Hybrid extreme learning machine approach for heterogeneous neural networks

DOI:
10.1016/j.neucom.2019.04.092

Document Version
Submitted manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
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. https://doi.org/10.1016/j.neucom.2019.04.092

Published in:
Neurocomputing

Hybrid extreme learning machine approach for heterogeneous neural networks

Vasileios Christou[a,*], Markos G. Tsipouras[b,c], Nikolaos Giannakeas[c], Alexandros T. Tzallas[c], Gavin Brown[a]

[a] *School of Computer Science, The University of Manchester, Oxford Road, M13 9PL, Manchester, United Kingdom*

[b] *Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Kozani, Greece*

[c] *School of Applied Technology, Department of Computer Engineering, Technological Educational Institute of Epirus, Kostakioi, GR-47100, Arta, Greece*

Abstract

In this paper, a hybrid learning approach, which combines the extreme learning machine (ELM) with a genetic algorithm (GA), is proposed. The utilization of this hybrid algorithm enables the creation of heterogeneous single layer neural networks (SLNNs) with better generalization ability than traditional ELM in terms of lower mean square error (MSE) for regression problems or higher accuracy for classification problems. The architecture of this method is not limited to traditional linear neurons, where each input participates equally to the neuron's activation, but is extended to support higher order neurons which affect the network's generalization ability. Initially, the proposed heterogeneous hybrid extreme learning machine (He-HyELM) algorithm creates a number of custom created neurons with different structure, which are used for the creation of homogeneous SLNNs. These networks are trained with ELM and an application specific GA evolves them into heterogeneous networks according to a fitness criterion utilizing the uniform crossover operator for the recombination process. After the completion of the evolution process, the network with the best fitness is selected as the most optimal.

* Corresponding Author.

Email addresses: `vasileios.christou@postgrad.manchester.ac.uk` (Vasileios Christou), `tsipouras@teiep.gr` (Markos G. Tsipouras), `giannakeas@teiep.gr` (Nikolaos Giannakeas), `tzallas@teiep.gr` (Alexandros T. Tzallas), `gavin.brown@manchester.ac.uk` (Gavin Brown)

Experimental results demonstrate that the proposed learning algorithm can get better results than traditional ELM, homogeneous hybrid extreme learning machine (Ho-HyELM) and optimally pruned extreme learning machine (OP-ELM) for homogeneous and heterogeneous SLNNs.

Keywords:
artificial neural network, classification problem, custom neuron, genetic algorithm, hybrid extreme learning machine, regression problem

1. Introduction

The extreme learning machine (ELM) learning algorithm proposed by Huang et al. [31, 32] for single layer neural networks (SLNNs) was created with purpose to overcome the slow training speed problem of gradient based algorithms. The main advantages of this approach include simplicity (since it doesn't require a learning rate and stopping criteria), efficiency (since it doesn't stuck in local minima) and it's ability to train a SLNN in one calculation step without the need of an iterative process like in gradient based approaches. The main characteristic of ELM is that it tends to reach low training error with small norm of output weights which according to Bartlett [6] leads to better generalization performance [69].

ELM trains a network without adapting the hidden layer weights and thresholds but by randomly selecting them; then with the use of the Moore-Penrose generalized pseudo-inverse analytically determines the output node weights. This approach has shown to be effective for both regression and classification problems although it may require more nodes than networks trained with gradient-based methods like back-propagation [24] due to the randomization of the hidden layer weights and thresholds [72, 69]. Furthermore, the ELM algorithm is unable to create SLNNs containing different combinations of neuron types[1] in the hidden layer (these network types will

[1] The term different combinations of neuron types refers to neuron types with structural units like the structured composite model (C-Model) proposed by Christou et al. [9] which is utilized in the proposed algorithm. This model divides the neuron into three subunits (subcomponents) named dendrite (D), activation (S_a) function, activation-output (S_{ao}) function and is able to create a custom neuron by varying any of the subunits from each category. Finally, supports both traditional linear neurons where each input participates equally to the neuron's activation and higher order neurons (the higher order units used in this paper will be analysed in Section 3).

be termed heterogeneous networks). Most previous studies contained SLNNs with the same neuron types in the hidden layer (these network types will be termed homogeneous networks) [24]. However, combining together different neuron types in the hidden layer affects the generalization performance of the network. The motivation of this research is to find the appropriate combination of hidden layer units in acceptable time that will have better generalization performance than their equivalent homogeneous ones for each regression or classification problem.

Since the invention of the ELM algorithm by Huang et al. [31, 32], a significant number of ELM-based approaches have been proposed with purpose to solve ELM's problems. A large number of these approaches are hybridizations with evolutionary algorithms aiming to optimize the hidden layer weights and thresholds.

The evolutionary ELM (E-ELM) method proposed by Zhu et al. [72] utilizes differential evolution (DE) for the optimization of the hidden layer weights and thresholds and the Moore-Penrose pseudo-inverse to analytically calculate the output weights. DE is known as one of the most efficient heuristic evolutionary based approaches for minimizing possibly non-linear and non-differentiable continuous space functions [56, 55, 72]. An extension of E-ELM for ordinal regression problems proposed by Sánchez-Monedero et al. [51]. This method uses a continuous weighted root mean square fitness function to guide the optimization process. The differential evolution ELM algorithm (DE-ELM) proposed by Bazi et al. [7] is an automatic method for classification of hyper-spectral images which also uses a DE to solve the model selection issue of ELM. Li et al. [36] proposed the tunable activation function ELM (TAF-ELM) algorithm. This method optimizes the hidden neurons transfer functions along with the hidden layer weights and thresholds altogether in one chromosome using a modified DE. The evolutionary selection ELM (ES-ELM) method for regression tasks proposed by Feng et al. [15] is based on the evolutionary algorithm and generates two ELM networks with L and $\frac{L}{2}$ hidden nodes accordingly. Then, it uses a natural selection strategy to make sure that the best hidden nodes will be selected for the next generation. The PSO-ELM approach by Xu and Shu [62] hybridizes ELM with the particle swarm optimization (PSO) algorithm in order to find the optimal combination of hidden layer weights and thresholds. PSO is an evolutionary based optimization algorithm proposed by Eberhart and Kennedy [13] which is inspired from the simulation of a simple social system (bird flocking or fish schooling). Similarly, the GSO-ELM algorithm by Silva et al.

[52] uses the group search optimization algorithm to select the ELM parameters. GSO was proposed by He et al. [21] and is also based in a simple social system (takes inspiration from the animal searching behaviour). These ELM variants employ an evolutionary algorithm mainly for the optimization of the hidden weights and thresholds. However, they lack a mechanism to create heterogeneous networks. An exception is the TAF-ELM approach with the use of the tunable parameter for the hidden units but it is unable to choose a suitable combination of transfer functions.

The Ho-HyELM approach proposed by Christou et al. [9] can create networks with different neuron types in the hidden layer but is restricted in the creation of homogeneous hidden layers.

Very few ELM-based approaches are able to create heterogeneous hidden layers. Optimally pruned ELM (OP-ELM) by Miche et al. [42] uses neurons with linear, sigmoid and Gaussian transfer functions to construct the heterogeneous layer. The construction is done by pruning of the least useful neurons from a large randomly created heterogeneous hidden layer. The ranking method used for the evaluation of the neurons is the multi-response sparse regression [53] algorithm. TROP-ELM proposed by Miche et al. [43] is an improvement of OP-ELM that uses a L_1 regularization penalty to rank the hidden layer neurons followed by a L_2 penalty on the regression weights for numerical stability. The synaptic kernel inverse method (SKIM) by Tapson et al. [59] for event-based systems redefines the hidden neurons as synaptic kernels in which the input event based signals are transformed into continuous-valued signals. The advanced ELM ensemble (AELME) from Abuassba et al. [1] constructs a network ensemble by training a randomly chosen ELM classifier on a subset of training data selected through random resampling. These methodologies work only with original units and they are not taking into consideration higher order units. Additionally, the SKIM methodology and AELME are ELM approaches that are used for spatio-temporal pattern recognition and classification problems respectively.

A series of ELM-based approaches is focused to improve the performance of ELM in sparse data sets. The unified ELM approach by Huang et al. [30] provides a unified framework which simplifies and combines different training methods, including support vector machine (SVM) [11] variants like least squares SVM (LS-SVM) [57] and proximal SVM (PSVM) [41]. Sparse ELM by Bai et al. [4], unifies different learning algorithms for classification, including SVM and radial basis function (RBF) networks. The main advantage of sparse ELM over unified ELM is its ability to create more compact networks which reduces storage space and testing time. Sparse Bayesian ELM (SBELM) for multi-class classification proposed by Luo et al. [40] estimates the marginal likelihood of network outputs and prunes most of the redundant hidden units during training, resulting in an accurate and compact model. These three methods work with original neurons only without taking into consideration higher order units.

A number of ELM-based approaches have been proposed to improve the transfer learning capability of ELM in cross domains. Zhang and Zhang [66] propose a unified framework named domain adaptation extreme learning machine (DAELM) with purpose to address the sensor drift issue which exhibits a non-linear dynamic property in electronic nose (E-nose) systems. This framework is able to learn a robust classifier utilizing a limited number of labelled target data for drift compensation as well as gas recognition in E-nose systems while retaining the speed and learning ability of ELM. This framework is comprised from two algorithms named source DAELM (DAELM-S) and target DAELM (DAELM-T). The former can learn a robust classifier on the source domain by leveraging a small number of labelled samples from the target domain. The latter can learn a classifier based on a small number of labelled data in target domain by leveraging a pre-learned base classifier in source domain. The source domain adaptation transfer ELM (TELM-SDA) and target domain adaptation transfer ELM (TELM-TDA) are proposed by Zhang and Zhang [67] for learning multi-domain tasks. The first algorithm is able to learn a classifier utilizing a small number of labelled instances while a large number of unlabelled data are exploited by approximating the prediction of the base classifier. The training of the base classifier is done in the source domain utilizing regularized ELM or support vector machine (SVM). On the other hand, TELM-SDA can learn a classifier utilizing a large number of labelled instances from the source domain and a very small number of labelled instances from the target domain as regularization. Both algorithms can form a unified ELM framework which refers to two stages including random feature mapping and output weights training. The ELM-based Domain Adaptation (EDA) framework by Zhang and Zhang [68] addresses the visual knowledge adaptation problem. It is able to learn a network classifier and a category transformation at the same time by utilizing labelled source domain data, a small number of labelled target domain data and unlabelled target domain data. The EDA method is extended to a joint learning framework of multiple views for structural information sharing of multiple local features with different feature representations. These methods are focused in the improvement of ELM's transfer learning capability in cross domains and they work with original neurons only without taking into consideration higher order units.

