912	0.014348	0.00561	0.033193	0.016024	0.031566
GDM	0.010161	0.013987	0.005975	0.033274	0.016757	0.033116
GDMALR	0.001052	0.008622	0.000125	0.021146	0.008427	0.008419
OSS	0.001216	0.004457	0.000097	0.011962	0.005264	0.003304
RPROP	0.000841	0.004311	0.000072	0.010924	0.005008	0.002244
SVM	0.001166	0.004692	0.000083	0.018107	0.005268	0.022118
OS-ELM	0.001042	0.00401	0.000074	0.016267	0.005092	0.011822
ELM	0.001032	0.004314	0.000074	0.015039	0.005196	0.011498
MCU-ELM	0.001013	0.004153	0.000073	0.013898	0.005015	0.010085
EHO-ELM	0.000503	0.003272	0.00007	0.006968	0.003846	0.000118

The significance of these results was evaluated using the Wilcoxon signed-ranked test. The outcome from this statistical test between EHO-ELM and the 14 alternative methods is summarized in Fig 9. In all cases, the p-value from these comparisons was less than 5%, indicating that the results from EHO-ELM are statistically significant.

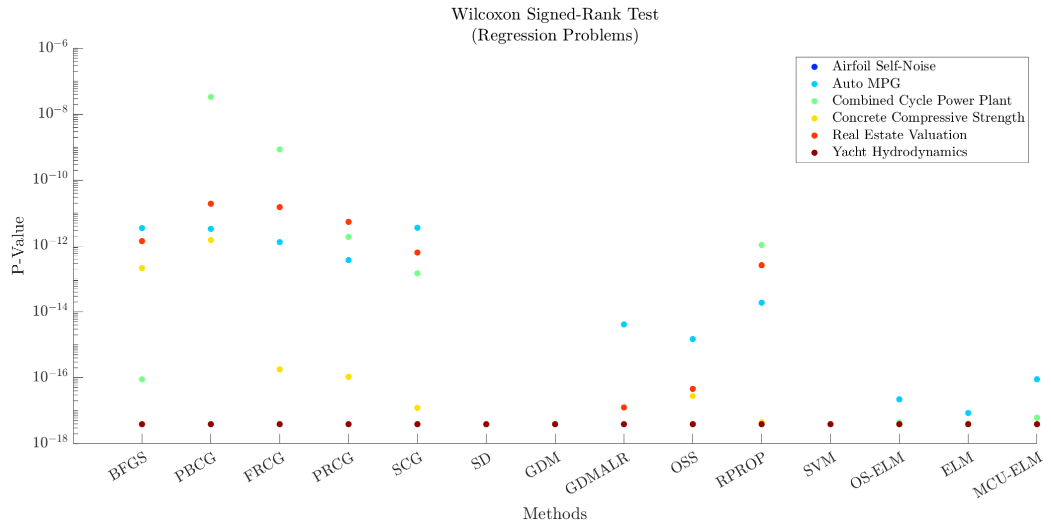

Figure 9: Wilcoxon Signed-Rank Test. This figure visualizes the experimental results from comparing EHO-ELM with 14 machine learning methods (BFGS, PBCG, FRCG, PRCG, SCG, GD, GDM, GDMALR, OSS, RPROP, SVM, OS-ELM, ELM, and MCU-ELM) in six classification datasets. It is shown that in all cases, the p-value was below 5%, indicating that the results from the comparison are statistically significant.

6. Discussion

The experimental study in this article verified that the EHO-ELM algorithm performs better than 14 machine-learning approaches in 12 datasets. One interesting observation from the “EEG Eye State” dataset is that EHO-ELM achieved 69.34% accuracy and outperformed the MCU-ELM method by 10.34%, which was the second best with 59% accuracy. MCU-ELM utilizes one-dimension sub-cubes and is unable to optimize the hidden layer parameters. Although MCU-ELM is a faster method than EHO-ELM since it trains only one neural network compared to EHO-ELM’s evolutionary process, which trains many networks at each generation, its generalization ability is inferior. A similar observation is shown in the “Leaf” dataset. The proposed method achieved a 57.57% accuracy compared to the SVM method, which was the second best with 38.04%, giving a 19.33% difference between these two methods. The “Leaf” dataset is categorized as a multi-class problem since it contains features from 36 leaf specimens justifying the low accuracy values achieved in this data collection from all methods. Similar observations were noted in the regression problems. Specifically in the “Yacht

Hydrodynamics" dataset, EHO-ELM achieved a 0.000118 MSE which is an order less than the 0.001548 MSE of the PBCG method having the second best generalization performance.

