Self-Adaptive Hybrid Extreme Learning Machine for Heterogeneous Neural Networks

Vasileios Christou*,† Georgios Ntritsos†,‡, Alexandros T. Tzallas†, Markos G. Tsipouras§ and Nikolaos Giannakeas†

* Q Base R&D, Science & Technology Park of Epirus, University of Ioannina Campus, Ioannina GR45110, Greece

† Department of Informatics and Telecommunications, University of Ioannina, Arta, Greece

‡ Department of Hygiene and Epidemiology, University of Ioannina Medical School, Ioannina, Greece

§ Deptartment of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece

Email: *,†bchristou1@gmail.com,†,‡gntritsos@uoi.gr,¶ktzimourta@cc.uoi.gr,†tzallas@uoi.gr,§mtsipouras@uowm.gr, †giannakeas@uoi.gr

***Abstract*—This paper presents a hybrid algorithm for the creation of heterogeneous single layer neural networks (SLNNs). The proposed self-adaptive heterogeneous hybrid extreme learning machine (SA-He-HyELM) trains a series of SLNNs with different neuron types in the hidden layer utilizing the extreme learning machine (ELM) algorithm. These networks are evolved into heterogeneous networks (networks having different combinations of hidden neurons) with the help of a modified genetic algorithm (GA). The algorithm is able to handle two architecturally different neuron types: traditional low order (linear) units and higher order units with different transfer functions. The GA is fully self-adaptive and uses one novel hybrid crossover operator along with a self-adaptive mutation operator in order to retain ELM's simplicity and minimize the number of parameters need tuning. The experimental part of the current paper involves testing SA-He-HyELM with traditional ELM and other three ELM-based methods. The experimental part utilized a series of regression and classification experiments on relatively large datasets. In all cases the proposed method managed to get lower MSE or higher classification accuracy when compared to the aforementioned methods.**

***Index Terms*—extreme learning machine, genetic algorithm, higher order unit, hybrid algorithm, single layer neural network**

I. Introduction

A relatively new method named extreme learning machine (ELM) was proposed by Huang et al. [1], [2] for a special case of feed forward artificial neural networks (ANNs) having one input layer, one hidden layer and one or more output nodes. This method treats an ANN like a linear system of equations. It is able to train it by randomizing the hidden layer weights and thresholds and then with the help of the Moore-Penrose pseudo-inverse it is able to analytically calculate the output node weights. This method showed that a single layer neural network (SLNN) can be trained by tuning only the output node(s) weights while the hidden layer weights and thresholds could be randomized [3]. One significant characteristic of this method, is it's ability to achieve low training error for regression problems or high accuracy for classification problems with small norm of weights which according to Bartlett's neural network generalization theory [4] leads to better generalization performance [5]. This approach is a lot faster than gradient-based methods which use an iterative process to adapt all weights and thresholds of the ANN.

The advantages of ELM include simplicity (since it doesn't have any parameters need tuning), speed (since it doesn't require a iterative process to train an ANN and efficiency (since it doesn't suffer from the local minimum problem).

Besides the aforementioned advantages, it has some disadvantages which include the requirement of a large number of hidden neurons due to the fact that the hidden layer weights and thresholds are randomized and not tuned [6], [7]. Another disadvantage, is it's inability to create networks with low generalization error or high accuracy containing different combinations of hidden neuron types in an automatic way. The heterogeneous hybrid extreme learning machine (He-HyELM) proposed by Christou et al. [8] tries to circumvent this problem by incorporating a genetic algorithm (GA) to evolve homogeneous networks[1] into heterogeneous networks[2]. The purpose of this algorithm is to select the best heterogeneous network at the end of the evolution process according to it's generalization ability in unknown test data. Although this method is able to create optimal heterogeneous networks, it introduces two user defined parameters which affect the convergence rate of the GA. These parameters are the mutation rate and the number of generations. Another issue of He-HyELM is that it cannot be used with large datasets because it is computationally intensive. The proposed self-adaptive heterogeneous hybrid extreme learning machine (SA-He-HyELM) solves these issues by introducing self-adaptive versions of the mutation rate and the ending criteria of the GA and introduces a novel application specific hybrid crossover operator. Furthermore, its architecture has been modified and now uses an application specific parallel GA which is able to run in a PC having multiple central processing units (CPUs)

This work is partly funded by the project entitled xBalloon, co-financed by the European Union and Greek national funds through the Operational Program for Research and Innovation Smart Specialization Strategy (RIS3) of Ipeiros (Project Code: $H\Pi 1AB - 0028178$).

[1]The term homogeneous networks refers to SLNNs having the same neuron types in the hidden layer.

[2]The term heterogeneous networks refers to SLNNs having different combinations of neuron types in the hidden layer.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

which contain multiple processing cores. This architecture greatly reduces the training time and makes SA-He-HyELM able to run in large datasets. The neuron types created and utilized by SA-He-HyELM are in accordance with the structured composite model (C-Model) proposed by Christou et al. [9]. The C-Model utilizes three neuron sub-components in order to create a custom neuron. These sub-components are the dendrite (D), the activation (S_a) and the activation-output function (S_{ao}). A custom neuron is created by using different sub-components from the above three categories. This model supports traditional neuron types where each input is multiplied with the corresponding weight and higher-order units. In the latter, the number of corresponding weights for each input are calculated according to a mathematical formula. SA-He-HyELM utilizes the higher-order units proposed by Gurney [10], [11]. Both neuron types are analyzed in Section III and utilized in the works of Neville et al. [12]–[14].