Some ELM-based approaches have been proposed for specific problems. The local discriminant preservation projection based kernelized ELM (LDPP-based KELM) method proposed by Zhang et al. [65] is a voltammetric electronic tongue (E-Tongue) system. This system is used as a taste analysis tool in taste recognition problems. The evolutionary cost-sensitive ELM (ECSELM) approach by Zhang and Zhang [69] addresses the robustness of ELM in cost-sensitive learning tasks like face recognition based access control system, where the misclassification of an unauthorized person as an authorized one can have greater impact than the misclassification of an authorized person as an unauthorized one. The self-expression ELM (SE^{2}LM) method proposed by Zhang and Deng [64] is an (E-nose) system which combines gas sensor technology and artificial intelligence for abnormal odor detection. These methods work only in specific tasks without taking into consideration higher order units.

A series of other non-ELM approaches uses neural networks ensembles to construct networks with heterogeneous structure. Zhao et al. [70] combines neural network ensembles and multi-population swarm intelligence to construct the proposed improved neural network ensemble (INNE). The $\underline{a}$ccurate an$\underline{d}$ $\underline{d}$iverse $\underline{e}$nsemble-$\underline{m}$aker giving $\underline{u}$nited $\underline{p}$redictions (ADDEMUP) algorithm from Opitz and Shavlik [48] creates ensemble networks by utilizing GAs to search for a correct and diverse population of neural networks to be used in the ensemble. Rosen [50] linearly combines outputs from individually trained networks using back-propagation to produce the output of the ensemble network. The evolutionary ensembles with negative correlation learning (EENCL) method from Liu et al. [39] utilizes an evolutionary algorithm based on evolutionary programming to search for a population of diverse individual neural networks that solve a problem together. It encourages different individual neural networks in the ensemble to learn different parts or aspects of the training data, so that the ensemble can learn better the entire training data. The GASEN method from Zhou et al. [71] assigns random weights to a set of neural networks and uses a GA to evolve these weights. Then it selects some neural networks based on the evolved weights to make up the ensemble. Finally, Bakker and Heskes [5] propose a method which summarizes large ensembles of models to a small number of representative models. These methodologies are focused in the creation of ensemble networks and work only with original units without taking into consideration higher order units.

This paper presents a novel ELM hybrid algorithm named heterogeneous hybrid extreme learning machine (He-HyELM) which is able to find the optimal heterogeneous SLNN for each specific problem type. The algorithm creates a pool of custom created neurons and utilizes them to construct a series of ELM trained homogeneous networks. These networks are evolved into heterogeneous networks utilizing a genetic algorithm (GA) with uniform crossover and the most optimal one is selected according to a fitness criterion. The GA was selected for the construction of the heterogeneous SLNNs, because it is straightforward to apply in this case, and forms ELM hybrids with improved generalization ability as seen from the numerous GA-based ELM hybrid approaches described above.

According to the experimental analysis, the He-HyELM created networks managed to achieve better generalization ability than homogeneous networks and heterogeneous networks created using optimally pruned extreme learning machine (OP-ELM). The challenging task for the creation of heterogeneous networks is to find the proper combination of hidden units in acceptable time since the search space rises exponentially with each neuron added to the hidden layer. These types of networks have the advantage of better generalization in comparison with traditional ELM in both regression and classification datasets as seen in Section 4, but they require more processing power for their creation.

This paper has been structured into 6 main sections, starting with Section 1 (Introduction) containing a brief explanation of the ELM algorithm, a literature review of the related work and a small description of the motivation and challenges of the proposed He-HyELM algorithm. Section 2 (Related Work) contains a detailed explanation of ELM and GA while Section 3 (The He-HyELM architecture), contains an in depth description of He-HyELM's architecture. The following section contains the experimental results in 2 different problem types (regression and classification). Finally, the last 2 sections contain the discussion and conclusion of the proposed method.

2. Related work

2.1. ELM

This section describes the ELM algorithm, which was originally proposed for shallow (one hidden layer) networks and later has been extended where the hidden layer need not be neuron-like [30, 27, 28].

In conventional neural network training methods, single feed-forward neural networks (SFLNNs) with additive or radial basis functions (RBFs) work as universal approximators when all networks parameters are adjusted [29]. Huang et al. [29] proved using an incremental construction method that in order to employ SLFNNs as universal approximators, the hidden layer nodes can be chosen randomly and need not to be tuned. Then the output node weights can be determined utilizing the Moore-Penrose pseudo-inverse. In this network type, the transfer functions for additive nodes can be any bounded non-constant piecewise continuous function $g : R \to R$, and the transfer functions for RBF nodes can be any integrable piecewise continuous function $g : R \to R$ and $\int_R g(x)dx \neq 0$ [27, 29].

A standard SLNN having:

- An input matrix $x = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \dots & \vdots \\ x_{N1} & \dots & x_{Nn} \end{bmatrix}_{N \times n}$
- A hidden weight matrix $w = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \vdots & \dots & \vdots \\ w_{h1} & \dots & w_{hn} \end{bmatrix}_{h \times n}$
- A threshold vector $\theta = [\theta_1, \dots, \theta_h]$
- An output nodes weight matrix $\beta = \begin{bmatrix} \beta_{11} & \dots & \beta_{1m} \\ \vdots & \dots & \vdots \\ \beta_{h1} & \dots & \beta_{hm} \end{bmatrix}_{h \times m}$

where $N \in \mathbb{N}^*$ is the number of input samples, $n \in \mathbb{N}^*$ is the number of inputs/features, $h \in \mathbb{N}^*$ is the number of hidden neurons, $m \in \mathbb{N}^*$ is the number of outputs/output neurons and $g(u)$ is the transfer function, can be mathematically modelled as:

$$\sum_{i=1}^{h}[\beta_{i1},\dots,\beta_{im}]g\left([w_{i1}\dots w_{in}]\begin{bmatrix}x_{j1}\\ \vdots \\ x_{jn}\end{bmatrix}+\theta_i\right), j=1,\dots,N \tag{1}$$

This network has the identity activation function ($g(u) = u$) and no threshold in the output nodes. These types of networks can be trained with the ELM algorithm which is illustrated in Algorithm 1.

Algorithm 1 ELM

```
$1 : w = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \vdots & \dots & \vdots \\ w_{h1} & \dots & w_{hn}\end{bmatrix}_{h\times n}$
Randomise the hidden layer weights.
$2 : \theta = [\theta_1, \dots, \theta_h]$
Randomise the hidden layer thresholds.
$3 : H = \begin{bmatrix} g\left([w_{11}\dots w_{1n}]\begin{bmatrix}x_{11}\\ \vdots \\ x_{1n}\end{bmatrix}+\theta_1\right) & \dots & g\left([w_{h1}\dots w_{hn}]\begin{bmatrix}x_{11}\\ \vdots \\ x_{1n}\end{bmatrix}+\theta_h\right) \\ \vdots & \dots & \vdots \\ g\left([w_{11}\dots w_{1n}]\begin{bmatrix}x_{N1}\\ \vdots \\ x_{Nn}\end{bmatrix}+\theta_1\right) & \dots & g\left([w_{h1}\dots w_{hn}]\begin{bmatrix}x_{N1}\\ \vdots \\ x_{Nn}\end{bmatrix}+\theta_h\right)\end{bmatrix}_{N\times h}$
Calculate the hidden layer matrix $H$.
$4 : T = \begin{bmatrix} t_{11} & \dots & t_{1m} \\ \vdots & \dots & \vdots \\ t_{N1} & \dots & t_{Nm}\end{bmatrix}_{N\times m}$
Calculate the target output matrix $T$.
$5 : \beta = H^{\dagger}T$
Calculate the output weights matrix $\beta$.
```

In line 1, it randomizes the hidden layer weight matrix w where each row contains the input weights of each hidden neuron. In line 2, it randomizes the threshold matrix θ where each entry to the matrix contains the threshold value of each hidden unit. In line 3, it calculates the hidden layer matrix H [26, 25]. The rows of the hidden layer matrix contain the output data of every hidden node. Each row of H contains the data for every training pattern introduced to the network. For this reason, the dimension of H is N (which is the number of the training patterns) multiplied by h (which is the number of output nodes). Line 4 involves the calculation of the target output matrix T which contains the expected values of the training set. Each row of T contains the corresponding expected values for each training pattern. In the fifth and final step, the algorithm calculates the output weights. The equation $\beta = H^{\dagger}T$ where $H^{\dagger}$ is the Moore-Penrose pseudo-inverse derives from the equation $T = H\beta$ which describes the relationship of the hidden layer and the output weights matrix with the target matrix [31, 32]. The output weights matrix β has dimension $h \times m$ where each column contains the weight values of each output node.

The parameters and the notation used for describing the ELM algorithm can be summarized in Table 1.

Table 1: ELM Parameter Settings

Parameter Name	Symbol	Values/Types
No of Hidden Neurons	h	$h \in \mathbb{N}^*$
No of Input Samples	N	$N \in \mathbb{N}^*$
No of Inputs/Features	n	$n \in \mathbb{N}^*$
No of Outputs/Output Neurons	m	$m \in \mathbb{N}^*$
Input Matrix	x	$x \in \mathbb{R}^{N \times n}, (N, n) \in \mathbb{N}^*$
Weight Matrix	w	$w \in [-1, 1]^{h \times n}, (h, n) \in \mathbb{N}^*$
Thresholds	θ	$\theta \in [-1, 1]^{h}, h \in \mathbb{N}^*$
Hidden Layer Output Matrix	H	$H \in \mathbb{R}^{N \times h}, (N, h) \in \mathbb{N}^*$
Activation Function	g	$g \in \mathbb{R}$
Target Output Matrix	T	$T \in \mathbb{R}^{N \times m}, (N, m) \in \mathbb{N}^*$
Output Weight Matrix	β	$\beta \in \mathbb{R}^{h \times m}, (h, m) \in \mathbb{N}^*$

2.2. GA

This section presents the generic structure of a GA. In organisms, the chromosomes can be arrayed in pairs or can be unpaired. The first ones are called diploids while the latter haploids. The structure of a GA that artificially represents the reproduction process for a haploid organism can be seen in Fig. 1. The GA is an iterative procedure that starts with an initial fixed set or pool of candidate solutions called population. A candidate solution is called chromosome and represents a possible solution to the problem which is usually encoded as bit strings (binary encoding scheme).

