Higher-Order Extreme Learning Machine

Vasileios Christou*

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

Abstract

The training process of feed-forward neural networks is a slow and computationally intensive procedure mainly due to the iterative nature of most algorithms. A solution to this problem was the creation of the extreme learning machine (ELM) algorithm for single-layer neural networks (SLNNs). This method uses a very fast approach where the hidden-layer weights and thresholds are randomized, and the output layer's weights are analytically calculated using the Moore-Penrose pseudo-inverse. Although it provides good generalization performance, it is restricted in traditional neuron types where each neuron's input is multiplied by its corresponding weight. This paper proposes the higher-order (sigma-pi) ELM algorithm, which generalizes original ELM in six higher-order SLNN variants. Higher-order units utilize more weights than traditional neurons to solve their restriction in linear separable problems. The experimental results showed that higher-order SLNN variants had better generalization performance than SLNNs trained using the classic ELM algorithm in 15 classification and 10 regression datasets taken from the University of California, Irvine (UCI) machine learning repository and www.kaggle.com website.

Keywords: artificial neural network, extreme learning machine, feed forward neural network, higher-order neuron, multi-cube neuron, sigma-pi neuron

*Corresponding Author.
Email address: bchristou1@gmail.com (Vasileios Christou)

1. Introduction

Artificial single-layer neural networks (SLNNs) are popular solutions in various classification and regression tasks. Gradient-based training methods like back-propagation are computationally intensive. The reason behind this is their iterative nature which requires a large number of iterations (epochs) in order to adapt the network's weights and thresholds. Extreme learning machine (ELM) training approach proposed by Huang et al. (2004, 2006b) circumvents this problem. It can train SLNNs faster than alternative methods by randomizing the hidden-layer weights and thresholds. Then, it utilizes the Moore-Penrose pseudo-inverse technique, which is employed to analytically calculate the output layer's weights. The advantages of this method include:

- The lack of user-defined parameters that can affect the training process (e.g., a learning rate).
- Its ability to use non-differentiable transfer functions in the network's neurons.
- It doesn't require a validation set like gradient-based methods.

The classic ELM method was designed to work with the traditional weighted sum of inputs neurons (here we term them low-order neurons) where the inputs are multiplied with an associated weight (Huang et al., 2004, 2006b). One issue that has been highlighted in the past is that a single weighted sum of inputs neuron cannot be used to solve non-linear separable problems like the approximation of the XOR function (Gurney, 2018). A possible solution to this problem was to replace low-order units with more complex neurons termed sigma-pi (which may be viewed as higher-order functionality) units (Gurney, 1989). These circumvent this issue by mapping a neuron input to multiple weights. The higher-order units interpret their inputs as a probability of addressing weights; where μ specifies the addresses of weights (w_μ) and the probability of addressing each weight is P_μ. P_μ can be viewed as coefficients that determine the weight's participation percentage[1] in the accumulated activation. These face the issue that their weights increase exponentially when the number of inputs increases, making them unsuitable for datasets with many attributes. Gurney (1989) provided a solution to this problem by inventing the multi-cube unit, which successfully circumvents this problem by allowing the user to control the number of weights at each assigned input (Gurney, 1989).

This paper presents an adaptation to the ELM model, where SLNN's nodes are replaced by higher-order or multi-cube units in their hidden and output layers. Specifically, it covers the following six network types shown in Tables 1 and 2.

Table 1: Higher-Order Unit Networks

Neuron Types	**Higher-Order**		
Hidden Layer	Higher-Order	Low-Order	Higher-Order
Output Layer	Low-Order	Higher-Order	Higher-Order

Table 2: Multi-Cube Unit Networks

Neuron Types	**Multi-Cube**		
Output Layer	Multi-Cube	Low-Order	Multi-Cube
Output Layer	Low-Order	Multi-Cube	Multi-Cube

The motivation for adapting the ELM algorithm to higher-order neuron SLNNs was to create advanced networks with better generalization ability than low-order ELM-trained models. Earlier works from Christou et al. (2018, 2019, 2020, 2022) support this claim since they presented positive results by utilizing SLNNs with higher-order neurons in their hidden layers. This article expands those works by covering the network models with higher-order neuron types in

[1]Participation percentage implies that a percentage (or contribution) of the weight is utilized to calculate the activation; this contribution is (in the original sigma-pi model) specified as the probability of addressing a weight (P_μ).

their output node(s).

The paper is structured in nine main sections starting with the "Introduction" containing a description of ELM, a description of the higher-order units proposed by Gurney (1989) and the motivation behind the utilization of those units in the creation of ELM-trained SLNNs. The following section describes existing works, while the third section thoroughly describes the low-order and higher-order nodes structure. In contrast, the fourth section describes ELM, while the fifth section presents the pseudo-codes and descriptions for the six higher-order networks proposed in the current article. The sixth section (Experimental work and simulations) shows the results from comparing the above six network types with classic ELM in 15 classification and 10 regression datasets. The following two sections are the "Discussion" and "Conclusion" of the proposed work.

2. Literature review

Since the original ELM invention, many variations of the algorithm have been proposed. Some of these variants aim to increase the robustness of ELM (Wang et al., 2021). The feed-forward network is constructed incrementally by adding hidden units in the incremental ELM (I-ELM) by Huang et al. (2006a). The paper also provides theoretical proof of ELM's universal approximation capability. It utilizes an incremental constructive method to show that SLNNs can work as universal approximators by simply randomizing the hidden layer's weights and thresholds and adjusting the output layer's neuron(s) weights. In these neural network types, the transfer functions for additive nodes can be any bounded non-constant piece-wise continuous function $g : R \to R$. On the other hand, the transfer function for radial basis function (RBF) neurons can be any integrable piece-wise continuous function $g : R \to R$ and $\int_R g(x)dx \neq 0$. Huang & Chen (2007) improved I-ELM by using a convex optimization method to recalculate the output weights of the exiting neurons every time a new hidden unit is added to the SLNN. Xu et al. (2016) proposed incremental recursive

ELM (IR-ELM), which updates output weight recursively every time a new hidden neuron is added to the network. They also created an improved version of IR-ELM named enhanced incremental recursive ELM (EIR-ELM), which contains a set of hidden units that will be added to the SLNN. Cao et al. (2012b) improved ELM for classification problems by training several networks with the same structure. Then, they utilized a voting mechanism on the results from these SLNNs to find the final classification outcome. Huang et al. (2011) created kernel ELM (KELM), which enhances the robustness of ELM by turning low-dimensional linearly non-separable data into linearly separable data. Deng et al. (2013, 2016) proposed reduced kernel ELM (reduced-KELM), which greatly decreases the training time of kernel ELM (KELM) by selecting a random subset of the given dataset. This approach was also successfully applied in cross-person activity recognition task (Deng et al., 2014). Luo et al. (2021) created multinomial Bayesian ELM (MBELM) for multi-class classification problems, which tries to solve some issues of sparse Bayesian ELM (SBELM) (Luo et al., 2013; Wong et al., 2015) in multi-class datasets. SBELM has shown better performance than ELM in generalization, sparsity, and runtime. The MBELM method has two variants that employ different sparse mechanisms. The first variant utilizes automatic relevance determination and is focused on problems where the model size or the execution time is the main priority with a small sacrifice on accuracy and training time. The second variant utilizes an L_1 penalty and is focused on problems where model size and accuracy have the same priority. Zhang & Luo (2015) created an outlier robust ELM variant for regression problems which utilizes the l_1-norm loss function to enhance ELM's robustness. Xing & Wang (2013) introduced a regularized correntropy criterion for training an ELM-based SLNN. This criterion can circumvent the problem of having a training dataset containing noise or outliers. Although the above methods manage to make significant improvements to the ELM algorithm, they do not take into consideration higher-order neurons.

A number of ELM methods are focused on parameter tuning. Chen et al. (2021) created a multi-objective parameter optimization strategy for ELM which can simultaneously optimize the model error and generalization performance. Cao et al. (2021) created an ELM variant based on particle swarm optimization (PSO) and crow search algorithm (CSA). The proposed (PSO-CSA-ELM) model utilizes CSA to optimize the hidden layer weights and thresholds of ELM and PSO to enhance the global search capability of CSA. Rathod & Wankhade (2022)combined cuckoo search (CS) and invasive weed optimization (IWO) for optimizing the hidden layer weights and thresholds. Perales-González et al. (2021) proposed the negative correlation hidden layer ELM (NCHL-ELM) where each hidden unit's output is corrected with an extra parameter's help. The purpose of this correction is to make each hidden neuron correlate negatively with the SLNN's output. All hidden layer units are considered equally significant, conforming with the negative correlation framework's assumption that all base learners have the same significance (Liu & Yao, 1997; Chen & Yao, 2009). The multi-objective optimization-based sparse ELM (MO-SELM) by Wu et al. (2018) integrates parameter optimization and structure learning into the learning process of ELM to resolve its over-fitting problem and increase its generalization performance. Cao et al. (2018) created affine transformation ELM (AT-ELM), which utilizes AT transfer functions and enforces the hidden units' outputs to have a uniform distribution inside the transfer function's range. The random initialization of hidden weights and thresholds moves most hidden node inputs into the transfer function's saturated or linear regions, causing poor generalization performance. Lu et al. (2017) used the active operators PSO (APSO) algorithm for optimizing the internal power parameters of KELM. The proposed APSO-KELM algorithm managed to accomplish good stability and classification performance. Zhu et al. (2005) created evolutionary ELM, which used a modified differential evolution algorithm for optimizing the hidden layer weights and thresholds. Cao et al. (2012a) utilized a self-adaptive differential evolution algorithm to optimize the hidden layer parameters. The proposed SaE-ELM algorithm overcame the limitations of previous approaches like evolutionary ELM (E-ELM), which involved a manual selection of the control parameters and vector generation strategies. Sevinc (2019) combined a genetic algorithm (GA) with ELM for finding the feature subset that would provide the highest classification accuracy. Zhang et al. (2013) tuned ELM's hidden layer parameters using the firefly algorithm while Zhang et al. (2016) utilized a memetic algorithm for the same task. The above algorithms make significant improvements to ELM algorithm but do not consider higher-order neurons.