Numerous ELM based approaches try to improve various aspects of the original algorithm. A significant number of these approaches are focused in optimizing the hidden layer weights and thresholds utilizing GA-based approaches. Zhu et al. [7] proposed evolutionary ELM (E-ELM) which utilizes differential evolution (DE) for optimizing the hidden layer weights and thresholds. Then, with the help of the Moore-Penrose pseudo-inverse, it is able to calculate the output node(s) weights. DE is a powerful population based, stochastic function minimizer which is able to optimize a problem by trying to improve a candidate solution with regard to a given measure of quality using an iterative approach [15], [16]. Evolving ELM (Evo-ELM) for classification problems by Li et al. [17] is also able to optimize the hidden layer weights and thresholds using an adaptive DE algorithm. It is able to online select the appropriate operator for the generation of the offspring, while the control parameters are adaptively tuned. The above approaches are unable to create heterogeneous networks and in most cases contain parameters which they need manual selection from the user's perspective.

The original ELM algorithm works in batch mode. A series of ELM-based approaches have modified its structure, so it can work with sequential data. Huang et al. [18] proposed on-line sequential ELM (OS-ELM) which utilizes the recursive least-squares (RLS) algorithm. RLS is able to find the coefficients that minimize a weighted linear least squares cost function according to the input signals by using a recursive approach [19]. Scardapane et al. [20] extended OS-ELM to work with implicit feature mappings in the proposed kernel on-line sequential ELM (KOS-ELM). The robust on-line sequential ELM (ROS-ELM) by Hoang et al. [21] extended OS-ELM by adding a systematic bias selection method following the input weights. The main purpose of the above approaches is to make traditional ELM work with sequential data without taking into consideration the creation of networks with heterogeneous structure.

The homogeneous hybrid ELM (Ho-HyELM) by Christou et al. [9] is able to create networks with custom created neurons in the hidden layer and includes support for higher-order neurons. Unfortunately, its lacking a mechanism to create networks with different combinations of hidden units.

A number of ELM-based methods are able to create heterogeneous networks. The ELM algorithm with tunable activation function (TAF-ELM) by Li et al. [22] is able to optimize the hidden layer weights, the hidden layer thresholds and the transfer functions with the use of a DE algorithm. The different neuron types are created adaptively by the DE according to the characteristics of each problem. The optimally pruned ELM (OP-ELM) by Miche et al. [23] is able to create heterogeneous networks by randomly creating one large SLNN with different neuron combinations in the hidden layer. Then, with the use of the multi-response sparse regression[3] (MRSR) algorithm, each hidden neuron is ranked according to its participation in the overall network error. Finally, the neurons that have the least contribution to the overall network error are selected to form the final SLNN. Although, these methods are able to create heterogeneous SLNNs, they don't take into consideration higher order units.

Finally, the non-ELM based method proposed by Tsoulos et al. [25] uses grammatical evolution for the creation of the network structure. This method is able to create networks with multiple hidden layers, including recurrent networks but is unable to create heterogeneous networks.

II. Related Work

A. The ELM Algorithm Structure

The ELM algorithm works in a different way than traditional neural network training algorithms where all the network weights need to be adapted in an iterative way. ELM trains a SLNN by randomizing the hidden layer weights and thresholds. Then, with the help of the Moore-Penrose pseudo-inverse, it analytically calculates the output layer weights [1], [2]. The hidden neurons can be of various types including additive nodes, RBF nodes [26], higher order units [10] and Fourier terms [27]. ELM can train also networks containing other node types as long as they are non-linear piece-wise continuous functions [28].

Huang et al. [26] utilized an incremental construction method to prove that in additive neurons with bounded non-constant piece-wise continuous activation functions $(g : \mathbb{R} \rightarrow \mathbb{R})$ or RBF neurons with integrable piece-wise continuous activation functions $(g : \mathbb{R} \rightarrow \mathbb{R}$ and $\int_R g(x)dx \neq 0)$, the network containing randomly generated hidden layer units can converge to any continuous target function by only tuning the output layer weights [28], [29].

The ELM algorithm begins with the random selection of the hidden layer weights and thresholds usually taken from the $[0, 1]$ or $[-1, 1]$ interval in order to be in accordance with Bartlett's theorem [4]. Then, it calculates the hidden layer matrix H, where each row of the table contains the data for each training pattern introduced to the SLNN while each

[3]Sparse regression is the procedure of selecting a small subset from all available regressors for the prediction of a target variable. MRSR is a general case where the regressors and the target can be multivariate [24].

column of the table contains the training patterns for each hidden unit. The next step is the creation of target output matrix T. Again, each row of the table contains the target values of each training pattern introduced to the SLNN while each column of the table corresponds to the target output values of each output node. The algorithm finalizes with the calculation of output node(s) weights matrix β by multiplying the Moore-Penrose pseudo-inverse of H, which is depicted with the symbol $H^{\dagger}$, with the target output matrix T as seen in formula $\beta = H^{\dagger}T$ [1], [2].

B. The Traditional GA Structure

Traditional GAs are inspired from the natural selection mechanism where stronger individuals have more possibilities to win in a competitive environment. The GA follows the same principle where an optimal solution is found by the winner of a genetic game. A potential solution of a problem is encoded as a set of parameters in binary string form, which is the digital equivalent of the gene sequence in a chromosome. A fitness value is used to evaluate each chromosome towards the problem. The fitness value has high relation with the problem's objective value. Through the evolution process, fitter chromosomes tend to produce high quality offspring which in turn result to a better solution to the problem.