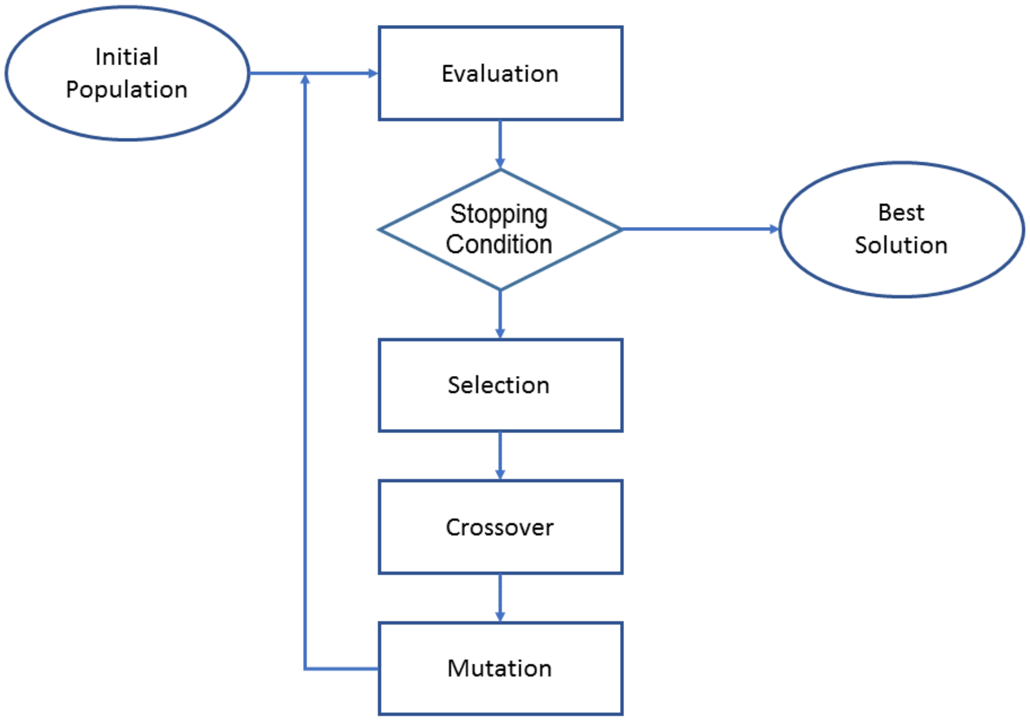

Figure 1: Flowchart of a typical GA. The GA starts with an initial pool of candidate solutions called population. Each iteration step selects the best solution according to a fitness criterion. The crossover operator involves the partial exchange of information between two parent chromosomes and the mutation operator alters one randomly chosen chromosome. The process is continued until a termination criterion is reached.

In a binary encoding scheme every chromosome is a string of bits, 0 or 1 with length L as seen in Table 2. An allele in a bit string is represented by 0 or 1 and represents the alternative forms of a gene. The variables for the problem that is going to be solved are discretized in a priori fashion and the range of the discretization corresponds to a power of 2. In the chromosomes of Table 2, it is possible to represent $2^{20} = 1048576$ discrete values. This type of encoding is acceptable and doesn't lead to any specific problems, if the parameters are continuous and the discretization provides enough level of accuracy for the desired output [45, 61].

Table 2: Binary Encoding Scheme

Chromosome A:	10101000011001101100
Chromosome B:	11000111001101010010

As seen in Algorithm 2, the GA starts with the creation of the initial population, which is usually chosen randomly or by using heuristic construction. Once the initial population has been defined, the evolution process begins (line 2), which involves a series of iterative steps. Each iteration of the algorithm is called generation and involves 4 genetic operators (evaluation, selection, crossover, mutation). In line 3, the GA evaluates the population according to the fitness value of each individual. Fitter chromosomes have more chances to be selected and the average fitness of the population is expected to grow with successive generations. The next step involves checking the stopping criteria of the algorithm. The GA stops the evolution process if the stopping criteria have been met (e.g. the maximum number of generations has been reached) or when it converges to the best chromosome which represents the optimal or a suboptimal solution of the problem. If these criteria have been met, the algorithm (line 5) selects the best solution and returns the outcome (line 6) [45].

The next genetic operator is the selection mechanism (line 8) which usually resides in the selection of fitter chromosomes, with higher probability, in the evolution process. The roulette wheel selection is one of the most straightforward implementations of this strategy. This method adjusts the selection probability of each chromosome according to it's fitness value. The formula which defines each chromosome's probability can be seen below.

$$P_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{2}$$

In this formula, N is the number of chromosomes, each defined from it's fitness $f_i > 0$ $(i = 1, 2, \ldots, N)$. This method mimics a roulette wheel with sectors of size proportional to f_i $(i = 1, 2, \ldots, N)$. The selection procedure of a chromosome is done by choosing a random point on the wheel and locating the corresponding sector. When a simple search is utilized, such a location requires $O(N)$ complexity while the binary search requires $O(logN)$ complexity [37, 22].

GAs explore the search space by using two different operations named crossover and mutation. Crossover (line 9) is a local search operator and can be viewed as the procedure of constructing the next population after constructing the intermediate population using selection. Crossover is applied to randomly paired parent chromosomes which are recombined according to various methods [61]. Single-point crossover is one of the simplest and most popular crossover operators. It initially selects the parents and uses a random selected crossover point. This point is used for mutual exchange of the parent information and produces two offspring as seen in Table 3 [23, 60]. In this example the crossover point is selected between the fourth and fifth gene.

Table 3: Single-Point Crossover

Parent A:	1010|10100
Parent B:	0011|11011
Offspring A:	1010|11011
Offspring B:	0011|10100

Mutation (line 9) is a simple search operator which allows the GA to escape from the local minimum trap. It is a global search operator used for searching the entire search space which introduces genetic diversity to the evolution process. For binary encoding, mutation can be implemented by simply changing the value of each gene (from 0 to 1 and 1 to 0) according to a small probability, which is usually proportional to the chromosome length. The mutation probability defines how many genes of the chromosome are going to be changed and must be selected very carefully. In general, this probability is low because high values of the mutation operator transform the GA to random search [54].

The new generation of chromosomes is then created according to the fitness measure by selecting from either the combined pool of parents and offspring or the offspring pool. Fitter chromosomes have more chances to be selected and the average fitness of the population is expected to grow with successive generations. The process is continued until the termination criterion is reached or until the algorithm converges to the best chromosome which represents the optimal or a suboptimal solution of the problem [45].

Algorithm 2 GA

```
1 : $create\ Pop$
Create the initial population.
2 : **Loop**
Start the evolution process.
3 : $Pop_{evaluate} = evaluate(Pop)$
Evaluate the population.
4 :     **If** $(stop = true)$ or $(solution_{best} = true)$
        Check if stopping criteria have been met or if the best solution has
        been found.
5 :         $solution_{best} \leftarrow best(Pop_{evaluate})$
            Select the optimal solution.
6 :         **Return** $solution_{best}$
            Stop the evolution process and return the optimal solution.
7 :     **End If**
8 :     $Pop_{select} \leftarrow select(Pop_{evaluate})$
        Select the chromosomes for reproduction.
9 :     $Pop_{crossover} \leftarrow crossover(Pop_{select})$
        Reproduce the selected population.
10 :    $Pop \leftarrow mutation(Pop_{crossover})$
        Mutate the offspring.
11 :**End Loop**
```

3. The He-HyELM architecture

3.1. He-HyELM structural units

In this section, a structured view of the custom neurons, that are utilized in this research, is presented. It divides neurons into three subcomponents (the last two form the neuron's soma S) according to the structured composite model (C-Model) proposed by Christou et al. [9]. These neuron subcomponents are the dendrite (D), the activation (S_a) function and the activation-output (S_{ao}) function. In this model, the subscript in S takes the values a and ao with purpose to distinguish between the neuron's soma activation function (S_a) and the neuron's soma activation-output function (S_{ao}).

In machine learning, feature vectors are used to represent numeric or symbolic characteristics called features of an object in a mathematical way [38]. The vector space, which is associated with these vectors is called feature space. The feature space in this research contains all artificial neural network input signals. These input signals are weighted and they are named dendrites. The summation (along with an optional threshold for the original neuron types) provides the neuron's soma activation function. The soma activation-output function or transfer function takes as input the activation function and produces the neuron's output. A neuron is depicted as a 3-tuple ($< D, S_a, S_{ao} >$) and contains three different building blocks: the dendrite, the activation function and the activation-output function. These building blocks form a composite function and produce the neuron's output $y = g(u) = S_{ao}(S_a(D))$ [9].

3.2. Varying the dendrites

The weighted input vectors form the neuron's dendrite which is summed by the activation function and passed as input to the activation-output function. This study used two different types of dendrites, the linear (D^l) dendrite and a variation of the cubic (D^c) dendrite called multi-cube (D^{mc}) dendrite [9] which are going to be analysed in Sections 3.2.1 and 3.2.2. The distinction between different types of dendrites is done utilizing the l, c and mc superscripts which define the linear, cubic and multi-cubic types accordingly.

3.2.1. Linear dendrites

The linear configuration is depicted in Fig. 2 [20] where each input participates equally to the neuron's activation. In this dendritic type, the inputs are modelled as the input vector $x = [x_1, x_2, \ldots, x_n], n \in \mathbb{N}^*, x \in \mathbb{R}$ and they are multiplied element-wise with the weight vector $w^l = [w_1, w_2, \ldots, w_n], n \in \mathbb{N}^*, w^l \in \mathbb{R}$ as seen in equation (3) [9].

$$D^l = [w_1 x_1, w_2 x_2, \ldots w_n x_n] \tag{3}$$

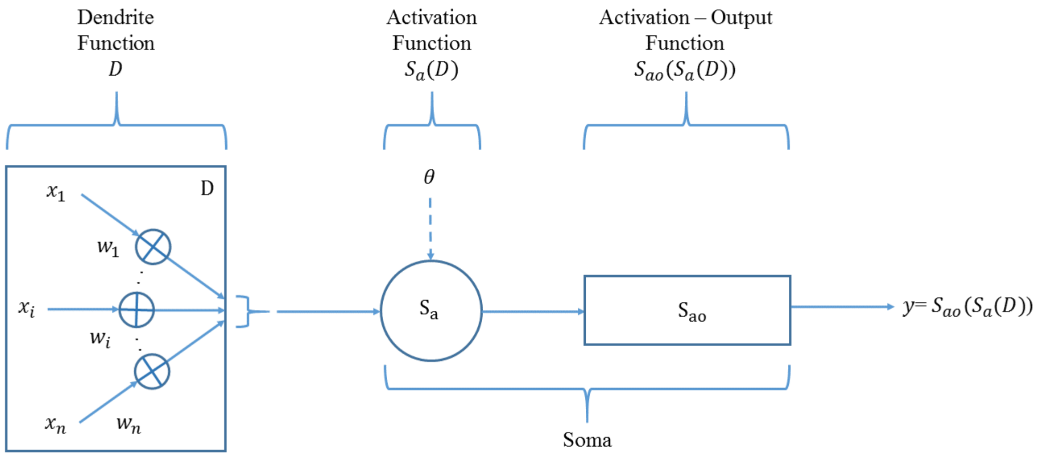

Figure 2: Linear neuron structure. The linear dendrite multiples element-wise the input vector with the weight vector and sends as input to the activation function, the vector containing the outcome of this multiplication. The 'sum-of-weights' activation function sums together all dendrite elements and adds an optional threshold θ depicted by a dashed arrow. Finally, the output of the activation function is sent to the activation-output function to produce the neuron's output [9].