This article aims to present a method capable of automatically selecting an optimal combination of sub-cubes for each MCU utilized in creating an SLNN that can be trained with the MCU-ELM algorithm while retaining its simplicity. Exhausting all possible combinations of sub-units cannot be done quickly since the search space increases exponentially with every neuron introduced to the network. Due to this reason, a custom GA was created to find an optimal network and optimize the hidden layer parameters in an acceptable time. All GA parameters are self-adaptive, thus retaining ELM's simplicity

One issue of EHO-ELM is that it is a computationally intensive method because it requires training a series of SLNNs at each evolution cycle. The adopted solution to this problem was the utilization of a multi-core system for training in parallel the SLNNs at each generation.

7. Conclusion

The EHO-ELM algorithm utilized an evolutionary method to train a series of SLNNs containing MCU units in both layers. The evolutionary method was a custom-created GA able to estimate the number and dimensions of each neuron's sub-cubes and optimize the hidden layer parameters. The evolution process finishes when the best network found remains the same for five generations, and this network is selected as the most optimal.

The proposed method was evaluated in 12 datasets with 14 machine learning methods. EHO-ELM outperformed all the compared methods in classification and regression problems by achieving the highest accuracy and lowest MSE values accordingly. The experimental results' significance was verified using the Wilcoxon signed-rank test. EHO-ELM was compared with each alternative method; in all cases, the p-value was less than 5% indicating that the two methods are different.

Although the proposed method trains a series of SLNNs, it is not an ensemble method because the best network found according to fitness is selected at the end of the evolution process. Future work will focus on adapting EHO-ELM to an ensemble method by creating and combining several models for improving the accuracy in classification problems or reducing the MSE in regression problems.

CRediT authorship contribution statement

Vasileios Christou: Conceptualization, Methodology, Software, Writing – original draft, Investigation, Resources. **Alexandros T. Tzallas:** Visualization, Validation, Writing – review & editing, Supervision, Project administration. **Markos G. Tsipouras:** Data curation, Validation, Writing – review & editing, Supervision, Project administration. **Nikolaos Giannakeas:** Formal analysis, Validation, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

We acknowledge support of this work by the project “MEGATRON” (MIS 5047227) which is implemented under the Action “Reinforcement of the Research and Innovation Infrastructure”, funded by the Operational Programme “Competitiveness, Entrepreneurship and Innovation” (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

References

Albadr, M. A. A., Tiun, S., Ayob, M., & Al-Dhief, F. T. (2022). Particle swarm optimization-based extreme learning machine for covid-19 detection. *Cognitive Computation*, (pp. 1–16).

Alexander, V., & Annamalai, P. (2016). An elitist genetic algorithm based extreme learning machine. In *Computational Intelligence, Cyber Security and Computational Models* (pp. 301–309). Springer.

Askarzadeh, A. (2016). A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. *Computers & structures*, *169*, 1–12.

Bacciu, D., Barsocchi, P., Chessa, S., Gallicchio, C., & Micheli, A. (2014). An experimental characterization of reservoir computing in ambient assisted living applications. *Neural Computing and Applications*, *24*, 1451–1464.

Bhat, A. U., Merchant, S. S., & Bhagwat, S. S. (2008). Prediction of melting points of organic compounds using extreme learning machines. *Industrial & engineering chemistry research*, *47*, 920–925.

Broyden, C. G. (1970a). The convergence of a class of double-rank minimization algorithms: 1. general considerations. *IMA Journal of Applied Mathematics*, *6*, 76–90.