The GA begins with the creation of the initial population which is a pool of usually randomly created chromosomes (although other methods can be utilized). In each evolution cycle which is termed generation, the population is evaluated according to a fitness function. A subsequent generation is created by the individuals of the current population. These individuals are named 'parents' and the fitter ones are selected by a selection mechanism for the reproduction process where the parent genes are recombined in order to produce the offspring of the next generation. This evolution process is expected to produce better (fitter) chromosomes since the best chromosomes, according to fitness, are selected for reproduction which in turn are going to produce better offspring. This process is nature inspired from the "survival of the fittest" mechanism [30].

The evolution process is repeated until a specific stopping criterion is reached. This criterion can be a fixed number of generations, a specific fitness value or when no fitter chromosomes are produced. Then, the optimal solution found is returned [30].

Each generation contains two basic operators named crossover and mutation. In the traditional GA, one-point crossover operator is responsible for the reproduction process. This procedure involves the mutual exchange of genes between two parent chromosomes according to a randomly selected crossover point.

The mutation operation is applied after the reproduction process at each individual where each bit is changed randomly according to a small probability (usually less than 0.1).

III. The SA-He-HyELM Architecture

A. SA-He-HyELM Neuron Types

The custom neurons used in SA-He-HyELM are comprised from three neuron sub-components named dendrite (D), activation function (S_a) and activation-output function (S_{ao}), following the structured composite model (C-Model) by Christou et al. [9]. The last two sub-components form the artificial equivalent to the biological soma of the neuron. The subscripts a and ao are used to distinguish between the soma activation function and soma activation-output function. These sub-components form the custom neuron's structure and can be mathematically modeled by the composite function $y = g(u) = S_{ao}(S_a(D))$ where the input patterns are introduced to the neuron as an input vector. The purpose of the dendrite is to weight the inputs according to a specific formula. The weighted inputs are summed together in the soma activation function and an optional threshold is added (this regards traditional unit types only, since the higher order units utilized in this paper do not support a threshold). Finally, the soma activation-output (transfer) function receives the outcome of the activation function and produces the neuron's output. The experimental part of SA-He-HyELM utilized two types of dendrites which are based on two different neuron types. The first is the linear dendrite (D^l) and is based on the traditional neuron type. The latter is the multi-cube dendrite (D^{mc}) and is based on a special case of the higher order unit proposed by Gurney [10] named multi-cube neuron. These dendrite types are going to be analyzed in the following two sections.

B. Linear Dendrite Structure

The linear dendrite is based on the traditional neuron structure where each input is multiplied by its corresponding weight. The input patterns introduced to the linear dendrite are expressed as the input vector $x = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$ which is multiplied element-wise with the weight vector $w^l = [w_1, w_2, \ldots, w_n] \in \mathbb{R}^n$ forming the formula $D^l = [w_1x_1, w_2x_2, \ldots w_nx_n]$ [31].

C. Cubic Dendrite Structure

In a cubic unit, the number of weights (w_{no}) is not proportional to the number of inputs (n) but increases exponentially according to the formula $w^c_{no} = 2^n \in \mathbb{N}^*$. The participation of each weight to the activation function is calculated according to a probability value which is determined using the equation $P_\mu = \frac{1}{2^n}\prod_{i=1}^{n}(1 + \mu_i \frac{x_i}{x_{max}})$. In this equation, the term $\frac{1}{x_{max}}$ is used to normalize the input at $[-1, 1]$. The term $\mu = \mu_1\mu_2 \ldots \mu_n$ is a positive integer converted to binary string form. The purpose of μ is to change the sign values of the inputs for each product term. The ones are interpreted as pluses while the zeroes as minuses [8]–[10]. The formula for the cubic unit dendritic type is $D^c = w^c_\mu \frac{1}{w^c_{max}} \prod_{i=1}^{n}(1+\mu_i \frac{x_i}{x_{max}})$. It uses the weight vector $w^c_\mu = [w_0, w_1, \ldots, w_{2^n-1}] \in \mathbb{R}^{2^n}$ which can be considered as a multi-dimensional hypercube where each weight corresponds to a site of the hypercube. In

this formula, the factor $\frac{1}{w^c_{max}}$ is used for normalization of the weights at $[-1, 1]$ where the term w^c_{max} defines the maximum weight vector value [8]–[10].

Unfortunately, these types of neurons suffer from scaling problems since the number of weights increases exponentially according to the number of inputs. In order to overcome this problem, Gurney [10] proposed the multi-cube unit which utilizes a set of low dimension hyper-cubes instead of one high dimension hypercube. The experimental part of this paper utilizes a special case of the multi-cube unit where all sub-cubes have the same dimension. The formula describing this dendritic type is $D^{mc} = [w^{mc}_{\mu,1}\frac{1}{w^{mc}_{max}}\prod^{d}_{i=1}(1 + \mu_i\frac{x_{i,1}}{x_{max}}), \ldots, w^{mc}_{\mu,q}\frac{1}{w^{mc}_{max}}\prod^{d}_{i=1}(1 + \mu_i\frac{x_{i,q}}{x_{max}})]$.

In this equation, q is the number of sub-cubes, d is the number of inputs for each sub-cube and w^{mc}_{max} defines the maximum weight vector value.

D. Activation Function Structure

The research part of this paper makes use of the 'sum of weights' soma (S) activation function which aggregates the dendrite elements and in the case of the linear dendrite adds an optional threshold. The formula $S^{SoW}_{\alpha_l} = \sum^{n}_{i=1} x_i w_i + \theta$ describes the 'sum of weights' activation function for linear units. In this equation, the superscript declares the activation function type (for 'sum of weights' activation function the superscript is SoW). The subscript declares the neuron type and can take the values l, c and mc which correspond to the linear, cubic and multi-cube neuron types accordingly. The formula for cubic units is $S^{SoW}_{\alpha_c} = \frac{1}{w^c_{max}2^n}\sum^{2^n-1}_{\mu=0} w^c_\mu \prod^{n}_{i=1}(1+\mu_i\frac{x_i}{x_{max}})$ while the formula for multi-cube units is $S^{SoW}_{\alpha_{mc}} = \frac{1}{w^{mc}_{max}2^d}\sum^{q}_{j=1}\sum^{2^d-1}_{\mu=0} w^{mc}_\mu \prod^{d}_{i=1}(1+\mu_i\frac{x_i}{x_{max}})$ [8]–[10].