3.2.2. Cubic dendrites

The main difference of the higher order neuron model units [2] [10, 16, 18, 19, 46, 47] from the linear units is that they can receive a multi-dimensional cube as input vector. Also, the number of neuron weights w^c_{no} is not proportional to the number of neuron inputs but increases exponentially according to the number of inputs ($w^c_{no} = 2^n, n \in \mathbb{N}^*$ where n is the number of inputs). In the higher order neurons dendrite, each input participates to the neuron's activation function according to a probability and each weight participates to the activation function according to the equation calculated in (4).

$$P_\mu = \frac{1}{2^n}\prod_{i=1}^{n}(1+\mu_i\frac{x_i}{x_{max}}) \tag{4}$$

The above formula is used to calculate the probability of each real-valued weight. The term $\frac{1}{x_{max}}$ normalizes the input to the range $[-1, 1]$. The μ_i is the i^{th} component of the n-bit cube site address string $\mu = \mu_1\mu_2\ldots\mu_i\ldots\mu_n$. This bit string takes only the values -1 and 1. Finally, with n we define the

[2] These types of neurons are termed cubic neurons or cubic units in this paper.

number of inputs [19, 9].

The higher order dendritic type is depicted in equation (5) and utilizes the weight vector $w^c_\mu = [w_1, w_2, \ldots, w_{2^n}], n \in \mathbb{N}^*$.

$$D^c = w^c_\mu \frac{1}{w^c_{max}} \prod_{i=1}^{n} (1 + \mu_i \frac{x_i}{x_{max}}) \tag{5}$$

The factor $\frac{1}{w^c_{max}}$ normalises the weights to the range of $[-1, 1]$ (as w^c_{max} is defined the maximum value from the weight vector w^c_μ) [19, 9]. A diagram presenting the structure of the 1 to n input cubic neuron is depicted in Fig. 3.

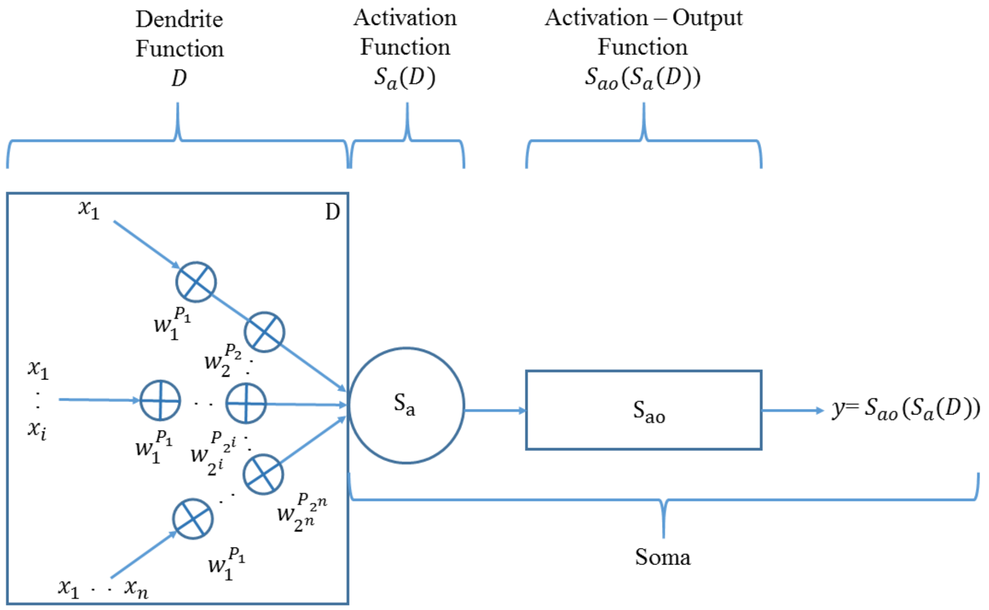

Figure 3: Cubic unit structure. This diagram depicts the structure of the 1 to n input cubic neuron. Initially the dendrite creates a vector containing $2^n, n \in \mathbb{N}^*$ elements where n is the number of inputs. This vector is created according to equation (5) and is sent as input to the activation function. The 'sum-of-weights' activation function sums together all dendrite elements and the output of the activation function is sent to the activation-output function to produce the neuron's output [9].

A significant issue of these cubic neurons is that the number of input weights rises exponentially according to the number of inputs which can cause performance problems even with relatively small number of inputs (e.g. a linear unit with 10 inputs requires 10 weights while a cubic unit requires $2^{10} = 1024$ weights). In order to circumvent this issue, Gurney [19] proposed the multi-cube unit. In the multi-cube unit instead of having one high dimension cube as input vector, we use several lower dimension cubes (sub-cubes) which do not necessary share the same dimension. The formula for the dendritic type which takes as input multiple cubes having the same dimension is given in (6). In this formula n is the number of inputs for each cube and q is the total number of cubes.

$$D^{mc} = \Bigg[w^{mc}_{\mu,1} \frac{1}{w^{mc}_{max}} \prod_{i=1}^{n} (1 + \mu_i \frac{x_{i,1}}{x_{max}}), \\ w^{mc}_{\mu,2} \frac{1}{w^{mc}_{max}} \prod_{i=1}^{n} (1 + \mu_i \frac{x_{i,2}}{x_{max}}), \ldots, w^{mc}_{\mu,q} \frac{1}{w^{mc}_{max}} \prod_{i=1}^{n} (1 + \mu_i \frac{x_{i,q}}{x_{max}}) \Bigg] \tag{6}$$

The multi-cube unit structure with the same dimension in all cubes which was used in the experimental part of this research can be seen in Fig. 4.

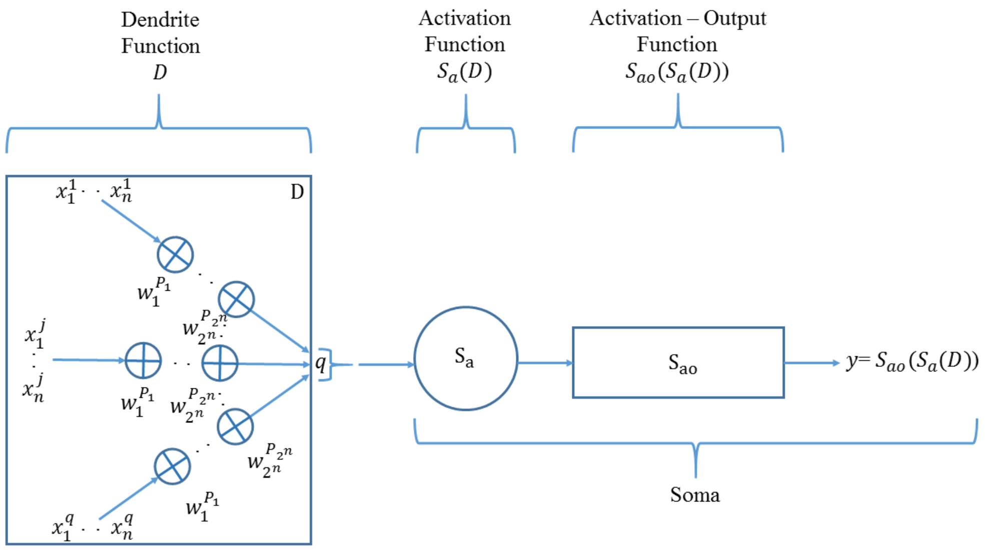
Dendrite Function D
Activation Function $S_a(D)$
Activation – Output Function $S_{ao}(S_a(D))$
D
$x_1^1 \cdot \cdot x_n^1$
$w_1^{P_1}$
$w_{2^n}^{P_{2^n}}$
x_1^j
x_n^j
q
S_a
S_{ao}
$y = S_{ao}(S_a(D))$
$x_1^q \cdot \cdot x_n^q$
Soma

Figure 4: Multi-cube unit structure with the same dimension in all cubes. This diagram depicts a multi-cube unit containing q number of n-dimensional cubes. Initially the dendrite creates a vector containing $q \cdot n, (q, n) \in \mathbb{N}^*$ elements where n is the number of inputs and q is the number of multi-cube units. This vector is created according to equation (6) and is sent as input to the activation function (the superscript in each input value denotes the current sub-cube). The 'sum-of-weights' activation function sums together all dendrite elements and the output of the activation function is sent to the activation-output function to produce the neuron's output.

3.3. Varying the activation function.

This paper utilizes the 'sum of weights' activation function[3] shown in equation (7) which it's main purpose is to aggregate the dendrite elements. For linear dendrites besides aggregating, it also adds an optional threshold θ [20, 9]. The distinction between the linear, cubic and multi-cubic types of activation functions is done utilizing the l, c and mc subscripts as seen in the following three formulas.

$$S_{\alpha_l}^{SoW} = \sum_{i=1}^{n} x_i w_i + \theta \tag{7}$$

The 'sum of weights' activation function for the cubic units is shown in (8) [19, 9].

$$S_{\alpha_c}^{SoW} = \frac{1}{w_{max}^{c} 2^n} \sum_{\mu=1}^{2^n} w_\mu^c \prod_{i=1}^{n} (1 + \mu_i \frac{x_i}{x_{max}}) \tag{8}$$

The 'sum of weights' activation function for the multi-cube units having the same dimension is shown in (9) which is derived from the sum of the multi-cube dendrite's matrix elements, $S_{\alpha_{mc}}^{SoW} = \sum_{j=1}^{q} \sum_{\mu=1}^{2^n} D_\mu^c = \sum_{j=1}^{q} \sum_{\mu=1}^{2^n} w_\mu^c \frac{1}{w_{max}^c 2^n} \prod_{i=1}^{n} (1+\mu_i \frac{x_i}{x_{max}}) = \frac{1}{w_{max}^c 2^n} \sum_{j=1}^{q} \sum_{\mu=1}^{2^n} w_\mu^c \prod_{i=1}^{n} (1+\mu_i \frac{x_i}{x_{max}})$ [19].

$$S_{\alpha_{mc}}^{SoW} = \frac{1}{w_{max}^{mc} 2^n} \sum_{j=1}^{q} \sum_{\mu=1}^{2^n} w_\mu^{mc} \prod_{i=1}^{n} (1 + \mu_i \frac{x_i}{x_{max}}) \tag{9}$$

3.4. Varying the activation-output function.

The activation-output function is responsible for the neuron's output. The following six activation functions were used during the experimental part of this research and utilize the $\rho = \{0.1, 0.2, \ldots, 1\}$ parameter to change their shape. The distinction between different types of activation-output functions is done utilizing the BI, G, HT, Sig, Sin, and $SSign$ superscripts.