Broyden, C. G. (1970b). The convergence of a class of double-rank minimization algorithms: 2. the new algorithm. *IMA journal of applied mathematics*, *6*, 222–231.

Cai, Z., Gu, J., Luo, J., Zhang, Q., Chen, H., Pan, Z., Li, Y., & Li, C. (2019). Evolving an optimal kernel extreme learning machine by using an enhanced grey wolf optimization strategy. *Expert Systems with Applications*, *138*, 112814.

Cao, J., Lin, Z., & Huang, G.-B. (2012). Self-adaptive evolutionary extreme learning machine. *Neural processing letters*, *36*, 285–305.

Cao, L., Yue, Y., Zhang, Y., & Cai, Y. (2021). Improved crow search algorithm optimized extreme learning machine based on classification algorithm and application. *Ieee Access*, *9*, 20051–20066.

Cauchy, A. et al. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, *25*, 536–538.

Christou, V. (2023). Higher-order extreme learning machine. Unpublished.

Christou, V., Koritsoglou, K., Ntritsos, G., Tsoumanis, G., Tsipouras, M. G., Giannakeas, N., Glavas, E., & Tzallas, A. T. (2022). Heterogeneous hybrid extreme learning machine for temperature sensor accuracy improvement. *Expert Systems with Applications*, *203*, 117488.

Christou, V., Ntritsos, G., Tzallas, A. T., Tsipouras, M. G., & Giannakeas, N. (2020). Self-adaptive hybrid extreme learning machine for heterogeneous neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). IEEE.

Christou, V., Tsipouras, M. G., Giannakeas, N., & Tzallas, A. T. (2018). Hybrid extreme learning machine approach for homogeneous neural networks. *Neurocomputing*, *311*, 397–412.

Christou, V., Tsipouras, M. G., Giannakeas, N., Tzallas, A. T., & Brown, G. (2019). Hybrid extreme learning machine approach for heterogeneous neural networks. *Neurocomputing*, *361*, 137–150.

Conover, W. J. (1999). *Practical nonparametric statistics* volume 350. john wiley & sons.

Dong, J., Wu, L., Liu, X., Li, Z., Gao, Y., Zhang, Y., & Yang, Q. (2020). Estimation of daily dew point temperature by using bat algorithm optimization based extreme learning machine. *Applied Thermal Engineering*, *165*, 114569.

Dou, J., Ma, H., Zhang, Y., Wang, S., Ye, Y., Li, S., & Hu, L. (2022). Extreme learning machine model for state-of-charge estimation of lithium-ion battery using salp swarm algorithm. *Journal of Energy Storage*, *52*, 104996.

Dua, D., & Graff, C. (2017). UCI machine learning repository. URL: `http://archive.ics.uci.edu/ml`.

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the sixth international symposium on micro machine and human science* (pp. 39–43). Ieee.

Endah, S. N., Widodo, A. P., Fariq, M. L., Nadianada, S. I., & Maulana, F. (2017). Beyond back-propagation learning for diabetic detection: Convergence comparison of gradient descent, momentum and adaptive learning rate. In *2017 1st international conference on informatics and computational sciences (ICICoS)* (pp. 189–194). IEEE.

Faris, H., Mirjalili, S., Aljarah, I., Mafarja, M., & Heidari, A. A. (2020). Salp swarm algorithm: theory, literature review, and application in extreme learning machines. *Nature-inspired optimizers*, (pp. 185–199).

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive science*, *6*, 205–254.

Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal*, *13*, 317–322.

Fletcher, R., & Reeves, C. M. (1964). Function minimization by conjugate gradients. *The computer journal*, *7*, 149–154.

Ginantra, N., Bhawika, G. W., Daengs, G. A., Panjaitan, P. D., Arifin, M. A., Wanto, A., Amin, M., Okprana, H., Syafii, A., & Anwar, U. (2021). Performance one-step secant training method for forecasting cases. In *Journal of Physics: Conference Series* (p. 012032). IOP Publishing volume 1933.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, *24*, 23–26.

Gurney, K. N. (1989). *Learning in networks of structured hypercubes*. Ph.D. thesis Brunel University London, UK.

Han, F., Yao, H.-F., & Ling, Q.-H. (2013). An improved evolutionary extreme learning machine based on particle swarm optimization. *Neurocomputing*, *116*, 87–93.

Heidari, A. A., Abbaspour, R. A., & Chen, H. (2019). Efficient boosted grey wolf optimizers for global search and kernel extreme learning machine training. *Applied Soft Computing*, *81*, 105521.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Huang, G., Song, S., Gupta, J. N., & Wu, C. (2014). Semi-supervised and unsupervised extreme learning machines. *IEEE transactions on cybernetics*, *44*, 2405–2417.

Huang, G.-B., Chen, L., Siew, C. K. et al. (2006a). Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks*, *17*, 879–892.

Huang, G.-B., Liang, N.-Y., Rong, H.-J., Saratchandran, P., & Sundararajan, N. (2005). On-line sequential extreme learning machine. *Computational Intelligence*, *2005*, 232–237.

Huang, G.-B., Zhou, H., Ding, X., & Zhang, R. (2011). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *42*, 513–529.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)* (pp. 985–990). Ieee volume 2.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006b). Extreme learning machine: theory and applications. *Neurocomputing*, *70*, 489–501.

Javed, K., Gouriveau, R., & Zerhouni, N. (2014). Sw-elm: A summation wavelet extreme learning machine algorithm with a priori parameter initialization. *Neurocomputing*, *123*, 299–307.

Karimkashi, S., & Kishk, A. A. (2010). Invasive weed optimization and its features in electromagnetics. *IEEE transactions on antennas and propagation*, *58*, 1269–1278.

Kaya, H., Tüfekci, P., & Gürgen, F. S. (2012). Local and global learning methods for predicting power of a combined gas & steam turbine. In *Proceedings of the international conference on emerging trends in computer and electronics engineering ICETCEE* (pp. 13–18).

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability engineering & system safety*, *91*, 992–1007.

Lyon, R. J., Stappers, B., Cooper, S., Brooke, J. M., & Knowles, J. D. (2016). Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, *459*, 1104–1123.

Mengcan, M., Xiaofang, C., & Yongfang, X. (2021). Constrained voting extreme learning machine and its application. *Journal of Systems Engineering and Electronics*, *32*, 209–219.

Meza, J. C. (2010). Steepest descent. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*, 719–722.

Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., & Lendasse, A. (2009). Op-elm: optimally pruned extreme learning machine. *IEEE transactions on neural networks*, *21*, 158–162.

Miche, Y., Van Heeswijk, M., Bas, P., Simula, O., & Lendasse, A. (2011). Trop-elm: a double-regularized elm using lars and tikhonov regularization. *Neurocomputing*, *74*, 2413–2421.

Mirjalili, S. (2019). Evolutionary algorithms and neural networks. In *Studies in computational intelligence*. Springer volume 780.

Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017). Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in engineering software*, *114*, 163–191.

Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in engineering software*, *95*, 51–67.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, *69*, 46–61.

Mirjalili, S. Z., Mirjalili, S., Saremi, S., Faris, H., & Aljarah, I. (2018). Grasshopper optimization algorithm for multi-objective optimization problems. *Applied Intelligence*, *48*, 805–820.

Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, *6*, 525–533.

Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, *826*, 37.

Polak, E., & Ribiere, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, *3*, 35–43.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, *4*, 1–17.

Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method. *Mathematical programming*, *12*, 241–254.

Rathod, N., & Wankhade, S. (2022). Optimizing neural network based on cuckoo search and invasive weed optimization using extreme learning machine approach. *Neuroscience Informatics*, (p. 100075).

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE international conference on neural networks* (pp. 586–591). IEEE.

Rögnvaldsson, T., You, L., & Garwicz, D. (2015). State of the art prediction of hiv-1 protease cleavage sites. *Bioinformatics*, *31*, 1204–1210.

Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (1995). Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, (pp. 1–34).