E. Activation-Output Function Structure

The neuron's output is produced by the soma (S) activation-output function. This paper uses six different activation-output functions with variable shapes defined by the tuning parameter $\rho = \{0.2, 0.4, 0.6, 0.8, 1\}$. These are bent identity $(S^{BI}_{ao} = \rho\Big(\frac{\sqrt{{S_a}^2+1}}{2} + S_a\Big))$, Gaussian $(S^{G}_{ao} = e^{-\left(\frac{S_a}{\rho}\right)^2})$, hyperbolic tangent $(S^{HT}_{ao} = tanh\Big(\frac{S_a}{\rho}\Big))$, sigmoid $(S^{Sig}_{ao} = \frac{1}{1+e^{-\frac{S_a}{\rho}}})$, sinusoid $(S^{Sin}_{ao} = sin\Big(\frac{S_a}{\rho}\Big))$ and soft sign $(S^{SSign}_{ao} = \frac{\frac{S_a}{\rho}}{1+|\frac{S_a}{\rho}|})$. The superscript at each formula $(BI, G, HT, Sig, Sin, SSign)$ defines the activation-output type while the subscript (ao) differentiates an activation-output function from an activation function.

F. SA-He-HyELM

The SA-HE-HyELM algorithm is constituted by two parts. The first part creates a set of homogeneous networks while the second part selects the most promising ones (50%) and evolves them into heterogeneous networks. Finally, the heterogeneous network with the lowest generalization error for regression problems or the highest accuracy for classification problems is selected as the most optimal.

1) Creation of the Homogeneous SLNNs: The creation of the homogeneous networks involves defining the neuron subcomponents which will form the custom neurons. Then, a set of SLNNs with fixed number of hidden units is created and trained using ELM. This procedure is described in Algorithm 1.

Algorithm 1 Creation of the Homogeneous SLNNs (Part 1)

$$1: Pool_D = \left\{ \begin{array}{l} [w_1x_1, w_2x_2, \ldots w_nx_n], \\ w^{mc}_{\mu,1}\frac{1}{w^{mc}_{max}}\prod^{d}_{i=1}(1+\mu_i\frac{x_{i,1}}{x_{max}}), \ldots, \\ w^{mc}_{\mu,q}\frac{1}{w^{mc}_{max}}\prod^{d}_{i=1}(1+\mu_i\frac{x_{i,q}}{x_{max}}) \end{array} \right\}$$

$$2: Pool_a = \left\{ \begin{array}{l} \sum^{n}_{i=1} x_i w_i + \theta, \\ \frac{1}{w^{mc}_{max}2^d}\sum^{q}_{j=1}\sum^{2^d-1}_{\mu=0} w^{mc}_\mu \prod^{d}_{i=1}(1+\mu_i\frac{x_i}{x_{max}}) \end{array} \right\}$$

$$3: Pool_{ao} = \left\{ \begin{array}{l} \rho\Big(\frac{\sqrt{{S_a}^2+1}}{2} + S_a\Big), \\ e^{-\left(\frac{S_a}{\rho}\right)^2}, tanh\Big(\frac{S_a}{\rho}\Big), \\ \frac{1}{1+e^{-\frac{S_a}{\rho}}}, sin\Big(\frac{S_a}{\rho}\Big), \frac{\frac{S_a}{\rho}}{1+|\frac{S_a}{\rho}|} \end{array} \right\}$$

$$4: Pool_{cn} = \left\{ \begin{array}{l} S^{BI}_{ao_l}, S^{BI}_{ao_{mc}}, S^{G}_{ao_l}, S^{G}_{ao_{mc}}, \\ S^{HT}_{ao_l}, S^{HT}_{ao_{mc}}, S^{Sig}_{ao_l}, S^{Sig}_{ao_{mc}}, S^{Sin}_{ao_l}, \\ S^{Sin}_{ao_{mc}}, S^{SSign}_{ao_l}, S^{SSign}_{ao_{mc}} \end{array} \right\}$$

$$5: Pool_{nets} = \sum^{h}_{i=1}[\beta_{i1}, \ldots, \beta_{im}]g(u), g(u) \in Pool_{cn}$$

The first three pools $(Pool_D, Pool_a, Pool_{ao})$ contain dendrites, activation functions and activation-output functions. The distinction between these pool types is done using the D, a and ao subscripts accordingly. The algorithm begins with the creation of the dendrite pool which contains linear and multi-cube dendrite types. Then, it creates the activation function pool containing linear and multi-cube activation functions (one for each previously created dendrite type). The third neuron sub-component pool contains the bent identity, Gaussian, hyperbolic tangent, sigmoid, sinusoid and soft sign activation-output functions. In step 4, it creates twelve different custom neuron types. In these neuron types, the superscript defines the activation-output type $(BI, G, HT, Sig, Sin, SSign)$. The subscripts ao_l and ao_{mc} distinguish an activation-output function receiving as input a linear activation function from an activation-output function receiving as input a multi-cube activation function. Step 5, involves creating a pool containing homogeneous networks $(Pool_{nets})$ where each network utilizes one different custom neuron from the previous step. A diagram of the procedure can be seen in Fig. 1.