[3] This specific type of activation function is depicted with the SoW superscript in the activation function's notation (S_α^{SoW}).

- Bent Identity: $S_{ao}^{BI} = \rho\Big(\frac{\sqrt{{S_a}^2+1}}{2} + S_a\Big)$.
- Gaussian: $S_{ao}^{G} = e^{-\Big(\frac{S_a}{\rho}\Big)^2}$.
- Hyperbolic Tangent: $S_{ao}^{HT} = tanh\Big(\frac{S_a}{\rho}\Big)$.
- Sigmoid: $S_{ao}^{Sig} = \frac{1}{1+e^{-\frac{S_a}{\rho}}}$.
- Sinusoid: $S_{ao}^{Sin} = sin\Big(\frac{S_a}{\rho}\Big)$.
- Soft Sign: $S_{ao}^{SSign} = \frac{\frac{S_a}{\rho}}{1+|\frac{S_a}{\rho}|}$.

3.5. He-HyELM

The He-HyELM algorithm contains two phases. The first phase creates a series of homogeneous networks while the second phase selects the best ones for the evolution process that will result in the creation of heterogeneous networks. The first phase creates three neuron pools containing different number of dendrites, activation functions and activation-output functions. The experimental part of this paper used the Pl_D, Pl_a and Pl_{ao} neuron sub-components pools (we used the Pl symbol to define each pool and a specific subscript to distinguish between different types of pools). These pools contain two dendrites (linear , multi-cube), the 'sum of weights' activation function for linear and multi-cube units and six activation-output functions ($S_{ao}^{BI}, S_{ao}^{G}, S_{ao}^{HT}, S_{ao}^{Sig}, S_{ao}^{Sin}, S_{ao}^{SSign}$) accordingly.

$$Pl_D = \left\{ \begin{aligned} &[w_1x_1, w_2x_2, \dots w_nx_n], \Bigg[w_{\mu,1}^{mc}\frac{1}{w_{max}^{mc}}\prod_{i=1}^{n}(1+\mu_i\frac{x_{i,1}}{x_{max}}), \\ &w_{\mu,2}^{mc}\frac{1}{w_{max}^{mc}}\prod_{i=1}^{n}(1+\mu_i\frac{x_{i,2}}{x_{max}}), \dots, w_{\mu,q}^{mc}\frac{1}{w_{max}^{mc}}\prod_{i=1}^{n}(1+\mu_i\frac{x_{i,q}}{x_{max}})\Bigg] \end{aligned} \right\}$$

$$Pl_a = \left\{ \sum_{i=1}^{n} x_iw_i + \theta, \frac{1}{w_{max}^{mc}2^n}\sum_{j=1}^{q}\sum_{\mu=1}^{2^n} w_{\mu}^{mc}\prod_{i=1}^{n}(1+\mu_i\frac{x_i}{x_{max}}) \right\}$$

$$Pl_{ao} = \left\{\begin{matrix} \rho\Big(\frac{\sqrt{{S_a}^2+1}}{2}+S_a\Big), e^{-\left(\frac{S_a}{\rho}\right)^2}, tanh\Big(\frac{S_a}{\rho}\Big), \\ \frac{1}{1+e^{-\frac{S_a}{\rho}}}, sin\Big(\frac{S_a}{\rho}\Big), \frac{\frac{S_a}{\rho}}{1+|\frac{S_a}{\rho}|} \end{matrix}\right\}$$

The above structural units are combined together to form the custom created neurons pool seen below. The contents of this pool include all the possible neuron combinations of neuron subcomponents from the previous three pools which result in the creation of the custom neuron pool (Pl_{cn}) containing 12 neurons. The subscripts l and mc are used to distinguish between linear and multi-cube neuron types.

$$Pl_{cn} = \left\{\begin{matrix} S^{BI}_{ao_l}, S^{BI}_{ao_{mc}}, S^{G}_{ao_l}, S^{G}_{ao_{mc}}, S^{HT}_{ao_l}, S^{HT}_{ao_{mc}}, \\ S^{Sig}_{ao_l}, S^{Sig}_{ao_{mc}}, S^{Sin}_{ao_l}, S^{Sin}_{ao_{mc}}, S^{SSign}_{ao_l}, S^{SSign}_{ao_{mc}}, \end{matrix}\right\}$$

These custom neurons are used to create a pool of neural networks (Pl_{nets}) which will contain the same number of nodes in the hidden layer and will be trained using ELM. A SLNN containing h hidden nodes, m output nodes and the activation function $g(u)$ in the hidden layer nodes is mathematically modelled in as $\sum_{i=1}^{h}[\beta_{i1},\ldots,\beta_{im}]g(u)$. This network has the identity activation function ($g(u) = u$), no threshold in the output nodes and can be trained with the ELM algorithm. The contents of the neural network pool Pl_{nets} for the experimental part of this research can be seen below while the first phase of the He-HyELM algorithm can be seen in Fig. 5.

$$Pl_{nets} = \sum_{i=1}^{h}[\beta_{i1},\ldots,\beta_{im}]g(u), g(u) \in Pl_{cn}$$

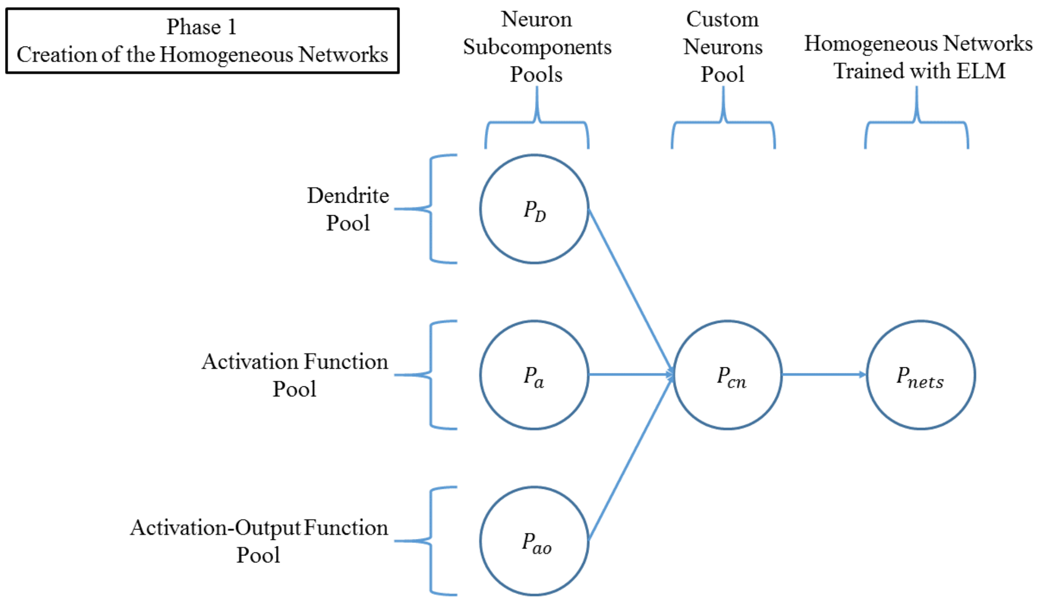
Phase 1
Creation of the Homogeneous Networks
Neuron Subcomponents Pools
Custom Neurons Pool
Homogeneous Networks Trained with ELM
Dendrite Pool
Activation Function Pool
Activation-Output Function Pool
P_D
P_a
P_{ao}
P_{cn}
P_{nets}

Figure 5: He-HyELM Structure (Phase 1). This diagram depicts the homogeneous networks creation phase of the He-HyELM algorithm. Initially, the algorithm utilises three different neuron subcomponents from the dendrite, activation function and activation-output function pools to form the custom neurons pool. Then it utilizes these neurons to create a series of SLNNs trained with ELM.

The second phase (as seen in Fig. 6) uses a GA for the evolution process and begins with the creation of the initial population. The initial population is created utilizing the ELM trained networks from the previous phase. After the creation of the initial population, the evolution process begins with the evaluation of the population followed by the selection, crossover and mutation procedures.

During the evaluation process the networks are trained with ELM and they are ranked from best to worst according to a fitness criterion.The fitness criterion uses the k-fold cross-validation method to calculate the fitness from all folds on training data. The He-HyELM method utilizes a test set containing unknown data at the end of the evolution process in order to avoid over-fitting. The selected statistic for regression problems is the average mean square error (MSE) over all folds. The MSE for each network is calculated on the training set during the evolution process and on the test set after the evolution process has completed where the network with lowest

MSE is selected as the most optimal.

$$MSE = \frac{1}{kp} \sum_{i=1}^{k} \Big(\sum_{j=1}^{p} (t_i^j - y_i^j)^2 \Big) \tag{10}$$

The above formula with k defines the number of folds, with p the number of testing patterns,with t_i^j the current pattern target network output value for the current fold and finally, with y_i^j the current pattern network output value for the current fold.