Rumelhart, D. E., Hinton, G. E., McClelland, J. L. et al. (1986a). A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, *1*, 26.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986b). Learning representations by back-propagating errors. *nature*, *323*, 533–536.

Scales, L. (1985). *Introduction to non-linear optimization*. Springer-Verlag.

Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, *24*, 647–656.

Shariati, M., Mafipour, M. S., Ghahremani, B., Azarhomayun, F., Ahmadi, M., Trung, N. T., & Shariati, A. (2022). A novel hybrid extreme learning machine–grey wolf optimizer (elm-gwo) model to predict compressive strength of concrete with partial replacements for cement. *Engineering with Computers*, (pp. 1–23).

Shin, Y., & Ghosh, J. (1991). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *IJCNN-91-Seattle international joint conference on neural networks* (pp. 13–18). IEEE volume 1.

Silva, P. F., Marcal, A. R., & da Silva, R. M. A. (2013). Evaluation of features for leaf discrimination. In *Image Analysis and Recognition: 10th International Conference, ICIAR 2013, Póvoa do Varzim, Portugal, June 26-28, 2013. Proceedings 10* (pp. 197–204). Springer.

Similä, T., & Tikka, J. (2005). Multiresponse sparse regression with application to multidimensional scaling. In *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005: 15th International Conference, Warsaw, Poland, September 11-15, 2005. Proceedings, Part II 15* (pp. 97–102). Springer.

Storn, R. (1996). On the usage of differential evolution for function optimization. In *Proceedings of north american fuzzy information processing* (pp. 519–523). Ieee.

Storn, R., & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*, 341.

Syswerda, G. et al. (1989). Uniform crossover in genetic algorithms. In *ICGA* (pp. 2–9). volume 3.

Tüfekci, P. (2014). Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, *60*, 126–140.

Wang, W., Liu, Y., Song, F., & Wang, Y. (2022). Color trend prediction method based on genetic algorithm and extreme learning machine. *Color Research & Application*, *47*, 942–952.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, .

Whitley, D. (2001). An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and software technology*, *43*, 817–831.

Wilcoxon, F. (). Individual comparisons by ranking methods, biometrics bulletin 1 (1945) 80–83. *URL: http://www.jstor.org/stable/3001968.doi*, *10*, 3001968.

Yang, W., Xia, K., Fan, S., Wang, L., Li, T., Zhang, J., & Feng, Y. (2022). A multi-strategy whale optimization algorithm and its application. *Engineering Applications of Artificial Intelligence*, *108*, 104558.

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm. *Nature inspired cooperative strategies for optimization (NICSO 2010)*, (pp. 65–74).

Yang, X.-S., & Deb, S. (2009). Cuckoo search via lévy flights. In *2009 World congress on nature & biologically inspired computing (NaBIC)* (pp. 210–214). Ieee.

Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, *28*, 1797–1808.

Yeh, I.-C., & Hsu, T.-K. (2018). Building real estate valuation models with comparative approach through case-based reasoning. *Applied Soft Computing*, *65*, 260–271.

Yeh, I.-C., Yang, K.-J., & Ting, T.-M. (2009). Knowledge discovery on rfm model using bernoulli sequence. *Expert Systems with Applications*, *36*, 5866–5871.

Yu, C., Chen, M., Cheng, K., Zhao, X., Ma, C., Kuang, F., & Chen, H. (2021). Sgoa: annealing-behaved grasshopper optimizer for global tasks. *Engineering with Computers*, (pp. 1–28).

Zahir, N., & Mahdi, H. (2015). Snow depth estimation using time series passive microwave imagery via genetically support vector regression (case study urmia lake basin). *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *40*, 555.

Zhang, Y., Wu, J., Cai, Z., Zhang, P., & Chen, L. (2016). Memetic extreme learning machine. *Pattern Recognition*, *58*, 135–148.

Zhu, Q.-Y., Qin, A. K., Suganthan, P. N., & Huang, G.-B. (2005). Evolutionary extreme learning machine. *Pattern recognition*, *38*, 1759–1763.