The homogeneous networks can be mathematically modeled as $\sum^{h}_{i=1}[\beta_{i1}, \ldots, \beta_{im}]g(u)$. In this formula, h is the number

of hidden units, m is the number of output units, with β_{im} is depicted the i^{th} weight of the output node m and $(g(u) = u)$ is the activation-output function of the output node(s). These SLNNs can be trained by ELM and they have no threshold in the output node(s).

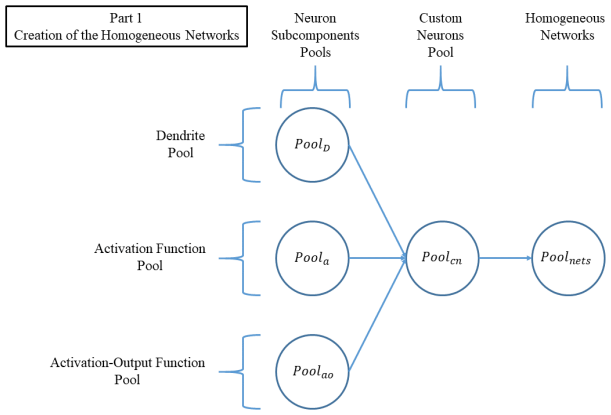

Figure 1. Creation of the Homogeneous SLNNs (Part 1). This figure shows the creation of the homogeneous networks. The algorithm begins with the creation of three pools containing different number of neuron subcomponents. These subcomponents are utilized for the creation of the custom neurons which will then form a set of SLNNs with the same number of neurons in the hidden layer.

2) Evolution of the Homogeneous SLNNs into Heterogeneous SLNNs : The second part of the algorithm utilizes a GA in order to evolve the homogeneous networks created in the previous part into heterogeneous networks. This procedure is described in Algorithm 2 and begins with the selection of the homogeneous networks which will form the initial population.

Algorithm 2 Creation of the Heterogeneous SLNNs (Part 2)

1 : $create\ Population$
2 : **Loop**
3 : $Population_{evaluate} = evaluate(Population)$
4 : **If** $best(Population_{evaluate})$ is unchanged for 3
5 : generations
5 : $Net \leftarrow best(Population_{evaluate})$
6 : **Return** Net
7 : **End If**
8 : $Population_{select} \leftarrow \frac{Population_{evaluate}}{2}$
9 : $Offspring \leftarrow crossover(Population_{select})$
10 : $Offspring \leftarrow mutation(Offspring)$
11 : $Population \leftarrow Population + Offspring$
12 :**End Loop**

Each hidden layer is represented as a chromosome with each hidden neuron encoded as a gene. The next step of the algorithm begins the evolution process. In step 3, the population is evaluated according to a fitness function. Initially, the SLNNs forming the population are trained with ELM. The training process is done in parallel by utilizing a multiple CPU system where each CPU contains multiple processing cores. When the training process is completed, the networks are ranked from best to worst according to fitness.

Each dataset is partitioned into training/validation/test sets. A percentage of the dataset is retained as test set and the rest of the dataset is divided into training/validation sets using the k-fold cross-validation method. The ranking mechanism utilized the average k-fold cross-validation error over all folds in order to rank the SLNNs from best to worst.

The fitness criterion for regression problems was the average mean square error (MSE) where the network which managed to get the lowest value is considered as the most optimal. The formula $MSE = \frac{1}{kp}\sum_{i=1}^{k}\left(\sum_{j=1}^{p}(t_i^j - y_i^j)^2\right)$ calculates the average MSE. In this equation, k is the number of folds, p is the number of validation patterns, t_i^j is the current pattern target network output value for the current fold and y_i^j is the current pattern network output value for the current fold. The fitness criterion for classification problems was the average accuracy (acc) where the network which managed to get the highest value is considered as the most optimal. The formula $acc = \frac{1}{k}\sum_{i=1}^{k}\left(1 - \frac{err}{p}\right)$ calculates the average acc. In this equation, k is the number of folds, p is the number of validation patterns and err is the number of misclassified test patterns for the current fold. When the ranking process is completed, 50% of the networks with the highest fitness values are selected to form the current population.

In step 4, the termination criterion checks if the best network found is unchanged for three generations. In case this condition is true, it selects the best network found according to fitness and terminates the evolution process (steps 5,6). In case the condition is false, the evolution process continues with the selection operator (step 8) where 50% of the networks with the lowest generalization error for regression problems or highest accuracy for classification problems are selected for the reproduction process.

In step 9, the GA reproduces the population using the proposed application specific hybrid neuron ranking masked crossover operator. This is an adaptive operator which takes into consideration the fitness value at each generation in order to switch between two different crossover operators. If the fitness value changes and a more optimal network is found, the crossover operator is set to the uniform crossover, otherwise the proposed neuron ranking masked crossover operator is utilized. The uniform crossover operator creates two offspring having n genes by exchanging information between the parent chromosomes utilizing a uniform random real number $u \in [0, 1]$. This number decides if the first descendant will have the i^{th} genes from the first or the second parent. [32], [33]. If the fitness value doesn't change and a more optimal network has not been found, the proposed neuron ranking masked crossover operator is utilized. This operator uses one binary vector (mask) for each parent chromosome with purpose to guide the reproduction process. These masks are created by ranking the genes (neurons) of each chromosome using the MRSR algorithm [24]. The MRSR algorithm ranks each hidden layer neuron according to its participation in the overall network error. The hidden layer neurons are divided as significant (50%) and less significant (50%). The neurons that contribute less to the overall network error receive the 0 value and they are considered significant while the neurons

that contribute more to the overall network error receive the value 1 and they are considered less significant. The parent mask vectors are created at each generation by setting the value 0 to the positions of the high ranked neurons and 1 to the positions of the low ranked neurons. The reproduction process for the first offspring is done according to the first mask vector. The gene positions with the 0 value from the first parent are retained at the first offspring while the gene positions with the 1 value are replaced with the genes at the same positions from the second chromosome. This procedure is repeated for the second parent using the second mask vector as a guide for the reproduction process and replacing the low ranked neurons of the second chromosome with the equivalent ones from the first chromosome. If the reproduction process produces two identical offspring, the second offspring is randomly shuffled.