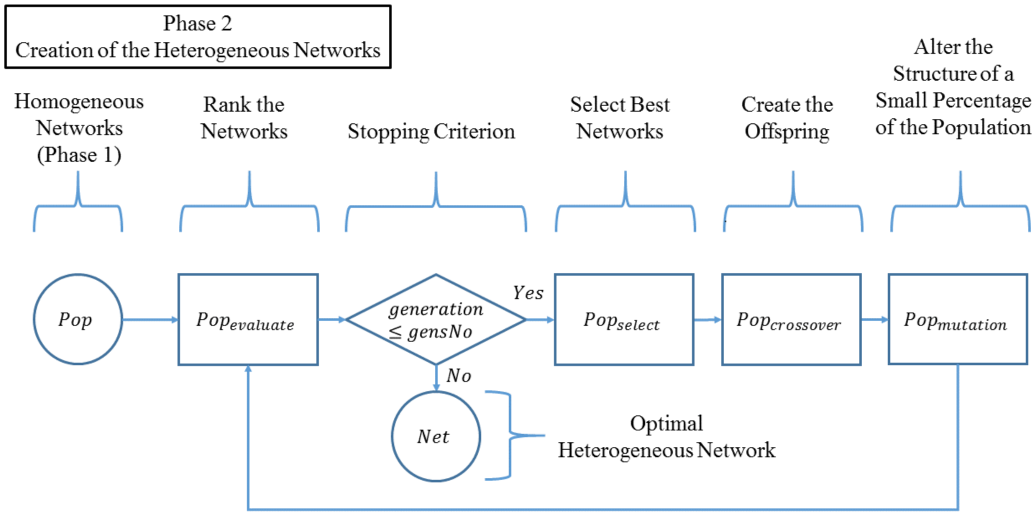

Figure 6: He-HyELM Structure (Phase 2). This diagram depicts the heterogeneous networks creation phase of the He-HyELM algorithm. The algorithm uses a GA to evolve the homogeneous networks created in the previous phase (they form the initial population) to heterogeneous networks. Initially, the algorithm ranks the networks from best to worst according to a fitness criterion. Then it checks if the maximum number of generations ($gensNo$) has been reached in order to stop the evolution process and select the best network according to fitness. If the maximum number of generations has not been reached, then selects the best networks for reproduction (according to fitness), creates the offspring and mutates a small percentage of the population. This procedure repeats until the maximum number of generations is reached.

The selected statistic for classification problems is the accuracy percentage over all folds shown in (11). The network acquiring the highest percentage is considered the optimum.

$$acc = \frac{1}{k}\sum_{i=1}^{k}\Big(1 - \frac{err}{p}\Big). \tag{11}$$

In the above formula k defines the number of folds, p is the number of testing patterns and finally, err is the number of misclassified test patterns for the current fold. After the evaluation process is finished, the GA checks if the maximum number of generations has been reached in order to terminate the evolution process. When the maximum number of generations is reached, the selection of the final optimal network is done using the proper fitness function on the test set. Alternatively, continues the evolution process with the selection process.

During the selection process, only 50% of the networks from the initial population and after each generation are selected. This is done in order to keep a fixed population number at each generation (the crossover operation described next doubles the population).

The crossover operation is aimed for local search between the best candidates and is done utilizing the uniform crossover operator. The uniform crossover operator exchanges information between the selected parent chromosomes by choosing a uniform random real number $u \in [0, 1]$ for each gene. It creates two offspring containing genes from both the parent chromosomes uniformly [60, 58].

In the mutation operation, a small percentage of the offspring (10%) will have one of its neurons replaced with a neuron taken from the neuron pool. This is done with purpose to have a global search amongst the search space.

After the selection, crossover and mutation procedures are finished, the generation completes, and the evolution process is repeated for a fixed number of generations.

3.6. The He-HyELM Algorithm

This section presents a detailed explanation of the proposed He-HyELM algorithm.

Algorithm 3 Heterogeneous Hybrid ELM Algorithm (Phase 1)

```
$1 : Pl_D = \left\{ [w_1x_1, w_2x_2, \dots w_nx_n], \left[ w^{mc}_{\mu,1}\frac{1}{w^{mc}_{max}}\prod_{i=1}^{n}(1+\mu_i\frac{x_{i,1}}{x_{max}}), w^{mc}_{\mu,2}\frac{1}{w^{mc}_{max}}\prod_{i=1}^{n}(1+\mu_i\frac{x_{i,2}}{x_{max}}), \dots, w^{mc}_{\mu,q}\frac{1}{w^{mc}_{max}}\prod_{i=1}^{n}(1+\mu_i\frac{x_{i,q}}{x_{max}}) \right] \right\}$
Dendrite pool.
$2 : Pl_a = \left\{ \sum_{i=1}^{n} x_iw_i + \theta, \frac{1}{w^{mc}_{max}2^n}\sum_{j=1}^{q}\sum_{\mu=1}^{2^n} w^{mc}_{\mu}\prod_{i=1}^{n}(1+\mu_i\frac{x_i}{x_{max}}) \right\}$
Activation functions pool.
$3 : Pl_{ao} = \left\{ \rho\Big(\frac{\sqrt{{S_a}^2+1}}{2}+S_a\Big), e^{-\left(\frac{S_a}{\rho}\right)^2}, tanh\Big(\frac{S_a}{\rho}\Big), \frac{1}{1+e^{-\frac{S_a}{\rho}}}, sin\Big(\frac{S_a}{\rho}\Big), \frac{\frac{S_a}{\rho}}{1+|\frac{S_a}{\rho}|} \right\}$
Activation-output functions pool.
$4 : Pl_{cn} = \left\{ S^{BI}_{ao_l}, S^{BI}_{ao_{mc}}, S^{G}_{ao_l}, S^{G}_{ao_{mc}}, S^{HT}_{ao_l}, S^{HT}_{ao_{mc}}, S^{Sig}_{ao_l}, S^{Sig}_{ao_{mc}}, S^{Sin}_{ao_l}, S^{Sin}_{ao_{mc}}, S^{SSign}_{ao_l}, S^{SSign}_{ao_{mc}}, \right\}$
Custom neurons pool.
$5 : Pl_{nets} = \sum_{i=1}^{h}[\beta_{i1}, \dots, \beta_{im}]g(u), g(u) \in Pl_{cn}$
Create the homogeneous SLNNs.
$6 : train(Pl_{nets})$
Train using ELM the homogeneous SLNNs.
```

In the first phase of the He-HyELM algorithm which involves the creation of a series of homogeneous networks with fixed number of hidden units, the algorithm starts with the initialization of the three neuron subcomponents pools in lines 1, 2 and 3. Then, creates the custom neurons pool (line 4), utilizing the neuron subcomponents taken from the previous pools. In line 5, it creates a series of homogeneous neural networks while in line 6 trains them using the ELM algorithm. The first 6 lines of the algorithm complete the homogeneous networks creation phase as seen in Algorithm 3.

Algorithm 4 Heterogeneous Hybrid ELM Algorithm (Phase 2)

```
1 : $create\ Pop$
Create the initial population.
2 : Loop
Start the evolution process.
3 : $generation \leftarrow generation + 1$
Increase the number of generations.
4 :     If $(generation = gensNo)$
        Check for maximum number of generations.
5 :         $Net \leftarrow best(Pop)$
            Select the optimal network.
6 :         Return $Net$
            Stop the evolution process and return the optimal network.
7 :     End If
8 :     $Pop_{select} \leftarrow \frac{Pop}{2}$
        Select 50% of the networks.
9 :     $Pop_{crossover} \leftarrow crossover(Pop_{select})$
        Reproduce the selected population.
10 :    $Pop \leftarrow mutation(Pop_{crossover})$
        Mutate the offspring.
11 :End Loop
```

The second phase involves the creation of the heterogeneous networks utilizing a GA and starts with the creation of the initial population. This process (line 1) utilizes all the created networks. Line 2 begins the evolution process. Line 3 increases the *generation* counter. Line 4 checks if the maximum allowed number of generations has been achieved and then in line 5 it selects the optimal heterogeneous network utilizing the test set. In line 6, returns the selected optimal network. Line 8 selects 50% of the networks with the best fitness for the reproduction process utilizing the training set. Line 9 reproduces the selected population while line 10 mutates the created offspring.

4. Simulation results

In this section, we compare the performance of He-HyELM with four existing algorithms (classic ELM, Ho-HyELM, OP-ELM for homogeneous networks and OP-ELM for heterogeneous networks) using ten regression and six classification problems. The regression problems include five function approximation problems (Ackley, Beale, Goldstein-Price, Holder Table $(x_1^2 - x_2^2)sin(0.5x_1))$ and five real world regression datasets ('airfoil self-noise', 'auto MPG', 'concrete compressive strength' [63], 'servo', 'yacht hydrodynamics'). The classification problems include six real world datasets ('banknote authentication', 'breast cancer Wisconsin', 'cryotherapy' [34, 35], 'diabetic retinopathy Debrecen' [2], 'HIV-1 protease cleavage' [49] and 'mammographic mass' [14]). These real world datasets were taken from the UCI machine learning repository [12] and we normalized the inputs and weights for every experiment run. Regarding the neuron subcomponents, we used original and multi-cube dendrites, the 'sum of the weights' activation function and six tunable activation-output functions (bent identity, Gaussian, hyperbolic tangent, sigmoid, sinusoid, soft sign). We dealt the missing values problem of the 'mammographic mass', 'auto MPG' and 'breast cancer Wisconsin' datasets by replacing them with the average values taken from the available data for the first two datasets while for the 'breast cancer Wisconsin' dataset we removed the entries contained those values (16 out of 699).

4.1. Parameter details

All the experiments were run in MATLAB 2017a environment utilizing the parameters shown in Table 4. The function approximation problems contained 150 randomly generated samples taken from $[-15, 15]$ interval. The above interval was divided into 30 equal sized subintervals and we took 5 random samples from each subinterval with purpose to have even distribution of the samples across the interval. The initialization of the hidden weights and thresholds was done by taking values from the $[-1, 1]$ interval. The dimension of each sub-cube was set to 1 which resulted utilizing n sub-cubes and $n2^1 = 2n$ weights in total. We also used a fixed number (15) for the hidden layer nodes and each activation-output function was tuned using the parameter $\rho = \{0.1, 0.2, \ldots, 1\}$. The GA was run for 10 generations using the uniform crossover and a fixed 10% mutation rate. For the validation method, 5-fold cross validation was used (80% for training and 20% for testing). Finally, each experiment was repeated 10 times having different values for the weights and thresholds.