Many ELM-based methods are created to solve specific problem types. Dou et al. (2022) developed an ELM-based battery capacity estimation method to avoid over-charging and over-discharging lithium-ion battery cells. The proposed method utilizes the salp swarm algorithm (SSA) to optimize the hidden layer nodes' parameters and chaotic mapping for making SSA's initialized individuals uniformly distributed. Tang & Li (2021) created a particle swarm optimized online regularized ELM (IPSO-IRELM) approach for online network intrusion detection. In contrast to classic ELM, which uses a batch learning approach, IPSO-IRELM has a sequential learning mechanism and utilizes an improved version of PSO for optimizing IRELM's initial weights and deviations. IRELM is an improved version of the RELM (Martínez-Martínez et al., 2011) algorithm, which can work with sequential data. RELM can automatically select the ELM architecture based on regularized regression methods. Sulaiman et al. (2022) combined empirical mode decomposition with ELM for solving the residential load forecasting problem. Tian et al. (2021) combined linear fitting with ELM to detect leaks in low-pressure gas distribution pipeline systems based on the negative pressure wave principle. Liu et al. (2021) hybridized ELM with the improved cuckoo search algorithm ICS for finding the actual junction temperature in insulated-gate bipolar transistors for switching and industrial control systems. The ICS algorithm's purpose was to find the optimal hidden layer parameters. Zhang et al. (2020) introduced improved incremental fuzzy-kernel-regularized ELM (I2FELM), a RELM-based Twitter spam detection approach. I2FELM utilizes fuzzy weights to address the unbalanced data problem. Also, Cholesky factorization without square root and composite kernel function is applied to increase performance. Finally, the hidden layer neurons can be calculated automatically using an incremental method. Lu et al. (2019) combined a restricted-Boltzmann strategy with ELM for gas path fault diagnosis of the turbofan engine. The proposed method creates a feature mapping and recursively tunes the hidden layer neurons. Cai et al. (2020) used PSO with ELM to create a short-term traffic flow forecasting system. The purpose behind hybridizing PSO with ELM was to increase the generalization performance of the latter. Similarly, recent research from Cui et al. (2022) used the gravitational search algorithm (GSA) for finding the optimal ELM parameters in the same problem type.

Alternative higher-order neuron types include the sigma-pi units by Feldman & Ballard (1982) which their output is calculated by summarizing the contributions from a set of independent multiplicative clusters of input weights (Rumelhart et al., 1986; Mel & Koch, 1989) and the pi-sigma units by Shin & Ghosh (1991). The pi-sigma networks use product cells as the output neurons to integrate higher-order network capabilities indirectly. Moreover, they utilize a reduced number of weights and processing units. The present article utilizes the higher-order units by Gurney (1989) because the multi-cube units allow the user to control the number of weights at each assigned input. This ability circumvents the problem of higher-order units where the number of weights increases exponentially to the neuron's inputs.

3. Node structures

The current section presents the linear and hyper-cube unit concept used in the higher-order/cubic neurons (cubes) by Gurney (1989). Initially, the connection between neuron inputs and hyper-cubes is explained. Then, a node structure that utilizes multiple hyper-cubes and solves the problem of higher-order units where the number of weights increases exponentially to the neuron's inputs is shown.

3.1. Linear units

In the linear unit, each member of the input vector $x = [x_1, x_2, \ldots, x_n] \in \mathbb{R}, n \in \mathbb{N}$ is multiplied to a corresponding weight taken from the weight vector $w^l = \begin{bmatrix} w_1^l \\ w_2^l \\ \vdots \\ w_n^l \end{bmatrix} \in \mathbb{R}$ and an optional threshold θ is added. It is defined by the activation shown in equation (1) where the l superscript denotes the linear neuron type, n is the number of inputs, x_i is the current input, w_i defines the current weight and θ is the optional threshold.

$$a^l = \sum_{i=1}^{n} x_i w_i^l + \theta \tag{1}$$

The activation's output in (1) is then introduced as input to the activation function $g(a^l)$ which produces the output y. The complete linear unit structure is visualized in Fig.1.

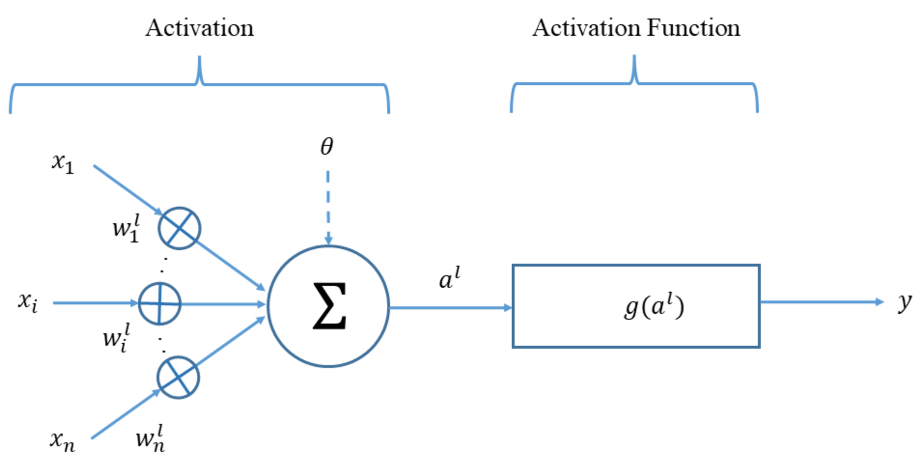

Figure 1: The linear neuron structure. This figure visualizes the linear neuron where each input is multiplied by a corresponding weight and an optional threshold is added. This procedure forms the neuron's activation which is inserted as input to the activation function. The latter is responsible for producing the neuron's output.

3.2. Cubic units

The main difference between low-order and the higher-order neurons by Gurney (1989) is that in the latter, the n-dimensional input vector corresponds to a n-order polynomial defined in $w^c_{no} = 2^n, n \in \mathbb{N}$. This polynomial is regarded as a n-dimensional hyper-cube where each cube's site corresponds to a specific weight (the c superscript denotes the cubic neuron type). This concept is visualized in the 3-dimensional cube in Fig 2.

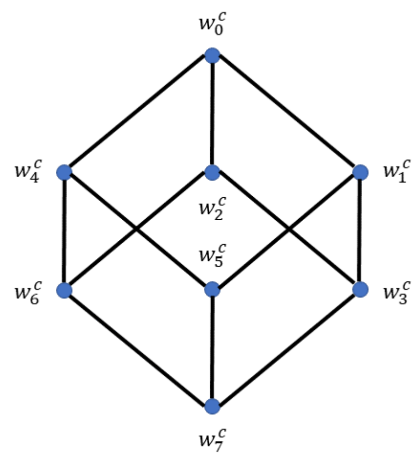

Figure 2: A 3^{rd} order polynomial depicted as a hyper-cube. This figure visualizes a 3^{rd} order polynomial as a hyper-cube where each neuron weight corresponds to a specific site.

The participation of each weight in the cubic activation function is done according to the probability function defined in (2). In this formula, the term $\mu = \mu_1, \mu_2, \ldots, \mu_n$ is an unsigned integer converted to binary form. The binary form conversion is done to modify the input sign value in each product term $(1 + \mu_i x_i)$. The binary format symbols interpretation involves converting each '0' as a '$-$' sign and each '1' as a '+' sign (Gurney, 1989). It should also be noted that the dataset entries introduced as input to the cubic neuron must be normalized. The common normalization method for these neuron types is to divide each dataset feature value with its corresponding highest absolute feature value.

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

The cubic activation defined in formula (3) multiplies each weight with the above probability function and calculates their normalized average. The term $\frac{1}{|w_{max^c}|}$ is used to normalize the activation by dividing its value by the highest absolute weight value, while w^c_μ denotes the current weight value.

$$a^c = \frac{1}{|w^c_{max}|2^n} \sum_{\mu=0}^{2^n-1} w^c_\mu \prod_{i=1}^{n} (1 + \mu_i x_i) \tag{3}$$

The activation's output in (3) is then introduced as input to the activation function $g(a^c)$ which produces the output y. The complete cubic unit structure is visualized in Fig.3.

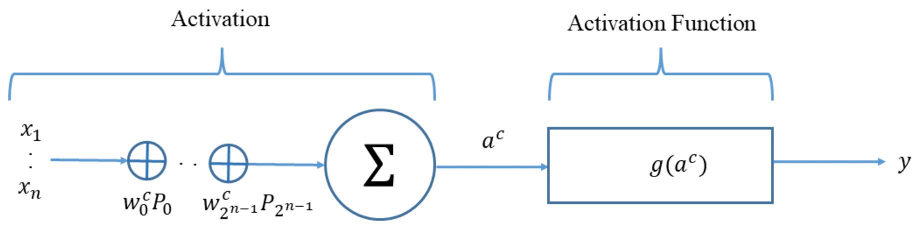

Figure 3: The cubic neuron structure. This figure visualizes the cubic neuron where an input vector containing n inputs corresponds to 2^n weights. Each weight is multiplied by a corresponding probability, and all weights' normalized average is calculated. This procedure forms the neuron's activation, which is inserted as input to the activation function. The latter is responsible for producing the neuron's output.

3.3. Multi-cube units

The multi-cube unit was created by Gurney (1989) to solve cubic neurons' scaling problems. The main difference from the cubic unit is that instead of one large cube, it utilizes a series of lower-order sub-cubes, which greatly reduces the number of needed weights. The n-dimensional input vector corresponds to a q number of lower-dimension sub-cubes where each one of them can have different dimension $d = [d_1, d_2, \dots, d_q]$. The number of weights in the multi-cube neuron is defined in the polynomial $w_{no}^{mc} = p_{no} = \sum_{i=1}^{q} 2^{d_j}, (i, d_i, q) \in \mathbb{N}$ (the mc superscript denotes the multi-cube neuron type).

The activation defining the general structure of this unit type can be seen in equation (4). In this formula q is the number of sub-cubes and d_j is the current sub-cube unit dimension taken from the vector $d = [d_1, d_2, \dots, d_q]$.

$$a^{mc} = \frac{1}{|w_{max}^{mc}|} \sum_{j=1}^{q} \frac{1}{2^{d_j}} \sum_{\mu=0}^{2^{d_j}-1} w_{\mu}^{mc} \prod_{i=1}^{d_j} (1 + \mu_i x_i) \tag{4}$$

The activation's output in (4) is then introduced as input to the activation function $g(a^{mc})$ which produces the output y. The complete multi-cube unit structure is visualized in Fig.4.

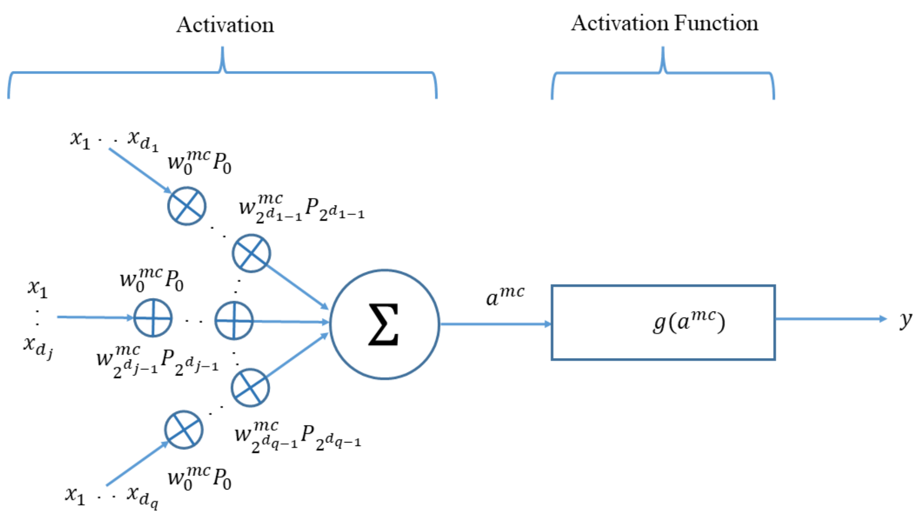

Figure 4: The multi-cube neuron structure. This figure visualizes the multi-cube neuron where an input vector containing n inputs is divided into series of q lower dimension sub-cubes. Each sub-cube can have different dimension $d = [d_1, d_2, \dots, d_q]$ contributing to a $\sum_{j=1}^{q} 2^{d_j}$ total number of weights. Each weight is multiplied by a corresponding probability and the normalized average of all weights is calculated. This procedure forms the neuron's activation which is inserted as input to the activation function. The latter is responsible for producing the neuron's output.

4. The ELM architecture

The main advantage of ELM is simplicity and speed since it treats SLNNs as linear systems where the hidden layer weights and thresholds are randomized, and the output layer weights are analytically calculated with the help of the Moore-Penrose pseudo-inverse. Also, it lacks user-defined parameters like the learning rate in back-propagation, which can affect the algorithm's convergence (Huang et al., 2004, 2006b).

The mathematical model of an ELM-trained SLNN is defined in formula (5).

$$\begin{bmatrix} g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) & \dots & g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \\ \vdots & \dots & \vdots \\ g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) & \dots & g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \end{bmatrix}_{N\times h} \cdot \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \tag{5}$$

In the above mathematical model:

- g is the activation function.
- $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.
- n are the neuron inputs.
- $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n\times h} \in \mathbb{R}^{n\times h}$ defines the input weights matrix.

- $\theta = [\theta_1 \ldots \theta_h] \in \mathbb{R}^h$ is the threshold vector.

- h is the hidden layer neurons number.

- N defines the number of input samples.

- $\beta^l = \begin{bmatrix} \beta^l_{1,1} & \ldots & \beta^l_{1,m} \\ \vdots & \ldots & \vdots \\ \beta^l_{h,1} & \ldots & \beta^l_{h,m} \end{bmatrix}_{h \times m} \in \mathbb{R}^{h \times m}$ is the output layer weights matrix.

- m is the output layer neurons number.

- $T = \begin{bmatrix} t_{1,1} & \ldots & t_{1,m} \\ \vdots & \ldots & \vdots \\ t_{N,1} & \ldots & t_{N,m} \end{bmatrix}_{N \times m}$ denotes the target output matrix.

In an ELM-trained network, the output neurons have the *identity* activation function ($g(u) = u$) and they lack a threshold.

The training process of ELM involves randomizing the hidden layer weights and thresholds, as seen in the first two lines of Algorithm 1. The following line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 4 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the neuron output weights matrix β^l where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

Algorithm 1 : ELM

$$1: w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n\times h}$$

The hidden layer neurons' weights matrix.

$2: \theta = [\theta_1, \dots, \theta_h]$

The hidden layer neurons' thresholds vector.

$3: H =$

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

The hidden layer matrix H.

$$4: T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m}$$

The target output matrix.

$5: \beta^l = H^\dagger T$

Calculation of the output weights matrix.

5. The higher-order ELM architecture

The current article proposes six ELM model variations for SLNNs with higher-order or multi-cube neurons in their hidden and output layers.

5.1. The higher-order ELM networks

The following section defines the mathematical models and pseudo-codes for three higher-order SLNN adaptations, which cover all possible combinations between higher-order and low-order unit types.

5.1.1. The higher-order/low-order ELM network

The higher-order/low-order SLNN contains higher-order neurons in the hidden layer and low-order neurons in the output layer. The mathematical model defining this network type is shown in formula (6).

$$\begin{bmatrix} g(a^c_{1,1}) & \dots & g(a^c_{1,h}) \\ \vdots & \dots & \vdots \\ g(a^c_{N,1}) & \dots & g(a^c_{N,h}) \end{bmatrix}_{N\times h} \cdot \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \tag{6}$$

In the above mathematical model:

- g is the activation function.
- $a^c = \begin{bmatrix} a^c_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T, \begin{bmatrix} w^c_{0,1} \\ \vdots \\ w^c_{2^n-1,1} \end{bmatrix}\right) & \dots & a^c_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T, \begin{bmatrix} w^c_{0,h} \\ \vdots \\ w^c_{2^n-1,h} \end{bmatrix}\right) \\ \vdots & \dots & \vdots \\ a^c_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T, \begin{bmatrix} w^c_{0,1} \\ \vdots \\ w^c_{2^n-1,1} \end{bmatrix}\right) & \dots & a^c_{N,h}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T, \begin{bmatrix} w^c_{0,h} \\ \vdots \\ w^c_{2^n-1,h} \end{bmatrix}\right) \end{bmatrix}_{N\times h} \in \mathbb{R}^{N\times h}$ is the cubic activation matrix.
- $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.
- n are the neuron inputs.

- $w^c = \begin{bmatrix} w^c_{0,1} & \dots & w^c_{0,h} \\ \vdots & \dots & \vdots \\ w^c_{2^n-1,1} & \dots & w^c_{2^n-1,h} \end{bmatrix}_{2^n \times h} \in \mathbb{R}^{2^n \times h}$ is the input weights matrix.
- h is the hidden layer neurons number.
- N defines the number of input samples.
- $\beta^l = \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h \times m} \in \mathbb{R}^{h \times m}$ is the output layer weights matrix.
- m is the output layer neurons number.
- $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$ denotes the target output matrix.

The ELM training process of an SLNN containing higher-order hidden layer and low-order output layer nodes begins by randomizing the hidden layer weights matrix having $2^n \times h$ weights as seen in line 1 of Algorithm 2. The following line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 3 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the linear neuron output weights matrix β^l where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

Algorithm 2 : Higher-Order/Low-Order ELM

$$1 : w^c = \begin{bmatrix} w^c_{0,1} & \dots & w^c_{0,h} \\ \vdots & \dots & \vdots \\ w^c_{2^n-1,1} & \dots & w^c_{2^n-1,h} \end{bmatrix}_{2^n \times h}$$

The hidden layer neurons' weights matrix.

$$2 : H = \begin{bmatrix} g(a^c_{1,1}) & \dots & g(a^c_{1,h}) \\ \vdots & \dots & \vdots \\ g(a^c_{N,1}) & \dots & g(a^c_{N,h}) \end{bmatrix}_{N \times h}$$

The hidden layer matrix H.

$$3 : T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$$

The target output matrix.

$4 : \beta^l = H^\dagger T$

Calculation of the output weights matrix.

5.1.2. The low-order/higher-order ELM network

The low-order/higher-order SLNN contains low-order units in the hidden layer and higher-order units in the output layer. The mathematical model defining this network type is shown in formula (7).

$$\begin{bmatrix} P_{1,0}\left(g(a^l_{1,1}),\dots,g(a^l_{1,h})\right) & \dots & P_{1,2^h-1}\left(g(a^l_{1,1}),\dots,g(a^l_{1,h})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a^l_{N,1}),\dots,g(a^l_{N,h})\right) & \dots & P_{N,2^h-1}\left(g(a^l_{N,1}),\dots,g(a^l_{N,h})\right) \end{bmatrix}_{N \times 2^h} \cdot \begin{bmatrix} \beta^c_{0,1} & \dots & \beta^c_{0,m} \\ \vdots & \dots & \vdots \\ \beta^c_{2^h-1,1} & \dots & \beta^c_{2^h-1,m} \end{bmatrix}_{2^h \times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m} \quad (7)$$

In the above mathematical model:

- P is the probability function.
- g is the activation function.

- $a^l =$
$$\begin{bmatrix} a^l_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1}\end{bmatrix} + \theta_1\right) & \dots & a^l_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h}\end{bmatrix} + \theta_h\right) \\ \vdots & \dots & \vdots \\ a^l_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1}\end{bmatrix} + \theta_1\right) & \dots & a^l_{N,h}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h}\end{bmatrix} + \theta_h\right) \end{bmatrix}_{N\times h} \in$$
$\mathbb{R}^{N\times h}$ is the linear activation matrix.

- $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.

- n are the neuron inputs.

- $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n\times h} \in \mathbb{R}^{n\times h}$ defines the input weights matrix.

- $\theta = [\theta_1 \dots \theta_h] \in \mathbb{R}^h$ is the threshold vector.

- h is the hidden layer neurons number.

- N defines the number of input samples.

- $\beta^c = \begin{bmatrix} \beta^c_{0,1} & \dots & \beta^c_{0,m} \\ \vdots & \dots & \vdots \\ \beta^c_{2^h-1,1} & \dots & \beta^c_{2^h-1,m} \end{bmatrix}_{2^h\times m} \in \mathbb{R}^{2^h\times m}$ is the output layer weights matrix.

- m is the output layer neurons number.

- $T = \begin{bmatrix} t_{11} & \dots & t_{1m} \\ \vdots & \dots & \vdots \\ t_{N1} & \dots & t_{Nm} \end{bmatrix}_{N\times m} \in \mathbb{R}^{N\times m}$ denotes the target output matrix.

The ELM training process of an SLNN containing low-order hidden layer and higher-order output layer nodes begins by randomizing the hidden layer weights and thresholds as seen in the first two lines of Algorithm 3.

Algorithm 3 : Low-Order/Higher-Order ELM

1 : $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n\times h}$

The hidden layer neurons' weights matrix.

2 : $\theta = [\theta_1, \dots, \theta_h]$

The hidden layer neurons' thresholds vector.

3 : $H =$

$$\begin{bmatrix} P_{1,0}\left(g(a^l_{1,1}),\dots,g(a^l_{1,h})\right) & \dots & P_{1,2^h-1}\left(g(a^l_{1,1}),\dots,g(a^l_{1,h})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a^l_{N,1}),\dots,g(a^l_{N,h})\right) & \dots & P_{N,2^h-1}\left(g(a^l_{N,1}),\dots,g(a^l_{N,h})\right) \end{bmatrix}_{N\times 2^h}$$

The hidden layer matrix H.

4 : $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m}$

The target output matrix.

5 : $\beta^c = H^\dagger T$

Calculation of the output weights matrix.

The next line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 4 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the cubic neuron output weights matrix β^c where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

5.1.3. The higher-order/higher-order ELM network

The higher-order/higher-order SLNN contains higher-order neurons in both hidden and output layers. The mathematical model defining this network type is shown in formula (8).

$$\begin{bmatrix} P_{1,0}\left(g(a^c_{1,1}),\ldots,g(a^c_{1,h})\right) & \ldots & P_{1,2^h-1}\left(g(a^c_{1,1}),\ldots,g(a^c_{1,h})\right) \\ \vdots & \ldots & \vdots \\ P_{N,0}\left(g(a^c_{N,1}),\ldots,g(a^c_{N,h})\right) & \ldots & P_{N,2^h-1}\left(g(a^c_{N,1}),\ldots,g(a^c_{N,h})\right) \end{bmatrix}_{N\times 2^h} \cdot \begin{bmatrix} \beta^c_{0,1} & \ldots & \beta^c_{0,m} \\ \vdots & \ldots & \vdots \\ \beta^c_{2^h-1,1} & \ldots & \beta^c_{2^h-1,m} \end{bmatrix}_{2^h\times m} = \begin{bmatrix} t_{1,1} & \ldots & t_{1,m} \\ \vdots & \ldots & \vdots \\ t_{N,1} & \ldots & t_{N,m} \end{bmatrix}_{N\times m} \tag{8}$$

In the above mathematical model:

- P is the probability function.
- g is the activation function.
- $a^c = \begin{bmatrix} a^c_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^c_{0,1} \\ \vdots \\ w^c_{2^n-1,1}\end{bmatrix}\right) & \ldots & a^c_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^c_{0,h} \\ \vdots \\ w^c_{2^n-1,h}\end{bmatrix}\right) \\ \vdots & \ldots & \vdots \\ a^c_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T, \begin{bmatrix} w^c_{0,1} \\ \vdots \\ w^c_{2^n-1,1}\end{bmatrix}\right) & \ldots & a^c_{N,h}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T, \begin{bmatrix} w^c_{0,h} \\ \vdots \\ w^c_{2^n-1,h}\end{bmatrix}\right) \end{bmatrix}_{N\times h} \in \mathbb{R}^{N\times h}$ is the cubic activation matrix.
- $x = \begin{bmatrix} x_{1,1} & \ldots & x_{1,n} \\ \vdots & \ldots & \vdots \\ x_{N,1} & \ldots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.
- n are the neuron inputs.

- $w^c = \begin{bmatrix} w^c_{0,1} & \dots & w^c_{0,h} \\ \vdots & \dots & \vdots \\ w^c_{2^n-1,1} & \dots & w^c_{2^n-1,h} \end{bmatrix}_{2^n \times h} \in \mathbb{R}^{2^n \times h}$ defines the input weights matrix.

- $\theta = [\theta_1 \dots \theta_h] \in \mathbb{R}^h$ is the threshold vector.

- h is the hidden layer neurons number.

- N defines the number of input samples.

- $\beta^c = \begin{bmatrix} \beta^c_{0,1} & \dots & \beta^c_{0,m} \\ \vdots & \dots & \vdots \\ \beta^c_{2^h-1,1} & \dots & \beta^c_{2^h-1,m} \end{bmatrix}_{2^h \times m} \in \mathbb{R}^{2^h \times m}$ is the output layer weights matrix.

- m is the output layer neurons number.

- $T = \begin{bmatrix} t_{11} & \dots & t_{1m} \\ \vdots & \dots & \vdots \\ t_{N1} & \dots & t_{Nm} \end{bmatrix}_{N \times m} \in \mathbb{R}^{N \times m}$ denotes the target output matrix.

The ELM training process of an SLNN containing higher-order hidden and output layer nodes begins by randomizing the hidden layer weights matrix having $2^n \times h$ weights, as seen in line 1 of Algorithm 4. The following line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 3 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the cubic neuron output weights matrix β^c where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

Algorithm 4 : Higher-Order/Higher-Order ELM

$$1: w^c = \begin{bmatrix} w^c_{0,1} & \dots & w^c_{0,h} \\ \vdots & \dots & \vdots \\ w^c_{2^n-1,1} & \dots & w^c_{2^n-1,h} \end{bmatrix}_{2^n \times h}$$

The hidden layer neurons' weights matrix.

$$2: H =$$

$$\begin{bmatrix} P_{1,0}\Big(g(a^c_{1,1}), \dots, g(a^c_{1,h})\Big) & \dots & P_{1,2^h-1}\Big(g(a^c_{1,1}), \dots, g(a^c_{1,h})\Big) \\ \vdots & \dots & \vdots \\ P_{N,0}\Big(g(a^c_{N,1}), \dots, g(a^c_{N,h})\Big) & \dots & P_{N,2^h-1}\Big(g(a^c_{N,1}), \dots, g(a^c_{N,h})\Big) \end{bmatrix}_{N \times 2^h}$$

The hidden layer matrix H.

$$3: T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$$

The target output matrix.

$$4: \beta^c = H^\dagger T$$

Calculation of the output weights matrix.

5.2. The multi-cube ELM networks

The following section defines the mathematical models and pseudo-codes for three multi-cube SLNN adaptations, which cover all possible combinations between multi-cube and low-order unit types.

5.2.1. The multi-cube/low-order ELM network

The multi-cube/low-order SLNN contains multi-cube neurons in the hidden layer and low-order neurons in the output layer. The mathematical model defining this network type is shown in formula (9).

$$\begin{bmatrix} g(a^{mc}_{1,1}) & \dots & g(a^{mc}_{1,h}) \\ \vdots & \dots & \vdots \\ g(a^{mc}_{N,1}) & \dots & g(a^{mc}_{N,h}) \end{bmatrix}_{N\times h} \cdot \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \tag{9}$$

In the above mathematical model:

- g is the activation function.

- $a^{mc} = \begin{bmatrix} a^{mc}_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1} \end{bmatrix}\right) & \dots & a^{mc}_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T, \begin{bmatrix} w^{c}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h} \end{bmatrix}\right) \\ \vdots & \dots & \vdots \\ a^{mc}_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1} \end{bmatrix}\right) & \dots & a^{mc}_{N,h}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h} \end{bmatrix}\right) \end{bmatrix}_{N\times h} \in \mathbb{R}^{N\times h}$ is the multi-cube activation matrix.

- $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.

- n are the neuron inputs.

- $w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{mc}_{p_h-1,h} \end{bmatrix}_{p_h\times h} \in \mathbb{R}^{p_h\times h}$ is the input weights matrix.

- p_h is the hidden layer weights number.

- h is the hidden layer neurons number.
- N defines the number of input samples.
- $\beta^l = \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h \times m} \in \mathbb{R}^{h \times m}$ is the output layer weights matrix.
- m is the output layer neurons number.
- $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$ denotes the target output matrix.

The ELM training process of an SLNN containing multi-cube hidden layer and low-order output layer nodes begins by randomizing the hidden layer weights matrix having $p_h \times h$ weights as seen in line 1 of Algorithm 5.

Algorithm 5 : Multi-Cube/Low-Order ELM

$$1 : w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{mc}_{p_h-1,h} \end{bmatrix}_{p_h \times h}$$

The hidden layer neurons' weights matrix.

$$2 : H = \begin{bmatrix} g(a^{mc}_{1,1}) & \dots & g(a^{mc}_{1,h}) \\ \vdots & \dots & \vdots \\ g(a^{mc}_{N,1}) & \dots & g(a^{mc}_{N,h}) \end{bmatrix}_{N \times h}$$

The hidden layer matrix H.

$$3 : T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$$

The target output matrix.

$4 : \beta^l = H^\dagger T$

Calculation of the output weights matrix.

The next line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 3 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the linear neuron output weights matrix β^l where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

5.2.2. The low-order/multi-cube ELM network

The low-order/multi-cube SLNN contains low-order units in the hidden layer and multi-cube units in the output layer. The mathematical model defining this network type is shown in formula (10).

$$\begin{bmatrix} P_{1,0}\left(g(a^l_{1,1}),\dots,g(a^l_{1,h})\right) & \dots & P_{1,p_o-1}\left(g(a^l_{1,1}),\dots,g(a^l_{1,h})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a^l_{N,1}),\dots,g(a^l_{N,h})\right) & \dots & P_{N,p_o-1}\left(g(a^l_{N,1}),\dots,g(a^l_{N,h})\right) \end{bmatrix}_{N\times p_o} \cdot \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m} \end{bmatrix}_{p_o\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \tag{10}$$

In the above mathematical model:

- P is the probability function.
- g is the activation function.
- $a^l =$

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

$\mathbb{R}^{N\times h}$ is the linear activation matrix.

- $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.

- n are the neuron inputs.

- $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n\times h} \in \mathbb{R}^{n\times h}$ defines the input weights matrix.

- $\theta = [\theta_1 \dots \theta_h] \in \mathbb{R}^h$ is the threshold vector.

- h is the hidden layer neurons number.

- p_o is the output layer weights number.

- N defines the number of input samples.

- $\beta^{mc} = \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m} \end{bmatrix}_{p_o\times m} \in \mathbb{R}^{p_o\times m}$ is the output layer weights matrix.

- m is the output layer neurons number.

- $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \in \mathbb{R}^{N\times m}$ denotes the target output matrix.

The ELM training process of an SLNN containing low-order hidden layer and multi-cube output layer nodes begins by randomizing the hidden layer weights and thresholds as seen in the first two lines of Algorithm 6.

Algorithm 6 : Low-Order/Multi-Cube ELM

$$1 : w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n \times h}$$

The hidden layer neurons' weights matrix.

$2 : \theta = [\theta_1, \dots, \theta_h]$

The hidden layer neurons' thresholds vector.

$3 : H =$

$$\begin{bmatrix} P_{1,0}\left(g(a^l_{1,1}), \dots, g(a^l_{1,h})\right) & \dots & P_{1,p_o-1}\left(g(a^l_{1,1}), \dots, g(a^l_{1,h})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a^l_{N,1}), \dots, g(a^l_{N,h})\right) & \dots & P_{N,p_o-1}\left(g(a^l_{N,1}), \dots, g(a^l_{N,h})\right) \end{bmatrix}_{N \times p_o}$$

The hidden layer matrix H.

$$4 : T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$$

The target output matrix.

$5 : \beta^{mc} = H^{\dagger} T$

Calculation of the output weights matrix.

The next line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 4 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the multi-cube neuron output weights matrix β^{mc} where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

5.2.3. The multi-cube/multi-cube ELM network

The multi-cube/multi-cube SLNN contains multi-cube neurons in both hidden and output layers. The mathematical model defining this network type is shown in formula (11).

$$\begin{bmatrix} P_{1,0}\left(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\right) & \dots & P_{1,p_o-1}\left(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\right) & \dots & P_{N,p_o-1}\left(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\right) \end{bmatrix}_{N\times p_o} \cdot \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m} \end{bmatrix}_{p_o\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \quad (11)$$

In the above mathematical model:

- P is the probability function.
- g is the activation function.
- $a^{mc} = \begin{bmatrix} a^{mc}_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1}\end{bmatrix}\right) & \dots & a^{mc}_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h}\end{bmatrix}\right) \\ \vdots & \dots & \vdots \\ a^{mc}_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1}\end{bmatrix}\right) & \dots & a^{mc}_{N,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h}\end{bmatrix}\right) \end{bmatrix}_{N\times h} \in \mathbb{R}^{N\times h}$ is the multi-cube activation matrix.
- $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ contains the input values.
- n are the neuron inputs.
- $w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{mc}_{p_h-1,h} \end{bmatrix}_{p_h\times h} \in \mathbb{R}^{p_h\times h}$ defines the input weights matrix.

- p_h is the hidden layer weights number.
- $\theta = [\theta_1 \dots \theta_h] \in \mathbb{R}^h$ is the threshold vector.
- h is the hidden layer neurons number.
- p_o is the output layer weights number.
- N defines the number of input samples.
- $\beta^{mc} = \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m} \end{bmatrix}_{p_o \times m} \in \mathbb{R}^{p_o \times m}$ is the output layer weights matrix.
- m is the output layer neurons number.
- $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m} \in \mathbb{R}^{N \times m}$ denotes the target output matrix.

The ELM training process of an SLNN containing multi-cube hidden and output layer nodes begins by randomizing the hidden layer weights matrix having $p_h \times h$ weights, as seen in line 1 of Algorithm 7. The following line creates the hidden layer output matrix H, which contains the hidden layer neuron outputs of every training pattern introduced to the model. Line 3 defines the target output matrix containing the expected network output values. The algorithm finishes by calculating the multi-cube neuron output weights matrix β^{mc} where the Moore-Penrose pseudo-inverse of the hidden layer matrix H is multiplied by the target output matrix T.

Algorithm 7 : Multi-Cube/Multi-Cube ELM

$$1 : w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{c}_{p_h-1,h} \end{bmatrix}_{n\times h}$$

The hidden layer neurons' weights matrix.

$2 : H =$

$$\begin{bmatrix} P_{1,0}\Big(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\Big) & \dots & P_{1,p_o-1}\Big(g(a^{mc}_{1,1}),\dots,g(a^{mc}_{1,h})\Big) \\ \vdots & \dots & \vdots \\ P_{N,0}\Big(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\Big) & \dots & P_{N,p_o-1}\Big(g(a^{mc}_{N,1}),\dots,g(a^{mc}_{N,h})\Big) \end{bmatrix}_{N\times p_o}$$

The hidden layer matrix H.

$$3 : T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m}$$

The target output matrix.

$4 : \beta^{mc} = H^{\dagger}T$

Calculation of the output weights matrix.

6. Experimental work and simulations

The six proposed ELM-variations are compared with traditional ELM in 15 classification and 10 regression real world datasets taken from the University of California, Irvine (UCI) machine learning repository (Dua & Graff, 2017) and `www.kaggle.com` website. The experimental part utilized binary and multi-class classification problems having different number of attributes as seen in Table 3. The first dataset is "balance scale" which was created for modeling psychological experimental results. Next, the "electroencephalogram (EEG) eye state" dataset contains data from a single continuous 117 second EEG measurement using the EEG neuro-headset from EMOTIV. "HIV-1 protease cleavage" is a list of octamers and a binary flag. The flag's value depends on weather the HIV-1 protease will cleave at the center (Rögnvaldsson et al., 2015). "Indoor user movement prediction from radio signal strength (RSS) data" contains real-life benchmark data from ambient assisted living applications (Bacciu et al., 2014). "Leaf" is a set of 36 leaf specimens with 15 input attributes which were reduced to 14 by removing the "specimen number" attribute.

Table 3: Classification Datasets Characteristics

Dataset	Inputs	Outputs	Entries
Balance Scale	4	3	625
EEG Eye State	14	2	14980
HIV-1 Protease Cleavage	8	2	746
Indoor User Movement Prediction from RSS Data	4	2	13197
Leaf	14	36	340
Mammographic Mass	5	2	961
Maternal Health Risk	6	3	1014
Nursery	8	5	12960
Page Blocks	10	5	5473
Qualitative Bankruptcy	6	2	250
Seeds	7	3	210
Speaker Accent Recognition	12	6	329
Statlog Heart	13	2	270
Wine	13	3	178
Yeast	8	10	1484

"Mammographic mass" contains entries for predicting the seriousness of a mammographic mass lesion utilizing the patient's age combined with breast imaging reporting and data system attributes (Elter et al., 2007). This dataset contained missing values which were replaced with the average values taken from the available data. "Maternal health risk" has entries taken from different health institutions in Bangladesh using a risk monitoring system. "Nursery" dataset was created from a hierarchical decision model for ranking nursery schools' applications. "Page blocks" contains entries for classifying all page layout blocks from a detected document during a segmentation process. "Qualitative bankruptcy" has qualitative input attributes for predicting bankruptcy. “Seeds” has geometrical properties of kernels taken from three wheat types. “Speaker accent recognition” contains data for accent detection and recognition taken from six countries. “Statlog Heart” is a heart disease database. “Wine” dataset contains chemical analysis data for determining wines’ origin while “yeast” has attributes for predicting the cellular localization sites of proteins.

The characteristics of the 10 regression datasets are summarized in Table 4.

Table 4: Regression Datasets Characteristics

Dataset	**Inputs**	**Outputs**	**Entries**
Airfoil Self Noise	5	1	1503
Auto MPG	8	1	398
California Housing Prices	9	1	20640
Carbon Nanotubes	5	3	10721
Combined Cycle Power Plant	4	1	9568
Concrete Compressive Strength	8	1	1030
Concrete Slump Test	7	3	103
QSAR Fish Toxicity	6	1	908
Synchronous Machine	4	1	557
Yacht Hydrodynamics	6	1	308

The first dataset is “airfoil self-noise” which contains data from aerodynamic and acoustic tests. “Auto miles per gallon (MPG)” dataset has data regarding city-cycle fuel consumption. “California housing prices” has the median house prices taken during 1990 in various California areas. This dataset contained missing values which were replaced with the average values taken from the available data. “Carbon nanotubes” contains initial and computed atomic coordinates from carbon nanotubes (Acı & Avcı, 2016; Aci et al., 2017). “Combined cycle power plant” has data points gathered from a plant working on full load during six years (Kaya et al., 2012; Tüfekci, 2014). “Concrete compressive strength” dataset contains attributes for predicting the compressive strength of concrete (Yeh, 1998). On the other hand, the “concrete slump test” dataset contains entries for predicting its slump flow (Yeh, 2007). "Quantitative structure-activity relationship (QSAR) fish toxicity" has data for predicting quantitative acute aquatic toxicity of fathead minnow fish (Cassotti et al., 2015). "Synchronous machine" contains real-time data from an experimental set (Kahraman et al., 2012; Kahraman, 2014). Finally, the "yacht hydrodynamics" dataset contains entries for predicting sailing yachts' hydrodynamic performance. The prediction algorithm receives as input the yachts' dimensions and velocity.

6.1. Parameter details

The experiments were run using the MATLAB 2017a environment and the user defined parameters are summarized in Table 5.

Table 5: Experiment Settings

Parameter Name	**Symbol**	**Values/Types**
Linear Neuron Weights	w^l	$[-1,1]^n, n \in \mathbb{N}^*$
Cubic Neuron Weights	w^c	$[-1,1]^{2^n}, n \in \mathbb{N}^*$
Multi-Cube Neuron Weights	w^{mc}	$[-1,1]^{p_{no}}, p_{no} \in \mathbb{N}^*$
Threshold	θ	$[-1,1]$
Inputs	x	$[-1,1]^n, n \in \mathbb{N}^*$
Activation Function	g	*sigmoid*
Hidden Layer Neurons No	h	10
Folds No	k	5
Experiment Sets	$expNo$	10

The linear, cubic, and multi-cube neuron weight values were chosen randomly from the uniform distribution. Moreover, they were restricted inside the $[-1,1]$ interval along with the linear unit type threshold. The input datasets were also normalized into the $[-1,1]$ interval approximately by dividing each input's attribute with their corresponding maximum absolute attribute value. The experimental part used a fixed hidden layer with ten nodes and the *sigmoid* function as the transfer function for all methods. The 5-fold cross-validation method was adopted, and all experiment runs were repeated ten times with different random values for the hidden node weights and thresholds. This experimental design was applied to avoid potential bias due to the random initialization of the hidden weights and thresholds. Finally, the MATLAB implementation for all the higher-order and multi-cube neuron combinations and the classic ELM algorithm (seven methods in total) are available for download in `https://github.com/bchristou1/HigherOrderELM`.

6.2. Classification problems

The experimental results in terms of classification accuracy (*acc*) from the comparison of ELM with the three cubic neuron methods are summarized in Table 6.

Table 6: Classification Results of ELM and the Three Higher-Order Neuron Networks

Dataset	**ELM**	**Cubic/ Linear**	**Linear/ Cubic**	**Cubic/ Cubic**
Balance Scale	85.23%	51.92%	**89.33%**	47.07%
EEG Eye State	57.61%	**69.77%**	57.90%	66.66%
HIV-1 Protease Cleavage	64.15%	60.22%	**67.37%**	62.93%
Indoor User Movement Prediction from RSS Data	65.66%	71.67%	64.92%	**72.11%**
Leaf	29.82%	28.88%	36.41%	**56.32%**
Mammographic Mass	78.53%	73.54%	**79.39%**	74.95%
Maternal Health Risk	61.94%	75.52%	62.74%	**76.74%**
Nursery	59.97%	83.73%	73.86%	**91.29%**
Page Blocks	92.14%	95.14%	92.19%	**95.85%**
Qualitative Bankruptcy	94.64%	94.32%	99.24%	**99.68%**
Seeds	92.48%	79.14%	**96.10%**	60.71%
Speaker Accent Recognition	54.76%	32.62%	**56.62%**	42.72%
Statlog Heart	70.96%	58.70%	**77.89%**	59.33%
Wine	83.61%	59.13%	**91.96%**	67.20%
Yeast	52.00%	50.21%	**54.85%**	54.03%

It is shown that in all 15 cases, at least one higher-order SLNN managed to get higher classification accuracy than traditional ELM (they are marked with bold font). The classification accuracy is calculated using formula 12 where k denotes the number of folds and (err) is the number of miss-classified patterns (p_{pat}).

$$acc = \frac{1}{k} \sum_{i=1}^{k} \left(1 - \frac{err}{p_{pat}} \right) \tag{12}$$

The significance of these results was tested using the Wilcoxon signed-rank test, a non-parametric statistical hypothesis test. It is utilized to evaluate a population's location based on a data sample or compare two populations' locations by utilizing two matching samples. The Wilcoxon signed-rank test's outcome is a p value. If this value is below a specific threshold (usually 5%), it can be concluded that the samples are from different populations (Conover, 1999). The p values from comparing ELM with the best higher-order-based SLNN are visualized in Table 7.

Table 7: Wilcoxon Signed-Rank Test Classification Results for Higher-Order Neuron Networks

Dataset	**Wilcoxon signed-rank test** $\boldsymbol{p}$ **value**
Balance Scale	0.0000000258
EEG Eye State	0.0000000008
HIV-1 Protease Cleavage	0.0001137480
Indoor User Movement Prediction from RSS Data	0.0000000008
Leaf	0.0000000007
Mammographic Mass	0.0264583076
Maternal Health Risk	0.0000000008
Nursery	0.0000000008
Page Blocks	0.0000000008
Qualitative Bankruptcy	0.0000000607
Seeds	0.0000013999
Speaker Accent Recognition	0.0363540214
Statlog Heart	0.0000016257
Wine	0.0000254932
Yeast	0.0000038329

The two sample vectors introduced as input to the Wilcoxon signed-rank test contained 50 values (5 folds × 10 experiment repeats). It is shown from the p values in Table 7 that in all cases, they were below the 5% threshold, indicating that the comparison results are statistically significant.

The experimental results in terms of classification accuracy (acc) from the comparison of ELM with the three multi-cube neuron methods are summarized in Table 8.

Table 8: Classification Results of ELM and the Three Multi-Cube Neuron Networks

Dataset	**ELM**	**MultiCube/ Linear**	**Linear/ MultiCube**	**MultiCube/ MultiCube**
Balance Scale	85.23%	86.48%	89.73%	**90.00%**
EEG Eye State	57.61%	58.31%	58.18%	**58.71%**
HIV-1 Protease Cleavage	64.15%	66.06%	68.01%	**69.09%**
Indoor User Movement Prediction from RSS Data	65.66%	65.79%	65.81%	**66.42%**
Leaf	29.82%	34.15%	42.38%	**45.56%**
Mammographic Mass	78.53%	78.85%	80.32%	**80.87%**
Maternal Health Risk	61.94%	62.08%	64.15%	**64.88%**
Nursery	59.97%	63.29%	85.07%	**85.94%**
Page Blocks	92.14%	**92.45%**	92.39%	**92.45%**
Qualitative Bankruptcy	94.64%	96.04%	**99.56%**	**99.56%**
Seeds	92.48%	94.24%	**96.76%**	96.43%
Speaker Accent Recognition	54.76%	57.48%	59.68%	**61.12%**
Statlog Heart	70.96%	73.81%	82.00%	**82.81%**
Wine	83.61%	85.48%	94.99%	**95.22%**
Yeast	52.00%	53.59%	55.70%	**56.11%**

It is shown that in all 15 cases, at least one multi-cube SLNN got higher classification accuracy than traditional ELM (marked with bold font).

The significance of these results was tested using the Wilcoxon signed-rank test. The p values from comparing ELM with the best multi-cube-based SLNN are visualized in Table 9.

Table 9: Wilcoxon Signed-Rank Test Classification Results for Multi-Cube Neuron Networks

Dataset	**Wilcoxon signed-rank test p value**
Balance Scale	0.0000000074
EEG Eye State	0.0000166316
HIV-1 Protease Cleavage	0.0000000845
Indoor User Movement Prediction from RSS Data	0.0042372699
Leaf	0.0000000007
Mammographic Mass	0.0000012946
Maternal Health Risk	0.0000004563
Nursery	0.0000000008
Page Blocks	0.0198564122
Qualitative Bankruptcy	0.0000000309
Seeds	0.0000001533
speaker Accent Recognition	0.0000001847
Statlog Heart	0.0000000014
Wine	0.0000000176
Yeast	0.0000000036

It is shown from the p values in Table 9 that in all cases, they were below the 5% threshold, indicating that the comparison results are statistically significant.

6.3. Regression problems

The experimental results in terms of mean square error (MSE) from the comparison of ELM with the three cubic neuron methods are summarized in Table 10.

Table 10: Regression Results of ELM and the Three Higher-Order Neuron Networks

Dataset	ELM	Cubic/ Linear	Linear/ Cubic	Cubic/ Cubic
Airfoil Self Noise	0.0045706201	0.0045706201	2.3101010533	**0.0010542772**
Auto MPG	0.0114402686	8862.0611659	**0.0051611919**	21.354901549
California Housing Prices	0.0284458575	2.3699995426	**0.0255194123**	0.3012180644
Carbon Nanotubes	0.0322716299	0.0039305962	0.0049386383	**0.0000708821**
Combined Cycle Power Plant	0.0002264695	0.0000679103	0.0000758965	**0.0000658865**
Concrete Compressive Strength	0.0280140475	9.1773606812	**0.0188405222**	0.2306207118
Concrete Slump Test	0.0491058296	655.05887151	**0.0366121605**	3.0171044070
QSAR Fish Toxicity	0.0139586256	35640211.855	**0.0105666324**	2448.4422511
Synchronous Machine	0.0003527827	0.0000000031	0.0000006875	**0.0000000001**
Yacht Hydrodynamics	0.0207609732	0.0022851373	0.0246121941	**0.0002876932**

It is shown that in all 10 cases, at least one higher-order SLNN managed to get lower MSE than traditional ELM (they are marked with bold font). The MSE is calculated using formula 13 where k denotes the number of folds, p_{pat} is the number of input patterns, t_j^i is the current target value and y_j^i is the current network output value.

$$MSE = \frac{1}{kp_{pat}} \sum_{j=1}^{k} \left(\sum_{i=1}^{p_{pat}} (t_j^i - y_j^i)^2 \right) \tag{13}$$

The significance of these results was tested using the Wilcoxon signed-rank test. The p values from comparing ELM with the best higher-order-based SLNN are visualized in Table 11.

Table 11: Wilcoxon Signed-Rank Test Regression Results for Higher-Order Neuron Networks

Dataset	**Wilcoxon signed-rank test** ***p* value**
Airfoil Self Noise	0.0000000015
Auto MPG	0.0000000277
California Housing Prices	0.0003233387
Carbon Nanotubes	0.0000000008
Combined Cycle Power Plant	0.0000000008
Concrete Compressive Strength	0.0000038542
Concrete Slump Test	0.0000087755
QSAR Fish Toxicity	0.0000000385
Synchronous Machine	0.0000000008
Yacht Hydrodynamics	0.0000000008

It is shown from the p values in Table 11 that in all cases, they were below the 5% threshold, indicating that the comparison results are statistically significant.

The experimental results in terms of MSE from the comparison of ELM with the three multi-cube neuron methods are summarized in Table 12. It is shown that in all 10 cases, at least one multi-cube SLNN got lower MSE than traditional ELM (marked with bold font).

Table 12: Regression Results of ELM and the Three Higher-Order Neuron Networks

Dataset	**ELM**	**MultiCube/ Linear**	**Linear/ MultiCube**	**MultiCube/ MultiCube**
Airfoil Self Noise	0.0045706201	0.0012622568	**0.0010796858**	0.0010833991
Auto MPG	0.0114402686	0.0061519741	0.0047101989	**0.0045328270**
California Housing Prices	0.0284458575	0.0257346565	0.0225020954	**0.0216683755**
Carbon Nanotubes	0.0322716299	0.0115985326	0.0002601128	**0.0002010187**
Combined Cycle Power Plant	0.0002264695	0.0000773334	0.0000756647	**0.0000752948**
Concrete Compressive Strength	0.0280140475	0.0178677258	0.0152020538	**0.0149246718**
Concrete Slump Test	0.0491058296	0.0427722855	0.0337404423	**0.0318368830**
QSAR Fish Toxicity	0.0139586256	0.0107685711	**0.0099209195**	0.0099418562
Synchronous Machine	0.0003527827	0.0000218708	0.0000000497	**0.0000000164**
Yacht Hydrodynamics	0.0207609732	0.0201042234	0.0134259389	**0.0123262905**

The significance of these results was tested using the Wilcoxon signed-rank test. The p values from comparing ELM with the best multi-cube-based SLNN are visualized in Table 13.

Table 13: Wilcoxon Signed-Rank Test Classification Results for Multi-Cube Neuron Networks

Dataset	**Wilcoxon signed-rank test *p* value**
Airfoil Self Noise	0.0000000012
Auto MPG	0.0000000015
California Housing Prices	0.0000000012
Carbon Nanotubes	0.0000000008
Combined Cycle Power Plant	0.0000000013
Concrete Compressive Strength	0.0000000019
Concrete Slump Test	0.0000000054
QSAR Fish Toxicity	0.0000000025
Synchronous Machine	0.0000000008
Yacht Hydrodynamics	0.0000038542

It is shown from the p values in Table 13 that in all cases, they were below the 5% threshold, indicating that the comparison results are statistically significant.

7. Discussion

The experimental results from comparing the higher-order ELM variants with traditional ELM revealed that in the classification problems, the linear-cubic and cubic/cubic networks managed to get the highest accuracy values in 14/15 datasets (8/15 for linear/cubic and 6/15 for cubic/cubic networks). These results were also consistent in the regression problems where 50% of the datasets (5/10) were linear/cubic, and the other 50% were cubic/cubic networks. Considering these findings, it is evident that the cubic neuron(s) in the output layer can contribute to a significant increase in the SLNN's generalization ability. The linear neuron(s) in the low-order SLNNs trained by the classic ELM algorithm provides a simple weighted aggregation of the hidden layer neurons' outputs in contrast to the more complex probabilistic model adopted by the higher-order neurons.

The comparison between the multi-cube SLNN variants with traditional ELM revealed that the multi-cube/multi-cube networks in 12/15 datasets had the best accuracy in the classification problems. Two datasets ("page blocks" and "qualitative bankruptcy") had the same accuracy values with the multi-cube/linear and linear/multi-cube networks. Only in the "seeds" dataset did the linear/multi-cube networks get a higher accuracy value than the multi-cube/multi-cube network. These results were also consistent in the regression problems where the multi-cube/multi-cube networks had the lowest MSE in 8/10 datasets. In comparison, the lowest MSE from the other two datasets was achieved by the linear/multi-cube networks. Considering these findings, it is evident that an ELM-trained SLNN with multi-cube units having one-dimension sub-cubes has lower MSE in most cases than SLNNs with low-order units in all layers. This finding shows that having multi-cube units with one-dimension sub-cubes in all layers increases the network's generalization ability.

8. Conclusion

The present article generalizes the ELM training algorithm for work with SLNNs having the cubic and multi-cube neurons proposed by (Gurney, 1989). A total number of six algorithms have been presented, which cover all combinations between cubic/low-order and multi-cube/low-order units.

The experimental results section tested the proposed six algorithms with traditional ELM in a series of classification and regression problems. The experimental results revealed that at least one cubic and multi-cube network had better generalization in all cases. Also, in most cases, using higher-order/multi-cube units in the output layer created networks with better generalization ability. This finding indicates that the probabilistic nature of higher-order neurons increases the classification accuracy or reduces the MSE in regression problems compared to the single aggregation provided by the low-order neurons.

Funding

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

References

Acı, M., & Avcı, M. (2016). Artificial neural network approach for atomic coordinate prediction of carbon nanotubes. *Applied Physics A*, *122*, 1–14.

Aci, M., Avci, M. et al. (2017). Reducing simulation duration of carbon nanotube using support vector regression method. *Journal of the Faculty of Engineering and Architecture of Gazi University*, *32*.

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

Cai, W., Yang, J., Yu, Y., Song, Y., Zhou, T., & Qin, J. (2020). PSO-ELM: A hybrid learning model for short-term traffic flow forecasting. *IEEE access*, *8*, 6505–6514.

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

Cao, J., Lin, Z., Huang, G.-B., & Liu, N. (2012b). Voting based extreme learning machine. *Information Sciences*, *185*, 66–77.

Cao, J., Zhang, K., Yong, H., Lai, X., Chen, B., & Lin, Z. (2018). Extreme learning machine with affine transformation inputs in an activation function. *IEEE transactions on neural networks and learning systems*, *30*, 2093–2107.

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

Cassotti, M., Ballabio, D., Todeschini, R., & Consonni, V. (2015). A similarity-based QSAR model for predicting acute toxicity towards the fathead minnow (pimephales promelas). *SAR and QSAR in Environmental Research*, *26*, 217–243.

Chen, H., & Yao, X. (2009). Regularized negative correlation learning for neural network ensembles. *IEEE Transactions on Neural Networks*, *20*, 1962–1979.

Chen, X., Hai, B., & Wang, L. (2021). Multiobjective parameters optimization of extreme learning machine based on MOEA/D. In *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)* (pp. 321–324). IEEE.

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

Christou, V., Ntritsos, G., Tzallas, A. T., Tsipouras, M. G., & Giannakeas, N. (2020). Self-adaptive hybrid extreme learning machine for heterogeneous neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN48605.2020.9207608.

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

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

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

Cui, Z., Huang, B., Dou, H., Tan, G., Zheng, S., & Zhou, T. (2022). GSA-ELM: A hybrid learning model for short-term traffic flow forecasting. *IET Intelligent Transport Systems*, *16*, 41–52.

Deng, W., Zheng, Q., & Zhang, K. (2013). Reduced kernel extreme learning machine. In *Proceedings of the 8th international conference on computer recognition systems CORES 2013* (pp. 63–69). Springer.

Deng, W.-Y., Ong, Y.-S., & Zheng, Q.-H. (2016). A fast reduced kernel extreme learning machine. *Neural Networks*, *76*, 29–38.

Deng, W.-Y., Zheng, Q.-H., & Wang, Z.-M. (2014). Cross-person activity recognition using reduced kernel extreme learning machine. *Neural Networks*, *53*, 1–7.

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

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

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*, 4164–4172.

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

Gurney, K. (2018). *An introduction to neural networks*. CRC press.

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

Huang, G.-B., & Chen, L. (2007). Convex incremental extreme learning machine. *Neurocomputing*, *70*, 3056–3062.

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

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

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

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

Kahraman, H., Bayindir, R., & Sagiroglu, S. (2012). A new approach to predict the excitation current and parameter weightings of synchronous machines based on genetic algorithm-based k-NN estimator. *Energy Conversion and Management*, *64*, 129–138.

Kahraman, H. T. (2014). Metaheuristic linear modeling technique for estimating the excitation current of a synchronous motor. *Turkish Journal of Electrical Engineering and Computer Sciences*, *22*, 1637–1652.

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

Liu, B., Chen, G., Lin, H.-C., Zhang, W., & Liu, J. (2021). Prediction of IGBT junction temperature using improved cuckoo search-based extreme learning machine. *Microelectronics Reliability*, *124*, 114267.

Liu, Y., & Yao, X. (1997). Negatively correlated neural networks can produce best ensembles. *Australian journal of intelligent information processing systems*, *4*, 176–185.

Lu, F., Wu, J., Huang, J., & Qiu, X. (2019). Restricted-boltzmann-based extreme learning machine for gas path fault diagnosis of turbofan engine. *IEEE Transactions on Industrial Informatics*, *16*, 959–968.

Lu, H., Du, B., Liu, J., Xia, H., & Yeap, W. K. (2017). A kernel extreme learning machine algorithm based on improved particle swam optimization. *Memetic Computing*, *9*, 121–128.

Luo, J., Vong, C.-M., & Wong, P.-K. (2013). Sparse Bayesian extreme learning machine for multi-classification. *IEEE Transactions on Neural Networks and Learning Systems*, *25*, 836–843.

Luo, J., Wong, C.-M., & Vong, C.-M. (2021). Multinomial Bayesian extreme learning machine for sparse and accurate classification model. *Neurocomputing*, *423*, 24–33.

Martínez-Martínez, J. M., Escandell-Montero, P., Soria-Olivas, E., Martín-Guerrero, J. D., Magdalena-Benedito, R., & Gómez-Sanchis, J. (2011). Regularized extreme learning machine for regression problems. *Neurocomputing*, *74*, 3716–3721.

Mel, B., & Koch, C. (1989). Sigma-pi learning: On radial basis functions and cortical associative learning. *Advances in neural information processing systems*, *2*.

Perales-González, C., Fernández-Navarro, F., Pérez-Rodríguez, J., & Carbonero-Ruz, M. (2021). Negative correlation hidden layer for the extreme learning machine. *Applied Soft Computing*, *109*, 107482.

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

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

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

Sevinc, E. (2019). A novel evolutionary algorithm for data classification problem with extreme learning machines. *IEEE Access*, *7*, 122419–122427.

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

Sulaiman, S., Jeyanthy, P. A., Devaraj, D., & Shihabudheen, K. (2022). A novel hybrid short-term electricity forecasting technique for residential loads using empirical mode decomposition and extreme learning machines. *Computers & Electrical Engineering*, *98*, 107663.

Tang, Y., & Li, C. (2021). An online network intrusion detection model based on improved regularized extreme learning machine. *IEEE Access*, *9*, 94826–94844.

Tian, X., Jiao, W., Liu, T., Ren, L., & Song, B. (2021). Leakage detection of low-pressure gas distribution pipeline system based on linear fitting and extreme learning machine. *International Journal of Pressure Vessels and Piping*, *194*, 104553.

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

Wang, J., Lu, S., Wang, S.-H., & Zhang, Y.-D. (2021). A review on extreme learning machine. *Multimedia Tools and Applications*, (pp. 1–50).

Wong, K. I., Vong, C. M., Wong, P. K., & Luo, J. (2015). Sparse Bayesian extreme learning machine and its application to biofuel engine performance prediction. *Neurocomputing*, *149*, 397–404.

Wu, Y., Zhang, Y., Liu, X., Cai, Z., & Cai, Y. (2018). A multiobjective optimization-based sparse extreme learning machine algorithm. *Neurocomputing*, *317*, 88–100.

Xing, H.-J., & Wang, X.-M. (2013). Training extreme learning machine via regularized correntropy criterion. *Neural Computing and Applications*, *23*, 1977–1986.

Xu, Z., Yao, M., Wu, Z., & Dai, W. (2016). Incremental regularized extreme learning machine and it's enhancement. *Neurocomputing*, *174*, 134–142.

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

Yeh, I.-C. (2007). Modeling slump flow of concrete using second-order regressions and artificial neural networks. *Cement and concrete composites*, *29*, 474–480.

Zhang, K., & Luo, M. (2015). Outlier-robust extreme learning machine for regression problems. *Neurocomputing*, *151*, 1519–1527.

Zhang, Q., Li, H., Liu, C., & Hu, W. (2013). A new extreme learning machine optimized by firefly algorithm. In *2013 Sixth International Symposium on Computational Intelligence and Design* (pp. 133–136). IEEE volume 2.

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

Zhang, Z., Hou, R., & Yang, J. (2020). Detection of social network spam based on improved extreme learning machine. *IEEE Access*, *8*, 112003–112014.

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