The proposed hybrid neuron masked crossover operator has the advantage of switching to a more promising intelligent reproduction process when the uniform crossover doesn't produce fitter offspring and switching back to the uniform crossover when fitter offspring are produced. Having only the neuron masked crossover operator would result in a slower evolution process since the MRSR ranking mechanism introduces a computational cost. The proposed hybrid crossover operator is able to retain a balance between efficiency and computational cost.

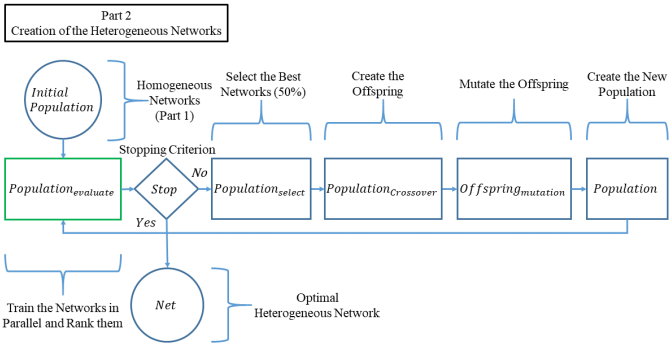

Figure 2. Creation of the Heterogeneous SLNNs (Part 2). This figure shows the creation of the heterogeneous networks. The algorithm begins with the creation of the initial population utilizing the homogeneous networks created in part 1. The evolution process begins with the evaluation of the population. Then, the GA checks if the stopping criteria have been satisfied in order to return the optimal heterogeneous network found. If the stopping criteria have not been satisfied, the evolution process continues with the selection of the best networks (50%) for the reproduction process (crossover). The produced offspring are then mutated and added to the existing population with purpose to form the population of the next generation.

In step 10, a percentage of the offspring has one of its neurons replaced randomly by a neuron taken from the custom neuron pool. The proposed application specific self-adaptive mutation operator is able to adapt the mutation rate by taking into consideration the value of the fitness function at each generation. If the evolution has just begun or if a more optimal fitness value has been found, the mutation rate is set to 10%. If the offspring do not produce fitter descendants, then the mutation rate gradually increases by 20% to a maximum value of 50%. The purpose of setting the mutation rate to a maximum value of 50% is because a higher mutation rate would convert the GA to a random search which is a primitive optimization method [34].

The proposed self-adaptive mutation operator is able to enhance the global search of the search space by gradually increasing the mutation rate to a maximum value (50%) when no fitter offspring have been produced. On the other hand, it is able to enhance the local search of the search space when fitter offspring have been produced by setting the mutation step to its lowest value (10%).

Finally, in step 11 the mutated offspring are added to the current population with purpose to form the population of the next generation. The evolution process is repeated until the stopping criterion is satisfied. A diagram visualizing this procedure is seen in Fig. 2.

IV. Experimental Results

The experimental results involve the comparison of the proposed SA-He-HyELM with traditional ELM and other three ELM-based methods (Ho-HyELM, OS-ELM and KOS-ELM) in three regression and three classification (one multi-class and two binary) problems. The regression problems are 'air quality [35]', 'appliances energy prediction [36]' and 'combined cycle power plant [37]'. The classification problems are 'crowd-sourced mapping [38]' (multi-class), 'electroencephalogram (EEG) eye state' (binary) and 'high time resolution universe 2 [39]' (binary). All datasets have a large number of samples and have been downloaded from the UCI machine learning repository [40]. The 'air quality' dataset contained missing values. The selected method for circumventing this problem was to replace the missing values from each column with the equivalent average column value.

A. Simulation Parameters

The experiments were executed in MATLAB 2017a using the parameters depicted in Table I.

Table I
SA-He-HyELM Parameters

Parameter Name	Symbol	Values/Types
Linear Inputs No	n	$n \in \mathbb{N}^*$
Linear Neuron Weights	w^l	$w^l \in [-1,1]^n$
Multi-Cube Inputs No	d	$d \in \mathbb{N}^*$
Multi-Cubes No	q	$q = n$
Multi-Cube Neuron Weights	w^{mc}	$w^{mc} \in [-1,1]^{2n}$
Threshold	θ	$\theta \in [-1,1]$
Inputs	x	$x \in [-1,1]^n$
Hidden Layer Neurons No	h	15
Tuning Parameter	ρ	$\rho = \{0.2, 0.4, 0.6, 0.8, 1\}$
Folds No	k	10
Experiment Sets	$expNo$	10

We utilized linear and multi-cube dendrites, the 'sum of the weights' activation function and the six tunable activation-output functions described in Section III-E for the Ho-HyELM and SA-He-HyELM approaches. The $[-1,1]$ interval was used for the normalization of all datasets and for the randomization

of the hidden layer weights and thresholds. The number of multi-cube neuron inputs was set to 1 and the number of multi-cubes was set to n which resulted in $q2^d = n2^1 = 2n$ number of weights. The number of hidden layer units was set to 15 while the tuning parameter ρ took the values 0.2, 0.4, 0.6, 0.8 and 1. Each dataset was partitioned into training/test sets were 20% of the data was kept as test data. One exception was the 'crowdsourced mapping' dataset which was already divided into training (10545 samples) and test (300 samples) sets. The training set for SA-He-HyELM was further divided using the 10-fold cross-validation method. Finally, we conducted 10 repeats of each experiment with different random values for the hidden layer weights and thresholds.