Table 4: He-HyELM Parameter Settings

Parameter Name	Symbol	Values/Types
Linear Unit Weights	w^l	$w^l \in [-1,1]^n, n \in \mathbb{N}^*$
Multi-Cube Unit Weights	w^{mc}	$w^{mc} \in [-1,1]^{2n}, n \in \mathbb{N}^*$
Threshold	θ	$\theta \in [-1,1]$
Inputs	x	$x \in [-15,15]^n, n \in \mathbb{N}^*$
Number of Hidden Layer Nodes	h	15
Tuning Parameter	ρ	$\{0.1, 0.2, \dots, 1\}$
Generations	$gensNo$	10
Crossover	$crossover$	$uniform$
Mutation Rate	mr	10%
Number of Folds	k	5
Experiment Sets	$expNo$	10

4.2. Regression problems

The first function approximation experiment is the approximation of the Ackley function [3] depicted in formula (12). The characteristics of the *Ackley* multimodal two-dimensional (2D) function include the existence of one global minimum at $x^* = f(0,0), f(x^*) = 0$ and multiple local minimums.

$$\begin{aligned} f_{Ackley}(x_1, x_2) = &- 20exp\left[- 0.2\sqrt{0.5(x_1^2 + x_2^2)} \right] - \\ &exp[0.5(cos2\pi x_1 + cos2\pi x_2)] + e + 20 \end{aligned} \tag{12}$$

The second experiment is the approximation of the *Beale* function depicted in formula (13). The characteristic of the *Beale* multimodal 2D function is the existence of one global minimum at $x^* = (3, 0.5), f(x^*) = 0$ [33].

$$\begin{aligned} f_{Beale}(x_1, x_2) =&(1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + \\ &(2.625 - x_1 + x_1x_2^3)^2 \end{aligned} \tag{13}$$

The third experiment is the approximation of the *Goldstein–Price* function [17] depicted in formula (14). The characteristic of the *Goldstein–Price* multimodal 2D function is the existence of one global minimum at

$x^* = f(0, -1), f(x^*) = 3$ [33].

$$\begin{aligned} f_{GoldsteinPrice}(x_1, x_2) =& [1 + (x_1 + x_2 + 1)^2 \cdot \\ & (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot \\ & [30 + (2x_1 - 3x_2)^2 \cdot \\ & (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned} \tag{14}$$

The fourth experiment is the approximation of the *Holder Table* function [44] depicted in formula (15). The characteristic of the *Holder Table* multimodal 2D function is the existence of four global minimums located at $x^* = f(\pm 9.646168, \pm 9.646168), f(x^*) = -26.920336$ [33].

$$f_{HolderTable}(x_1, x_2) = -\left| sin(x_1)cos(x_2)\exp\left(\left|1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right|\right)\right| \tag{15}$$

The fifth experiment is the approximation of the 2D function depicted in formula (16) which has multiple global minimums.

$$f_(x_1, x_2) = (x_1^2 - x_2^2)sin(0.5x_1). \tag{16}$$

The results in terms of MSE from the comparison of He-HyELM with classic ELM, Ho-HyELM, OP-ELM for homogeneous networks and OP-ELM for heterogeneous networks for the above function approximation problems can be summarized in Table 5. The linear unit with the *sigmoid* activation-output function ($\rho = 1$) was selected for ELM and homogeneous OP-ELM. For heterogeneous OP-ELM the created networks contained linear units with three different transfer functions (*Gaussian, linear, sigmoid*). Ho-HyELM and He-HyELM both used custom neurons containing linear and multi-cube dendrites, the 'sum of the weights' activation function and six activation-output functions (*bent identity, Gaussian, hyperbolic tangent, sigmoid, sinusoid, soft sign*). The number of hidden units for ELM , Ho-HyELM and He-HyELM and the maximum number of hidden units allowed for both OP-ELM variations were set to 15.

Table 5: Comparison of the Approximation Results for Each Function in terms of MSE

Datasets	ELM	Ho-HyELM	OP-ELM (Hom.)	OP-ELM (Het.)	He-HyELM
Ackley	3.88×10^{-3}	1.19×10^{-3}	1.83×10^{-3}	1.26×10^{-3}	$\mathbf{7.44 \times 10^{-4}}$
Beale	1.68×10^{-4}	1.11×10^{-4}	1.45×10^{-3}	1.33×10^{-4}	$\mathbf{7.8 \times 10^{-6}}$
Goldstein Price	7.45×10^{-4}	4.79×10^{-4}	1.54×10^{-3}	9.4×10^{-4}	$\mathbf{6.02 \times 10^{-5}}$
Holder Table	9.91×10^{-3}	9.23×10^{-3}	1.16×10^{-2}	1.13×10^{-2}	$\mathbf{6.99 \times 10^{-3}}$
$(x_1^2 - x_2^2)$ $sin(0.5x_1)$	1.64×10^{-2}	1.33×10^{-2}	4.41×10^{-2}	3.28×10^{-2}	$\mathbf{8.64 \times 10^{-3}}$

The results in terms of MSE from the comparison of He-HyELM with ELM, Ho-HyELM, OP-ELM for homogeneous networks and OP-ELM for heterogeneous networks for the 'airfoil self-noise', 'auto MPG', 'concrete compressive strength', 'servo' and 'yacht hydrodynamics' real world regression datasets can be summarized in Table 6.

Table 6: Comparison of Results Based on Real-world Regression Datasets in terms of MSE

Datasets	ELM	Ho-HyELM	OP-ELM (Hom.)	OP-ELM (Het.)	He-HyELM
Airfoil Self-Noise	9.63×10^{-4}	9.51×10^{-4}	1.32×10^{-3}	9.44×10^{-4}	$\mathbf{7.56 \times 10^{-4}}$
Auto MPG	3.93×10^{-3}	3.89×10^{-3}	6.61×10^{-3}	3.81×10^{-3}	$\mathbf{3.35 \times 10^{-3}}$
Concrete Compressive Strength	1.31×10^{-2}	1.22×10^{-2}	2.32×10^{-2}	1.12×10^{-2}	$\mathbf{9.77 \times 10^{-3}}$
Servo	1.74×10^{-2}	1.49×10^{-2}	2.38×10^{-2}	1.6×10^{-2}	$\mathbf{1.08 \times 10^{-2}}$
Yacht Hydrodynamics	8.17×10^{-3}	6.23×10^{-3}	1.54×10^{-2}	5.89×10^{-3}	$\mathbf{2.94 \times 10^{-3}}$

As shown in the above Tables, the heterogeneous ELM trained networks created from the proposed He-HyELM algorithm achieved lower MSE than the compared methods in all regression datasets.

4.3. Classification problems

The results in terms of accuracy percentage from the comparison of He-HyELM with ELM, Ho-HyELM, OP-ELM for homogeneous networks and OP-ELM for heterogeneous networks for the 'banknote authentication', 'breast cancer Wisconsin', 'cryotherapy', 'diabetic retinopathy Debrecen', 'HIV-1 protease cleavage' and 'mammographic mass' real world classification datasets can be summarized in Table 7.

Table 7: Comparison of Results Based on Real-world Classification Datasets in terms of Accuracy Percentage

Datasets	ELM	Ho-HyELM	OP-ELM (Hom.)	OP-ELM (Het.)	He-HyELM
Banknote Authentication	98.36%	98.91%	97.05%	98.98%	**100%**
Breast Cancer Wisconsin	96.89%	97.26%	95.48%	96.93%	**97.61%**
Cryotherapy	89.11%	90.44%	81.28%	88.03%	**94.11%**
Diabetic Retinopathy Debrecen	68.13%	69.75%	60.39%	71.28%	**72.94%**
HIV-1 Protease Cleavage	70.76%	71.2%	67.6%	72.77%	**75.31%**
Mammographic Mass	81.44%	81.97%	78.14%	80.58%	**82.57%**

As shown in the above Table, the heterogeneous ELM trained networks created from the proposed He-HyELM algorithm achieved higher accuracy percentage than the compared methods in all classification datasets.

5. Discussion

The He-HyELM experimental results depicted in Sections 4.2 and 4.3 experimentally proved that the proposed method outperformed the other

ELM methods. Specially, in the function approximation problems, the results showed a significant difference in the generalization error since in four out of five datasets the error was significantly reduced. The results were also consistent in the real world regression and classification datasets, but the error reduction for the regression problems and the accuracy increase for the classification problems did not have a very significant change. One interesting result was in the 'banknote authentication' classification dataset where He-HyELM managed to get 100% accuracy compared to 98.98% accuracy from heterogeneous OP-ELM which was the second best.

The purpose of this study was to propose a method which would be able to create heterogeneous SLNNs with better generalization ability than their equivalent homogeneous ones in acceptable time. The exhaustive search for all possible combinations of heterogeneous layers would be unfeasible in short time. It would require an enormous number of networks, to be trained with ELM, which rises exponentially with every new neuron added to the network's hidden layer. The utilization of a modified GA for this task resulted in the creation of heterogeneous networks with lower MSE for regression problems and higher accuracy for classification problems in acceptable time than the ELM-based compared methods.

One issue of the He-HyELM approach is its increased processing power demand if compared with the OP-ELM method. This is due to the fact that He-HyELM requires to train a set of networks at each generation. A solution to this problem would be the utilization of a multi-core system or a CUDA compatible graphics processing unit.

6. Conclusion

The He-HyELM approach used a set of custom neurons to create a series of homogeneous SLNNs that were evolved to heterogeneous SLNNs with the help of a GA. At the end of the evolution process the best network was selected as the most optimal according to a fitness criterion.

A series of regression and classification problems were used to validate whether the He-HyELM algorithm was able to create heterogeneous SLNNs with good generalization ability. The purpose of these experiments was to find the optimal combination of hidden layer units for each problem. Experimentally, we validated that the creation of different heterogeneous networks for each specific problem achieved better generalization performance than the homogeneous networks having the same number of hidden units. This claim was experimentally proven because in all three different categories of datasets, the He-HyELM networks outperformed the homogeneous networks created using ELM, HyELM and OP-ELM. Also, it managed to outperform the heterogeneous networks created using OP-ELM.

The proposed He-HyELM method can be modified in order to create an ensemble classifier, by generating a number of different architectures using multiple GA optimizations. This can result to individual classifiers of high strength, while the GA optimization will ensure the low correlation; this two attributes (low correlation between classifiers and individual classifier strength) guarantee the quality of the ensemble according to Breiman [8]. This issue will be addressed in future communications.

References

[1] Abuassba, A. O., Zhang, D., Luo, X., Shaheryar, A., Ali, H., 2017. Improving classification performance through an advanced ensemble based heterogeneous extreme learning machines. Computational intelligence and neuroscience 2017.

[2] Antal, B., Hajdu, A., 2014. An ensemble-based system for automatic screening of diabetic retinopathy. Knowledge-based systems 60, 20–27.

[3] Back, T., 1996. Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms. Oxford university press.

[4] Bai, Z., Huang, G.-B., Wang, D., Wang, H., Westover, M. B., 2014. Sparse extreme learning machine for classification. IEEE transactions on cybernetics 44 (10), 1858–1870.

[5] Bakker, B., Heskes, T., 2003. Clustering ensembles of neural network models. Neural networks 16 (2), 261–269.

[6] Bartlett, P. L., 1998. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE transactions on Information Theory 44 (2), 525–536.

[7] Bazi, Y., Alajlan, N., Melgani, F., Hichri, H., Malek, S., Yager, R. R., 2014. Differential evolution extreme learning machine for the classification of hyperspectral images. IEEE Geosci. Remote Sensing Lett. 11 (6), 1066–1070.

[8] Breiman, L., 2001. Random forests. Machine learning 45 (1), 5–32.

[9] Christou, V., Tsipouras, M. G., Giannakeas, N., Tzallas, A. T., 2018. Hybrid extreme learning machine approach for homogeneous neural networks. Neurocomputing.

[10] Clarkson, T. G., Gorse, D., Taylor, J. G., Ng, C., 1992. Learning probabilistic RAM nets using VLSI structures. IEEE Transactions on computers 41 (12), 1552–1561.

[11] Cortes, C., Vapnik, V., 1995. Support-vector networks. Machine learning 20 (3), 273–297.

[12] Dheeru, D., Karra Taniskidou, E., 2017. UCI machine learning repository.
URL `http://archive.ics.uci.edu/ml`

[13] Eberhart, R., Kennedy, J., 1995. A new optimizer using particle swarm theory. In: Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on. IEEE, pp. 39–43.

[14] Elter, M., Schulz-Wendtland, R., Wittenberg, T., 2007. The prediction of breast cancer biopsy outcomes using two cad approaches that both emphasize an intelligible decision process. Medical physics 34 (11), 4164–4172.

[15] Feng, G., Qian, Z., Zhang, X., 2012. Evolutionary selection extreme learning machine optimization for regression. Soft Computing 16 (9), 1485–1491.

[16] Ferguson, A., 1995. Learning in RAM-based artificial neural networks. Ph.D. thesis, University of Herfordshire.

[17] Goldstein, A., Price, J., 1971. On descent from local minima. Mathematics of Computation 25 (115), 569–574.

[18] Gorse, D., Taylor, J., 1990. A general model of stochastic neural processing. Biological Cybernetics 63 (4), 299–306.

[19] Gurney, K., 1989. Learning in networks of structured hypercubes. Tech. rep., Brunel Univ., Uxbridge (UK).

[20] Gurney, K., 1997. An introduction to neural networks. CRC press.

[21] He, S., Wu, Q. H., Saunders, J., 2009. Group search optimizer: an optimization algorithm inspired by animal searching behavior. IEEE transactions on evolutionary computation 13 (5), 973–990.

[22] Holland, J., Goldberg, D., 1989. Genetic algorithms in search, optimization and machine learning. Massachusetts: Addison-Wesley.

[23] Holland, J. H., 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

[24] Huang, G., Huang, G.-B., Song, S., You, K., 2015. Trends in extreme learning machines: A review. Neural Networks 61, 32–48.

[25] Huang, G.-B., 2003. Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Transactions on Neural Networks 14 (2), 274–281.

[26] Huang, G.-B., Babri, H. A., 1998. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. IEEE Transactions on Neural Networks 9 (1), 224–229.

[27] Huang, G.-B., Chen, L., 2007. Convex incremental extreme learning machine. Neurocomputing 70 (16-18), 3056–3062.

[28] Huang, G.-B., Chen, L., 2008. Enhanced random search based incremental extreme learning machine. Neurocomputing 71 (16-18), 3460–3468.

[29] Huang, G.-B., Chen, L., Siew, C. K., et al., 2006. Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Networks 17 (4), 879–892.

[30] Huang, G. B., Zhou, H., Ding, X., Zhang, R., April 2012. Extreme learning machine for regression and multiclass classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42 (2), 513–529.

[31] Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., 2004. Extreme learning machine: a new learning scheme of feedforward neural networks. In: Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on. Vol. 2. IEEE, pp. 985–990.

[32] Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., 2006. Extreme learning machine: theory and applications. Neurocomputing 70 (1), 489–501.

[33] Jamil, M., Yang, X.-S., 2013. A literature survey of benchmark functions for global optimisation problems. International Journal of Mathematical Modelling and Numerical Optimisation 4 (2), 150–194.

[34] Khozeimeh, F., Alizadehsani, R., Roshanzamir, M., Khosravi, A., Layegh, P., Nahavandi, S., 2017. An expert system for selecting wart treatment method. Computers in biology and medicine 81, 167–175.

[35] Khozeimeh, F., Jabbari Azad, F., Mahboubi Oskouei, Y., Jafari, M., Tehranian, S., Alizadehsani, R., Layegh, P., 2017. Intralesional immunotherapy compared to cryotherapy in the treatment of warts. International journal of dermatology 56 (4), 474–478.

[36] Li, B., Li, Y., Rong, X., 2013. The extreme learning machine learning algorithm with tunable activation function. Neural Computing and Applications 22 (3-4), 531–539.

[37] Lipowski, A., Lipowska, D., 2012. Roulette-wheel selection via stochastic acceptance. Physica A: Statistical Mechanics and its Applications 391 (6), 2193–2196.

[38] Liu, H., Motoda, H., 2012. Feature selection for knowledge discovery and data mining. Vol. 454. Springer Science & Business Media.

[39] Liu, Y., Yao, X., Higuchi, T., 2000. Evolutionary ensembles with negative correlation learning. IEEE Transactions on Evolutionary Computation 4 (4), 380–387.

[40] Luo, J., Vong, C.-M., Wong, P.-K., 2014. Sparse bayesian extreme learning machine for multi-classification. IEEE Transactions on Neural Networks and Learning Systems 25 (4), 836–843.

[41] Mangasarian, O. L., Wild, E. W., 2001. Proximal support vector machine classifiers. In: Proceedings KDD-2001: Knowledge Discovery and Data Mining. Citeseer.

[42] Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., Lendasse, A., 2010. OP-ELM: Optimally pruned extreme learning machine. IEEE Transactions on Neural Networks 21 (1), 158–162.

[43] 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 (16), 2413–2421.

[44] Mishra, S. K., 2006. Global optimization by differential evolution and particle swarm methods: Evaluation on some benchmark functions.

[45] Mitchell, M., 1998. An introduction to genetic algorithms. MIT press.

[46] Morgan, P., Ferguson, A., Bolouri, H., 1994. Cost-performance analysis of FPGA, VLSI and WSI implementations of a RAM-based neural network. In: Microelectronics for Neural Networks and Fuzzy Systems, 1994., Proceedings of the Fourth International Conference on. IEEE, pp. 235–243.

[47] Neville, R., Glover, R., Stonham, J., 1995. Mapping sigma–pi networks and the associative reward-penalty training regime to the associative string processor. IEEE Transactions on Massively Parallel Processors/Associative Processors (Special Issue).

[48] Opitz, D. W., Shavlik, J. W., 1996. Generating accurate and diverse members of a neural-network ensemble. In: Advances in neural information processing systems. pp. 535–541.

[49] Rögnvaldsson, T., You, L., Garwicz, D., 2014. State of the art prediction of hiv-1 protease cleavage sites. Bioinformatics 31 (8), 1204–1210.

[50] Rosen, B. E., 1996. Ensemble learning using decorrelated neural networks. Connection science 8 (3-4), 373–384.

[51] Sánchez-Monedero, J., Gutiérrez, P. A., Hervás-Martínez, C., 2013. Evolutionary ordinal extreme learning machine. In: International Conference on Hybrid Artificial Intelligence Systems. Springer, pp. 500–509.

[52] Silva, D. N., Pacifico, L. D., Ludermir, T. B., 2011. An evolutionary extreme learning machine based on group search optimization. In: Evolutionary computation (CEC), 2011 IEEE congress on. IEEE, pp. 574–580.

[53] Similä, T., Tikka, J., 2005. Multiresponse sparse regression with application to multidimensional scaling. In: International Conference on Artificial Neural Networks. Springer, pp. 97–102.

[54] Sivanandam, S., Deepa, S., 2008. Genetic algorithms. In: Introduction to genetic algorithms. Springer, pp. 15–37.

[55] Storn, R., 1996. On the usage of differential evolution for function optimization. In: Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American. IEEE, pp. 519–523.

[56] Storn, R., Price, K., 1997. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11 (4), 341–359.

[57] Suykens, J. A., Vandewalle, J., 1999. Least squares support vector machine classifiers. Neural processing letters 9 (3), 293–300.

[58] Syswerda, G., 1989. Uniform crossover in genetic algorithms. In: Proceedings of the third international conference on Genetic algorithms. Morgan Kaufmann Publishers, pp. 2–9.

[59] Tapson, J., Cohen, G., van Schaik, A., 2015. ELM solutions for event-based systems. Neurocomputing 149, 435–442.

[60] Umbarkar, A., Sheth, P., 2015. Crossover operators in genetic algorithms: A review. ICTACT journal on soft computing 6 (1).

[61] Whitley, D., 1994. A genetic algorithm tutorial. Statistics and computing 4 (2), 65–85.

[62] Xu, Y., Shu, Y., 2006. Evolutionary extreme learning machine–based on particle swarm optimization. In: International Symposium on Neural Networks. Springer, pp. 644–652.

[63] Yeh, I.-C., 1998. Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete research 28 (12), 1797–1808.

[64] Zhang, L., Deng, P., 2017. Abnormal odor detection in electronic nose via self-expression inspired extreme learning machine. IEEE Transactions on Systems, Man, and Cybernetics: Systems (99), 1–11.

[65] Zhang, L., Wang, X., Huang, G.-B., Liu, T., Tan, X., 2018. Taste recognition in e-tongue using local discriminant preservation projection. IEEE Transactions on Cybernetics.

[66] Zhang, L., Zhang, D., 2015. Domain adaptation extreme learning machines for drift compensation in e-nose systems. IEEE Transactions on instrumentation and measurement 64 (7), 1790–1801.

[67] Zhang, L., Zhang, D., 2015. Domain adaptation transfer extreme learning machines. In: Proceedings of ELM-2014 Volume 1. Springer, pp. 103–119.

[68] Zhang, L., Zhang, D., 2016. Robust visual knowledge transfer via extreme learning machine-based domain adaptation. IEEE Trans. Image Processing 25 (10), 4959–4973.

[69] Zhang, L., Zhang, D., 2017. Evolutionary cost-sensitive extreme learning machine. IEEE transactions on neural networks and learning systems 28 (12), 3045–3060.

[70] Zhao, Z.-S., Feng, X., Lin, Y.-y., Wei, F., Wang, S.-K., Xiao, T.-L., Cao, M.-Y., Hou, Z.-G., 2015. Evolved neural network ensemble by multiple heterogeneous swarm intelligence. Neurocomputing 149, 29–38.

[71] Zhou, Z.-H., Wu, J., Tang, W., 2002. Ensembling neural networks: many could be better than all. Artificial intelligence 137 (1-2), 239–263.

[72] Zhu, Q.-Y., Qin, A. K., Suganthan, P. N., Huang, G.-B., 2005. Evolutionary extreme learning machine. Pattern recognition 38 (10), 1759–1763.