Traditional ELM and OS-ELM used linear units with *sigmoid* transfer functions while Ho-HyELM used the same custom neuron types with SA-He-HyELM. KOS-ELM utilized *Gaussian* kernels and has two parameters that affect significantly its generalization ability. These are the regularization parameter C and the bandwidth γ of the Gaussian kernel which were both searched in $\{2^{-8}, 2^{-4}, 2^0, 2^4, 2^8, 2^{13}\}$ [20]. The network which produced the best results on the test set was selected as the most optimal.

B. Experimental Results for Regression Problems

Table II
RESULTS COMPARISON FOR REGRESSION PROBLEMS

Methods	Air Quality	Appliances Energy Prediction	Combined Cycle Power Plant
ELM	0.00356	0.00848	0.0000719
Ho-HyELM	0.00348	0.00809	0.0000719
OS-ELM	0.06406	0.09087	0.0085477
KOS-ELM	0.00408	0.0112	0.0001941
SA-He-HyELM	**0.00201**	**0.00779**	**0.0000711**

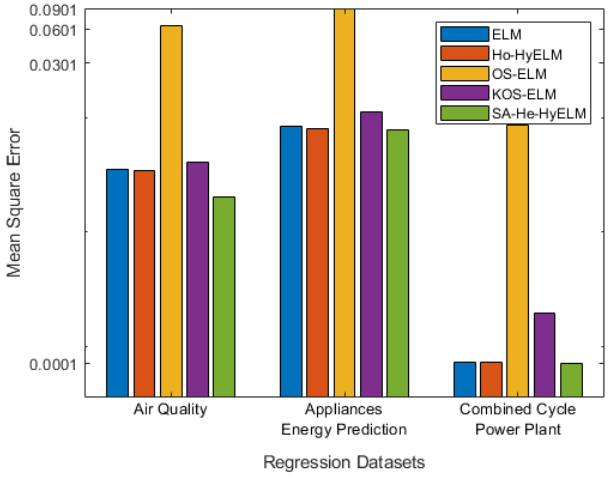

Figure 3. Results Comparison for Regression Problems.

The results from the comparison of SA-He-HyELM with the other methods in terms of MSE are depicted in Table II and visualized in Fig. 3. It can be seen that in all compared datasets, it managed to get the lowest average MSE over all experiment runs (the lowest MSE is marked in bold).

C. Experimental Results for Classification Problems

Table III
RESULTS COMPARISON FOR CLASSIFICATION PROBLEMS

Methods	Crowdsourced Mapping	EEG Eye State	High Time Resolution Universe 2
ELM	37.6%	61.8%	97.3%
Ho-HyELM	39.6%	62.2%	97.5%
OS-ELM	37%	52.9%	97.2%
KOS-ELM	40.3%	58.6%	96.9%
SA-He-HyELM	**47.3%**	**65.6%**	**97.7%**

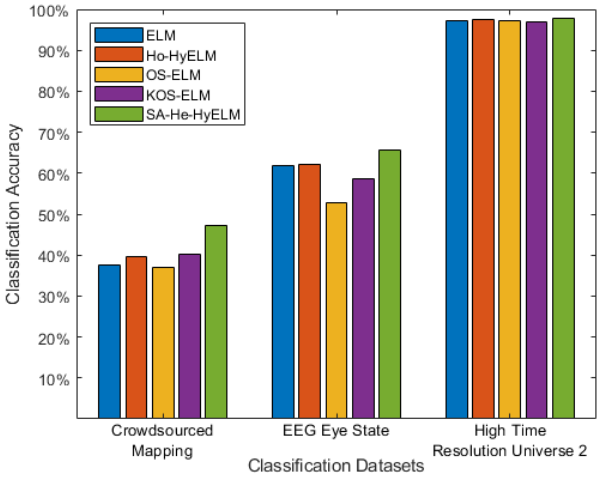

Figure 4. Results Comparison for Classification Problems.

The results from the comparison of SA-He-HyELM with the other methods in terms of accuracy are depicted in Table III and visualized in Fig. 4. It can be seen that in all compared datasets, it managed to get the highest average accuracy over all experiment runs (the highest accuracy is marked in bold).

V. CONCLUSION

SA-He-HyELM is able to evolve homogeneous networks into heterogeneous ones with better generalization ability while retaining ELM's simplicity. The only user defined parameter is the selection of the neuron sub-components that are going to form the custom neurons.

The main purpose of the proposed algorithm was to retain the simplicity of ELM by reducing the number of parameters need tuning. For this reason, the number of generations and the mutation rate of the GA are no longer user defined but they are automatically adjusted. The algorithm is able to exploit the fitness at each generation, in order to take vital decisions for the evolution process. These include the switching criterion between the uniform crossover and the proposed intelligent neuron ranked masked crossover operator, the mutation rate and the termination of the evolution process. Finally, by adapting the algorithm to work in multi-CPU environments, we further reduced the computationally intensive evolution

process and we made SA-He-HyELM to be able to work with large datasets.

REFERENCES

[1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "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, 2004, pp. 985–990.

[2] ——, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[3] G. Huang, G.-B. Huang, S. Song, and K. You, "Trends in extreme learning machines: A review," *Neural Networks*, vol. 61, pp. 32–48, 2015.

[4] P. L. Bartlett, "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*, vol. 44, no. 2, pp. 525–536, 1998.

[5] G.-B. Huang, "An insight into extreme learning machines: random neurons, random features and kernels," *Cognitive Computation*, vol. 6, no. 3, pp. 376–390, 2014.

[6] L. Zhang and D. Zhang, "Evolutionary cost-sensitive extreme learning machine," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 12, pp. 3045–3060, 2016.

[7] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, "Evolutionary extreme learning machine," *Pattern recognition*, vol. 38, no. 10, pp. 1759–1763, 2005.

[8] V. Christou, M. G. Tsipouras, N. Giannakeas, A. T. Tzallas, and G. Brown, "Hybrid extreme learning machine approach for heterogeneous neural networks," *Neurocomputing*, vol. 361, pp. 137–150, 2019.

[9] V. Christou, M. G. Tsipouras, N. Giannakeas, and A. T. Tzallas, "Hybrid extreme learning machine approach for homogeneous neural networks," *Neurocomputing*, vol. 311, pp. 397–412, 2018.

[10] K. Gurney, "Learning in networks of structured hypercubes." 1991.

[11] K. N. Gurney, "Training nets of hardware realizable sigma-pi units," *Neural Networks*, vol. 5, no. 2, pp. 289–303, 1992.

[12] R. S. Neville and T. J. Stonham, "Adaptive critic for sigma-pi networks," *Neural Networks*, vol. 9, no. 4, pp. 603–625, 1996.

[13] R. S. Neville and S. Eldridge, "Transformations of sigma–pi nets: obtaining reflected functions by reflecting weight matrices," *Neural Networks*, vol. 15, no. 3, pp. 375–393, 2002.

[14] R. S. Neville, T. J. Stonham, and R. J. Glover, "Partially pre-calculated weights for the backpropagation learning regime and high accuracy function mapping using continuous input ram-based sigma–pi nets," *Neural Networks*, vol. 13, no. 1, pp. 91–110, 2000.

[15] R. Storn, "On the usage of differential evolution for function optimization," in *Proceedings of North American Fuzzy Information Processing*. IEEE, 1996, pp. 519–523.

[16] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[17] K. Li, R. Wang, S. Kwong, and J. Cao, "Evolving extreme learning machine paradigm with adaptive operator selection and parameter control," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 21, no. supp02, pp. 143–154, 2013.

[18] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, and N. Sundararajan, "On-line sequential extreme learning machine," *Computational Intelligence*, vol. 2005, pp. 232–237, 2005.

[19] M. H. Hayes, "Recursive least squares," *Statistical Digital Signal Processing and Modeling*, p. 541, 1996.

[20] S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini, "Online sequential extreme learning machine with kernels," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 9, pp. 2214–2220, 2014.

[21] M.-T. T. Hoang, H. T. Huynh, N. H. Vo, and Y. Won, "A robust online sequential extreme learning machine," in *International Symposium on Neural Networks*. Springer, 2007, pp. 1077–1086.

[22] B. Li, Y. Li, and X. Rong, "The extreme learning machine learning algorithm with tunable activation function," *Neural Computing and Applications*, vol. 22, no. 3-4, pp. 531–539, 2013.

[23] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "Op-elm: optimally pruned extreme learning machine," *IEEE transactions on neural networks*, vol. 21, no. 1, pp. 158–162, 2009.

[24] T. Similä and J. Tikka, "Multiresponse sparse regression with application to multidimensional scaling," in *International Conference on Artificial Neural Networks*. Springer, 2005, pp. 97–102.

[25] I. G. Tsoulos, D. Gavrilis, and E. Glavas, "Neural network construction using grammatical evolution," in *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005.* IEEE, 2005, pp. 827–831.

[26] G.-B. Huang, L. Chen, C. K. Siew *et al.*, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[27] F. Han and D.-S. Huang, "Improved extreme learning machine for function approximation by encoding a priori information," *Neurocomputing*, vol. 69, no. 16-18, pp. 2369–2373, 2006.

[28] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, no. 16-18, pp. 3460–3468, 2008.

[29] ——, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, no. 16-18, pp. 3056–3062, 2007.

[30] K.-F. Man, K. S. Tang, and S. Kwong, *Genetic algorithms: concepts and designs*. Springer Science & Business Media, 2001.

[31] K. Gurney, *An introduction to neural networks*. CRC press, 2014.

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

[33] A. Umbarkar and P. Sheth, "Crossover operators in genetic algorithms: A review." *ICTACT journal on soft computing*, vol. 6, no. 1, 2015.

[34] T. Panda, T. Theodore, and R. A. Kumar, *Statistical Optimization of Biological Systems*. CRC Press, 2015.

[35] S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. Di Francia, "On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario," *Sensors and Actuators B: Chemical*, vol. 129, no. 2, pp. 750–757, 2008.

[36] L. M. Candanedo, V. Feldheim, and D. Deramaix, "Data driven prediction models of energy use of appliances in a low-energy house," *Energy and buildings*, vol. 140, pp. 81–97, 2017.

[37] P. Tüfekci, "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*, vol. 60, pp. 126–140, 2014.

[38] B. A. Johnson and K. Iizuka, "Integrating openstreetmap crowdsourced data and landsat time-series imagery for rapid land use/land cover (lulc) mapping: Case study of the laguna de bay area of the philippines," *Applied Geography*, vol. 67, pp. 140–149, 2016.

[39] R. J. Lyon, B. Stappers, S. Cooper, J. Brooke, and J. Knowles, "Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach," *Monthly Notices of the Royal Astronomical Society*, vol. 459, no. 1, pp. 1104–1123, 2016.

[40] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml