A centered bithreshold neural network regressor

Vladyslav Kotsovsky

State University "Uzhhorod National University", Narodna Square 3, 88000, Uzhhorod, Ukraine

Abstract

The application of the multithreshold approach in the design of neural network regressors is considered in the paper. A new model of a hybrid 2-layer neural network is proposed whose hidden layer consists of centered bithreshold neurons and employs the softmax activation method. This model is intended for solving regression tasks. It is a modification of the earlier model of bithreshold neural network regressor. The proposed model reduces two main observed drawbacks of the basic model by using in the hidden layer a new model of centered bithreshold neural unit with continuous output values instead of classical binary-valued bithreshold neurons. A supervised algorithm was designed for the synthesis of centered bithreshold neural network regressor. It uses one hyperparameter—the number of discretization levels. The proposed algorithm usually yields a smaller network with higher accuracy of prediction compared to the basic bithreshold network. The performance of the proposed model is compared with that of popular machine learning regressors on both synthetic and real-world datasets. The simulation results collected for two benchmark datasets confirm that synthesized neural network is suitable for making predictions for datasets of different sizes and the dimensionalities of the feature space.

Keywords

bithreshold neuron, neural network, regressor, computational intelligence

1. Introduction

There is no need to emphasize once again that a wide variety of models and systems have been recently developed in the field of computational intelligence [1], many of which are grounded in neural-based concepts [1–3] applied in both system design [3] and training or synthesis of models [4, 5]. The history of neural networks (NN) and neural computation offers numerous examples of successful applications in artificial intelligence [6], demonstrating their ability to solve a wide range of scientific challenges [7, 8] as well as real-world problems [9, 10]. One notable approach in machine learning is the *multithreshold method*, which involves the use of neural units with multiple threshold levels [11]. In first attempts based on this method, classical step activation functions [3] (such as Heaviside or sign functions) are replaced with functions that incorporate two or more thresholds [12, 13]. Modern applications concern continuous modification of multithreshold activations, which can overperform modern activations such as ReLU [3], Swish [14], ELU [3, 15], SELU [15, 16], GELU [17], Mish [18], etc, by using the smoothing of discrete multithreshold activations [19].

Basically, the multithreshold approach was designed for the classification of multidimensional patterns [20]. Binary valued bithreshold neurons as well as their multithreshold generalizations are capable to increase the performance of NN [21] by the proper use of the enhanced capacity of multithreshold units compared to single-threshold ones. Moreover, the application of multithreshold NNs in pattern classification can significantly reduce the network complexity [22].

This focus on classification is understandable, because multithreshold neurons have discrete range of output values [21, 22]. Another important computational intelligence task, namely regression, remained long time outside the scope of the application of multithreshold neural network approach. Until recently, rare examples were known of using multithreshold techniques for solving regression problems.

⋆*CIAW-2025: Computational Intelligence Application Workshop, September 26–27, 2025, Lviv, Ukraine.*

1* Corresponding author.

vladyslav.kotsovsky@uzhnu.edu.ua (V. Kotsovsky)

000-0002-7045-7933 (V. Kotsovsky)

However, there are precedents in computational intelligence of transformation of discrete models in order to proceed with continuous outputs [3]. For instance, several classification models were adapted to perform as regressor tasks (e.g., *k*-nearest neighbors, support vector machines and decision trees) [3, 23, 24]. This idea was used in author's last year's paper (coauthored with A. Batyuk) devoted to function approximation using multithreshold neural units [25]. But results of additional simulations showed that regressor performed poorly as the dimensionality of input feature space increased [19]. The *objective of current research* is to explain this performance decay of the basic model of bithreshold NN regressor from [25] and to improve the initial approach in the design of multithreshold regressor by reducing the drawbacks of this model in order to predict continuous target values more precisely.

2. Related works

Neural computation based on gates with multiple thresholds have been studied for many years [4]. Early research on binary-valued multithreshold neural units was initiated by D. R. Haring [26] and further developed by other researchers such as R. Spann, C. W. Mow, N. N. Necula, Y. T. Yen, and S. Ghosh [27] within the framework of multithreshold logic (see [22] for a detailed review and references).

As noted in [8], the primary motivation for adopting multithreshold units was the hypothesis that additional thresholds would significantly enhance the computational power of neural models. This hypothesis was later validated in the next series of work, which demonstrated the superiority of multithreshold units over single-threshold ones [20]. The comparison used the counting of the number of distinct dichotomies that a neural unit could implement on a finite set of *n*-dimensional input vectors [28]. However, early contributions to multithreshold logic did not yield convenient training algorithms for such systems. The only notable attempt was a heuristic proposed by Takiyama and later refined by Anthony [29], though it lacked guarantees of correctness and convergence. The difficulties in learning binary-valued multithreshold models were later attributed to inherent computational complexity: even the learning of systems with just two thresholds was shown to be a NP-complete problem [22].

Interest in multithreshold systems reemerged approximately two decades later, boosted by the development of multi-valued multithreshold neurons and the formalization of multi-valued multithreshold functions [29, 30]. This marked the beginning of a second wave of research, with contributions from Z. Obradović and I. Parberry [21], Ngom [22], M. Anthony [29], and I. Prokíc [30]. Their work not only analyzed the theoretical properties of these systems but also introduced an online learning algorithm for multithreshold units, based on an incremental correction rule [21, 29]. This line of research continued in [22], where faster online and offline variants of the learning algorithm were proposed, incorporating relaxation methods for solving computational tasks more efficiently. Recent progress in the use of multithreshold systems has been reported in [31, 32]. Paper [32] explored NN architectures where the first hidden layer is composed of multithreshold units. Such architectures proved effective in both binary and multiclass classification tasks, enabled by the development of efficient synthesis algorithms [32]. Further enhancements were achieved by adding a secondary hidden layer consisting of bithreshold units, winner-take-all (WTA) units, and standard single-threshold neurons, resulting in a more powerful hybrid architecture [8]. As emphasized in [5], bithreshold neural networks represent a promising direction in artificial intelligence and neural computation, offering greater compactness and computational power compared to traditional single-threshold systems.

The choice of two thresholds is particularly advantageous, as it strikes a balance between increased expressive power and manageable synthesis complexity for the hidden layers [8]. Nevertheless, in more recent works binary-valued multithreshold units with an arbitrary number of thresholds were employed instead of just two [25] as well as multi-valued neuron were used instead of binary-valued ones [32]. Initial attempts at learning such generalized models were also explored in [22], with a focus on their application to pattern classification tasks.

It should be noted that multithreshold approach in neural computation does not consists only in the development of model of neural units and networks as well as learning algorithms for such systems. It also comprises the design of hardware implementation of multithreshold devices proposed in papers of T. Gowda [33], M. Nikodem [34] and O. Vyshnevskyy [35].

Recent research devoted to the application of multithreshold approach in regression are also worthy to be mentioned. E.g., J. Li et al. [36] and J. Wang et al. [37] developed multithreshold statistical models and gave their application in medicine, A. Reinke et al. [38] considered multithreshold metrics. The model of NN regressor employed binary-valued bithreshold units was designed by the author and A. Batyuk in [25], whereas its above-mentioned continuous-valued generalization within the gradient-based learning framework was proposed in [19].

3. Models and methods

As mentioned in introduction, the paper is devoted to the improvement of the model of bithreshold regressor proposed in [25]. Let us remember the main principle of the design of this model in order to highlight its serious weaknesses, which were exposed by its application in solving regression tasks on both synthetic [23] and real-world [39] datasets in the case when the dimension of the tasks grows. This will make it easier for us to understand the causes of such drawbacks and show the ways of possible corrections.

3.1. Basic model of neural network regressor with bithreshold hidden layer

3.1.1. Architecture and synthesis of hybrid neural network regressor

The bithreshold NN regressor from [25] uses binary-valued bithreshold nodes in the network hidden layer. Namely, a binary-valued *bithreshold neuron* with *weight vector* $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbf{R}^n$ and thresholds $t_1, t_2 \in \mathbf{R}$ $(t_1 < t_2)$ is a computation unit whose single output y is obtained after the application of the following activation function

$$f_{t_1,t_2}(s) = \begin{cases} 1, & \text{if } t_1 \le s < t_2, \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

to the weighted sum $\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + \ldots + w_n x_n$. It is natural to call the half-interval $[t_1, t_2)$ an *activation range* of function (1). Note that (1) is the simplest kind of multithreshold activation function with exactly two thresholds that are parameters of this function. The short notation $\mathrm{BN}(\mathbf{w}, t_1, t_2)$ will be used for a such neural unit.

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ be a dataset containing m training pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ be a n-dimensional real vector ($\mathbf{x}_i \in \mathbf{R}^n$), y_i be a real number, $i = 1, \ldots, m$. S contains data describing a dependency F existing between multidimensional input $\mathbf{x}$ (*feature vector*) and scalar outputs y (*target value*) (actually, S may contain only the part of data, because the rest can be reserved for special purpose, e.g., for the test set). Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ be a set of all feature vectors. Further we also will need $y_{\max} = \max\{y_i : i = 1, \ldots, m\}$, $y_{\min} = \min\{y_i : i = 1, \ldots, m\}$. Bithreshold regressor from [25] deals with continuous targets (or labels) instead of discrete class labels which are typical for classifiers. The main idea behind it is very simple—when predicting the output corresponding to a new instance $\mathbf{x}$, the model returns the mean of all targets of training pairs, which fire the same hidden neurons that input pattern $\mathbf{x}$ does. This leads to a step-wise approximation of sought dependency F that is provided by the neural model learnt on a given dataset S. It is achieved by the discretization of the target dependency by a given number of levels l. This idea is illustrated in Figure 1, where every training pair is two-dimensional data point consisting of one feature "x" and one value "y". Orange circles are used in Figure 1 to depict data points belonging to dataset S, green lines show the step-wise approximation of the dependency between x and y.

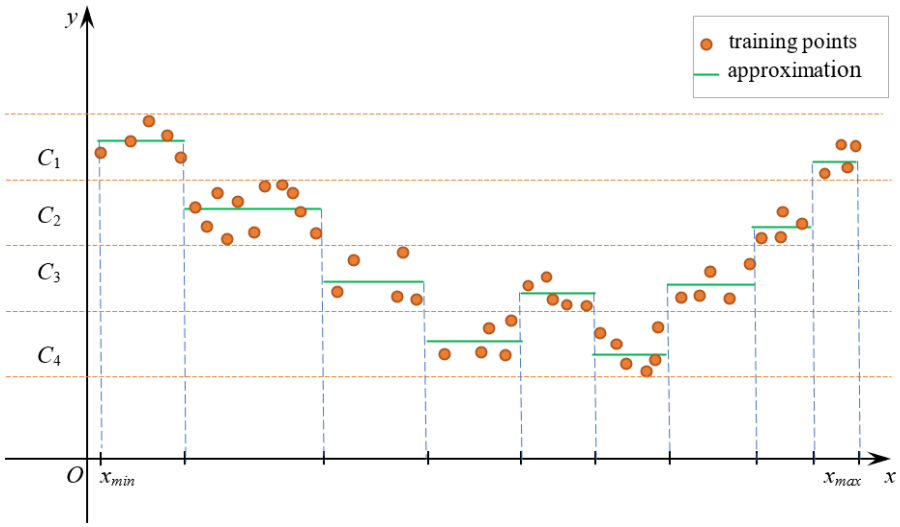

Figure 1: Illustration of the operation of bithreshold neural network regressor.

Four discretization levels are used in Figure 1 in order to approximate given dependency presented by the training set. The range of values (i.e., segment $[y_{\min}, y_{\max}]$) is divided by $l = 4$ parts of equal length that are depicted by horizontal orange dotted lines. This results in the partition of the domain of the function by 9 parts $P_1, \ldots, P_9$ (indicated by blue dashed lines). The above partitions imply that the set X is divided by four subsets C_1, C_2, C_3, C_4 (which can be roughly considered as "classes"). I.e., in Figure 1 "class" C_1 contains x from the union of first and nineth parts, C_2—the second and eighth parts, C_3—the third, fifth and seventh parts, and C_4—fourth and sixth parts. Elements of part $P_1, \ldots, P_9$ can be separated from members of other parts by using partitioning provided by bithreshold activation (1). In one dimension every bithreshold neuron with a single weight $w_1 = 1$ divides the x-axis by three parts, thereby the output of bithreshold neural element is unit when its input lies in the middle of these parts (otherwise, the function (1) returns 0). Thus, by moving along the x-axis from $x_{\min}$ to $x_{\max}$, we walk through successive parts $P_1, \ldots, P_i, \ldots, P_9$, which can be considered as activation ranges of corresponding 9 bithreshold activation functions (1) with two thresholds equal to left and right bounds of the current part P_i, respectively, $i = 1, \ldots, 9$.

The previous example suggests the network architecture. It was called in [25] *l-level hybrid neural network* and used there as a regressor. Actually, it is quite simple, because this is multilayer perceptron with two layers of computation. There are k bithreshold neurons in the hidden layer ($k \geq l$). The output layer contains a single linear neuron (i.e., a neuron with the linear activation function). The diagram of the architecture graph of l-level hybrid NN regressor can be found in [25].

The construction of the hidden layer as well as the rule of node activation is crucial for the family of designed in [8, 25] a hybrid NN with hidden layer formed by multithreshold nodes. Units in this layer are divided by l groups, each of which corresponds to a separate level of discretization, where l is a total number of such levels. The final layer activation rule is rather sophisticated and employ WTA (winner-takes-all) mode to the preliminary preactivation of bithreshold hidden units. The single output unit has linear activation function without a bias.

Suppose that $\mathrm{BN}(\mathbf{w}, t_1, t_2)$ wins the "competition" in the hidden layer. Therefore, all other layer outputs are set to zero and the network output is equal to the weight coefficient corresponding to the connection between $\mathrm{BN}(\mathbf{w}, t_1, t_2)$ and the output node. Assume this weight to be equal to the average value of all targets of training instances, which activate this neuron. Thus, our regressor performs in the way that was stated when main idea of the bithreshold approach in the regressor design was described.

Let us show that such choice is quite reasonable in one dimension. Let us return to the example in Figure 1. Suppose that x belongs to part P_i, $(1 \leq i \leq 9)$. Under our assumption, the output value of the network on input x is equal to mean of values of targets corresponding to all instances in P_i. Therefore, for all $x \in P_i$ the network output is the same and the plot of the regressor output

function is step-wise and coincides with the green curve. Thus, a bithreshold regressor with 9 hidden nodes produce a desired 4-level approximation when two neurons are required for "classes" C_1, C_2, C_4, and three neurons—for "class" C_3.

Let us consider the synthesis algorithm NNRegressor(X, $\mathbf{y}$, l, α) proposed in [25]. Note that the second parameter $\mathbf{y}$ is a target vector of the length m, whose components are target values extracted from the dataset S. NNRegressor includes 21 steps and can be divided into two principal stages. The first stage corresponds to the discretization and results in the partition of training data by l classes C_1, ..., C_l. This preprocessing is necessary for the future synthesis of modified version of regressor and it is extracted in the following function:

Partition(X, $\mathbf{y}$, l)

1. $y_{\min} \leftarrow \min\{y_i : 1 \le i \le m\}$
2. $y_{\max} \leftarrow \max\{y_i : 1 \le i \le m\}$
3. $d \leftarrow 1.001(y_{\max} - y_{\min}) / k$
4. for $j \leftarrow 1$ to m:
5. $\quad i \leftarrow \lfloor (y_j - y_{\min}) / d \rfloor + 1$
6. $\quad$ include $\mathbf{x}_j$ in C_i
7. return (C_1, ..., C_l)

The whole algorithm NNRegressor(X, $\mathbf{y}$, l, α) uses design principle described in the previous subsection and illustrated in Figure 1 in 1D. It is appropriate in the case of n-dimensional input vectors. Notice that the explanation of used notations as well as the detailed justification of algorithm steps and the role of hyperparameter α can be found in [25].

3.1.2. Analysis of the basic model of neural network regressor

Let us analyze the above model of bithreshold regressor. Its performance is intuitively clear in the case $n = 1$, but this clarity quickly disappears in the case of many dimensions.

The advantage of the model is that it produces a step-wise approximation of a studied dependency without the keeping the full dataset in the memory that is typical for *instance-based models* of regressor, e.g., k-nearest neighbor regressor [3]. The *model-based approach* [4] reduces the necessary memory size roughly by n times, where n is the number of features [39].

Consider now difficulties related to the application of the model in the solving of regression tasks. They are summarized in the following list:

1. Model is often indecisive and cannot make a prediction when a new instance is presented to the network (so-called "indecisiveness").

2. Model prediction for a new instance that is close enough to a "cluster" C in the set X can differs considerably from the targets corresponding to instances from cluster C. Conversely, prediction for a new instance that is very distant from cluster C can be obtained as mean of targets corresponding to this cluster, because this instance activates the hidden neuron corresponding to cluster C (these two downsides are combined in so-called "nonlocality" of regressor).

3. The size of the hidden layer is too large in the case of large values of discretization parameter l as well as for large datasets.

4. The size of the hidden layer of the network depends on the order in which the training pairs were stored in the dataset.

It seems that the cause of first two drawbacks is the nature of the bithreshold activation function as well as the activation mode of the hidden layer. Consider the impact of the bithreshold

activation (1). In the case of single input feature ($n = 1$) this activation was able to produce "dense" partition of the range of function (namely, $[x_{\min}, x_{\max}]$) without any "holes". But in the case of numerous features the situation changes dramatically. It is shown in Figure 2 in the case of only two dimensions. Suppose that $l = 3$ and this results in the partition of the set X by "classes" C_1, C_2 and C_3, as it shown in Figure 2. Remember the geometrical interpretation of bithreshold neuron that consists in the partition of the space $\mathbf{R}^n$ by pair of parallel hyperplanes. Their parameters are defined in steps 14–16 of mentioned procedure NNRegressor [25] in order to separate a large number as possible of representatives of the set, which is handled in the current iteration by "classes".

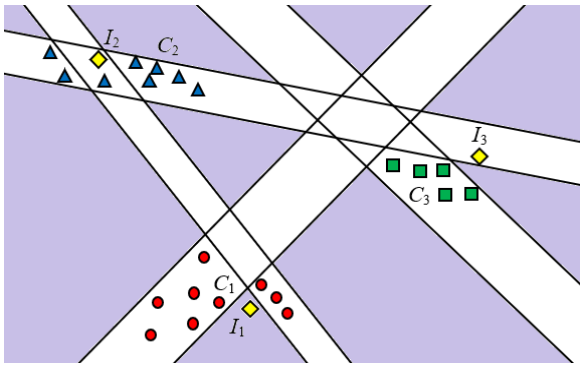

Figure 2: Illustration of the performance of bithreshold regressor in the case $n = 2$.

But as shown in Figure 2, it often results in pair of hyperplanes with very narrow gap. Therefore, after the partitioning of points by regions separated by hyperplanes, numerous empty places arise. E.g., in Figure 2 the most of the plane (highlighted by lavender color) is not covered by the area between any pair of parallel lines, which define the decision surface of the regressor. It implies the possibility of the great indecisiveness of bithreshold regressor. For example, new instance I_1 is rejected by regressor despite it is situated very close to representatives of set C_1, which are separated by two bithresholds neurons.

Instances I_2 and I_3 demonstrate the nonlocality of basic regressor model. I_2 is "caught" by two bithreshold neurons, first of which appeared during the separation of points belonging to set C_1, whereas the second neuron corresponds to set C_2. Thus, two very distant sets would be used to determine the output of the regressor on the input I_2 despite that it is intuitively obvious from Figure 2 that only targets corresponding to C_2 would be considered for the prediction making. The same is true for instance I_3, which is attributed by regressor as representative of C_2, but seems to be considerably closer to C_3.

Consider last two drawbacks. The accuracy of the regressor is influenced by the number of levels of discretization l. This introduces trade-off between the model precision and complexity, which is typical for the computational intelligence. Setting l equal to the size of the dataset causes the model to memorize every training example by heart, resulting in perfect performance on the training set. However, this produces a large instance-based model, potentially exceeding the size of the dataset itself. Such a model is prone to severe overfitting and is likely to generalize poorly to unseen data. Conversely, small l leads to more compact models that yield only coarse approximation, which may result in the highly inaccurate predictions. For example, the dependency between the feature and the corresponding value in the second interval in Figure 1 is not adequately represented by a single horizontal "green" segment of the step-wise approximation curve.

The last issue appears to stem from the way NNRegressor is designed. It could be overcome by the randomization of the choice of instances in step 6 as well as by repeated synthesis of the networks and the selection of the network with the best performance.

The simulation results confirmed the above analysis of the performance of bithreshold regressor. However, it should be emphasized that the performance decay of the bithreshold regressor is not caused by the errors in the synthesis algorithm NNRegressor(X, $\mathbf{y}$, l, α), but by the

construction of hidden layer related to the activation mode of neurons, which form this layer, and general network design principles.

3.2. Neural network regressor with a centered activation

Let us show that the nonlocality and the indecisiveness of the bithreshold regressor can be reduced by the proper modification of the activation function.

This goal can be achieved in various ways. One of them consists in the use during the activation of a particular hidden neuron parameters that reflect some essential characteristics of the training instances, which were used for the synthesis of this neuron. One of the simplest parameters is the centroid of these training instances, as proposed in [39]. This idea can be implemented using the following model of centered bithreshold neuron $\mathrm{CBN}(\mathbf{w}, t, D, \mathbf{c})$ whose output y on the input $\mathbf{x}$ is defined as follows:

$$y(\mathbf{x}) = \begin{cases} 1, & \text{if } |\mathbf{w}\cdot\mathbf{x} - t| < \dfrac{D}{\left\| p_{\mathbf{w},t}(\mathbf{x}) - \mathbf{c} \right\| + 1}, \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

where $t \in \mathbf{R}$ is a "threshold" of CBN, $p_{\mathbf{w},t}(\mathbf{x}) = \mathbf{x} - \|\mathbf{w}\|^{-2}(\mathbf{w}\cdot\mathbf{x} - t)$ is the projection of the point $\mathbf{x}$ on the hyperplane $\pi = \{\mathbf{x} \in \mathbf{R}^n : \mathbf{w}\cdot\mathbf{x} = t\}$, $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbf{R}^n$ is a *center of neuron* that might belong to the same hyperplane π and D is a positive number representing the maximum acceptable offset to the hyperplane π. Equation (2) uses $\|\cdot\|$ as notation for the Euclidean norm (i.e., $\left\| p_{\mathbf{w},t}(\mathbf{x}) - \mathbf{c} \right\|$ is the distance between the projection and the center). Notice that unlike (1), the activation function (2) depends essentially not only on two thresholds but on two vector parameters $\mathbf{w}$ and $\mathbf{c}$ as well as two scalar values t and D. Therefore, each CBN has its own activation. Let us make a remark regarding the name of this model of neural unit. The term "bithreshold" can be not so obvious, because the only one explicit threshold (namely, t) is used. But the performance of CBN is similar to the performance of BN as its decision region is halved by pivot hyperplane π and the proximity of the point to decision surface is measured using parameter D instead of pair of parallel hyperplanes, the distance between which is depending on thresholds t_1 and t_2 of BN.

Consider the difference in the performance of $\mathrm{BN}(\mathbf{w}, t_1, t_2)$ and $\mathrm{CBN}(\mathbf{w}, t, D, \mathbf{c})$ Let C_1 and C_2 be two sets of instances obtained after discretization by l levels in two-dimensional space. In Figure 3 the representatives of the set C_1 and C_2 are marked by circles and diamonds, respectively. It is evident that the separation of sets C_1 and C_2 cannot be achieved by a single BN (note that in two dimensions decision region is defined by a pair of parallel lines). I.e., for the two parallel lines drawn in Figure 3 (a) two circles remain outside the activation range. It is caused by the relatively narrow distance between lines (namely, $2d$) due to the presence of diamonds (representatives of the second set C_2) close to the dashed pivot line.

It is easy to see that CBN separating sets C_1 and C_2 by properly handling instance I_1. The choice of the center $\mathbf{c}$ of CBN and the activation rule (2) ensure that maximum distance $2D$ between two curves bounding activation range of this neuron is significantly greater than $2d$ (of basic BN), as it is shown in Figure 3 (b). It is obtained as a result of the gradual reduction of the width of the decision region. Thus, the application of CBN instead of BN leads, in general, to the wider decision region near the center of a neuron, reduces the size of "holes" and can decrease the number of neurons in the hidden layer. Thus, a network with CBN-based hidden layer can reduce the first downside (indecisiveness) of bithreshold regressor as well as decrease the size of hidden layer.

Moreover, the integration of centered neurons in the network may considerably boost the regressor performance by reducing the second drawback of its basic version. Le I_2 be a new sample presented to the network. If we assume that only close instances have close targets it seems plausible that the set C_1 is not a good candidate to be involved in the prediction of the target value of I_2.

Nevertheless, the basic BN in Figure 3 (a) is activated by I_2, because this instance is between (unbound) hyperplanes that define the weights and threshold of this neuron. Note, that the CBN from Figure 3 (b) operates in the way that is more local due to gradual shrinkage. Since I_2 is distant from the center of CBN, the second regressor does not use only the class C_2 in order to make prediction.

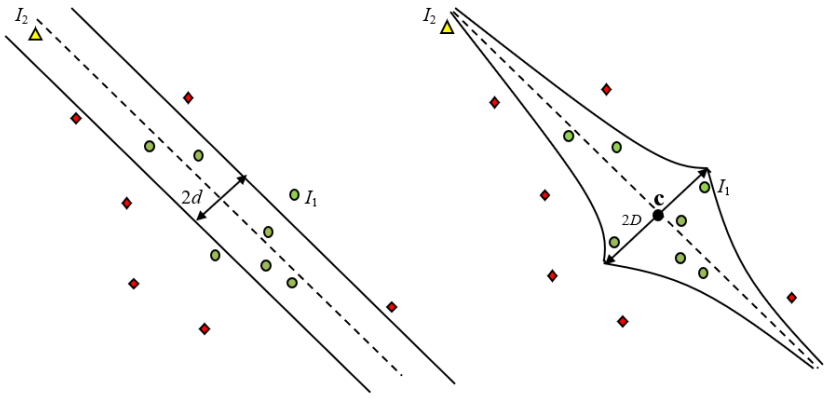

Figure 3: Comparison of the performance of BN (*a*) and CBTN (*b*).

Note that the application of CBN does not eliminate the first drawback. Actually, it only slightly reduces the indecisiveness of regressor outside the training set. It is necessary an extra tool in order to force the regressor make prediction for every input vector. This can be provided by change of the activation mode. Let us use the softmax activation mode instead of binary output in WTA-mode. Thus, we can employ a smoothed continuous version of a winner-take-all nonlinearity. Let $z_k(\mathbf{x})$ be an output of kth hidden CBN. Then

$$z_k\left(\mathbf{x}\right)=\frac{e^{y_k(\mathbf{x})}}{\sum\limits_{i=1}^{K}e^{y_i(\mathbf{x})}}, \tag{3}$$

where K is the size of the hidden layer and

$$y_k\left(\mathbf{x}\right)=\frac{D}{\left\|p_{\mathbf{w},t}\left(\mathbf{x}\right)-\mathbf{c}\right\|+1}-\left|\mathbf{w}\cdot\mathbf{x}-t\right|,\ k=1,\ldots,K. \tag{4}$$

Notice that now the operation mode (3)–(4) of the hidden layer does not employ any discreteness. This (along with the output linear node) ensures that the gradient-based learning techniques are acceptable for such modification of regressor. Nevertheless, the procedure below is the modification of the basic NNRegressor from [25] in the case of the use of centered bithreshold neurons:

CenteredNNRegressor (X, $\mathbf{y}$, l)
1 $C \leftarrow$ Partition(X, $\mathbf{y}$, l)
2 $k \leftarrow 0$
3 for $i \leftarrow 1$ to l:
4 $A_i \leftarrow C_i$:
5 while $A_i \neq \varnothing$:
6 $k \leftarrow k+1$
7 $r \leftarrow \min\left\{n, \left|A_i\right|\right\}$
8 Move r instances from A_i into matrix A
9 Solve the linear system $\mathbf{w}\cdot A^T = \mathbf{1}$

10 $\mathbf{c} \leftarrow \mathbf{0}$, $v_k \leftarrow 0$

11 for $\mathbf{x}$ in A

12 $\mathbf{c} \leftarrow \mathbf{c} + \mathbf{x}$

13 $v_k \leftarrow v_k + y(\mathbf{x})$

14 $\mathbf{c} \leftarrow \mathbf{c} / r$

15 $D \leftarrow +\infty$

16 for $\mathbf{x}$ in $C \setminus C_i$:

17 $\mathbf{p} \leftarrow \mathbf{x} - \|\mathbf{w}\|^{-2} (\mathbf{w} \cdot \mathbf{x} - 1)$

18 $D \leftarrow \min\left\{D, \frac{|\mathbf{w} \cdot \mathbf{x} - t| + \varepsilon}{\|\mathbf{p} - \mathbf{c}\| + 1}\right\}$

19 for $\mathbf{x}$ in A_i:

20 $p_w \leftarrow \mathbf{x} - \|\mathbf{w}\|^{-2} (\mathbf{w} \cdot \mathbf{x} - 1)$

21 if $|\mathbf{w} \cdot \mathbf{x} - 1| < \frac{0.99D}{\|\mathbf{p} - \mathbf{c}\| + 1}$:

22 $r \leftarrow r + 1$

23 $v_k \leftarrow v_k + y(\mathbf{x})$

24 remove $\mathbf{x}$ from A_i

25 $v_k \leftarrow v_k / r$

26 Add CBN$(\mathbf{w}, 1, \mathbf{c}, 0.99D)$ in the hidden layer

27 Add the output linear node with weight vector $(v_1, ..., v_k)$

Note that in step 13 and 23 $y(\mathbf{x})$ denotes the target that corresponds to the instance $\mathbf{x}$ and ε is a small constant used to avoid undesired random landing on the plane. E.g., it is possible to use $\varepsilon = 10^{-8}$.

4. Experiment

The performance of centered bithreshold regressor proposed in the previous section was compared with performance of the basic bithreshold regressor as well as some popular regressors in order to estimate its ability to solve the regression task. Two datasets were used. The first was the synthetic dataset with 1000 30-dimensional instances generated by make_regression() method provided by Scikit-learn machine learning platform [23]. Ten informative features were used along with small gaussian noise. The second was real-world "California Housing" dataset [4] downloaded from OpenML machine learning repository [40]. This is a preprocessed version of the original dataset [4] without missing values and categorical feature "ocean proximity". Dataset contains 20600 instances with 8 numeric (floating point) input features and a single dependent numeric target feature. The names and description of model features are available in [3, 23, 40].

During the simulation performances of following 5 classical and 2 bithreshold-like regression models were measured. Classical models were: ElasticNet (linear regression with combined L_1 and L_2 regularization), 7-nearest neighbors, random forest (averaging algorithms based on randomized decision trees) [3], LinearSVR (support vector regression with linear kernel) [23] and MLPRegressor (multilayer perceptron regressor) [3]. Data preprocessing techniques, including normalization, scaling, and standardization, were not applied.

The Scikit Learn library implementation of each classical regressor were used with recommended parameter set [23]. The bithreshold NN regressor and its centered modification were studied more carefully. The basic model has two hyperparameters (l and α), the modified regressor employs only first of them. Thus, only the impact of the number of discretization levels to the performance was considered. It is not very strict restriction, because as shown in [25] the "optimal" values of

hyperparameter $\alpha \approx 1$. The grid search was employed in order to find the "optimal" value of parameter l, where all l from the set $L = \{5, 6, \ldots, 50\}$ were tried.

Similar to [25] usual for regression analysis metrics were used: mean squared error (MSE), mean absolute error (MAE) and R^2 [23]. 5-fold cross-validation [3, 23] was employed in order to obtains consistent results. The results of experiments are presented and discussed in the following section.

5. Results and discussion

Experiment results are presented in Table 1 and Table 2, respectively, where mean values by all 5 cross-validation splits are shown. In both tables the best model of regressor (winner by all metrics) is highlighted.

Table 1
Experiment results on synthetic dataset

Regressor	**Mean metric value**		
	MSE	MAE	R^2
ElasticNet	4556.61	54.87	0.88
7-nearest neighbors	17495.36	106.84	0.55
LinearSVR	0.02	0.09	1
RandomForest	30504.02	140.91	0.21
MLPRegressor	2702.58	37.77	0.93
Basic Bithreshold NN	42003.84	152.45	0.17
Centered Bithreshold NN	3115.57	40.13	0.91

Table 2
Experiment results on "California Housing" dataset

Regressor	**Mean metric value**		
	MSE	MAE	R^2
ElasticNet	1.03	0.8	0.27
7-nearest neighbors	0.38	0.41	0.71
LinearSVR	13.25	0.57	−9.16
RandomForest	0.65	0.61	0.5
MLPRegressor	0.36	0.38	0.72
Basic Bithreshold NN	0.39	0.47	0.68
Centered Bithreshold NN	0.36	0.37	0.73

The last two rows of both tables contain best results of performance of both considered bithreshold regressors corresponding to the "best" number of discretization level l (found by using the grid search in the set L). For the first dataset best values of hyperparameter l were 17 and 11, respectively. For the second dataset—22 and 15, respectively. By analyzing simulation results, it is possible to conclude:

1. Regressor performance strongly depends on the concrete task (i.e., linear support vector machine was excellent of synthetic dataset and failed completely on the real-world one).
2. Basic Bithreshold NN regressor has the worst performance on the first dataset and the third worst on the second dataset.
3. The growth of the dimensionality of the feature space results in the considerable loss of

efficiency of the performance of the basic modification of bithreshold regressor, whereas this model was not so sensitive to the change of the dataset size.

4. Centered Bithreshold NN regressor performed relatively well on both datasets (third and first positions, respectively).
5. The number of discretization levels l has direct impact on the performance of both modifications of bithreshold regressor. It must be large enough in order to ensure the good quality of prediction produced by such regressors.
6. The centered modification of bithreshold NN required significantly lower value of discretization level hyperparameter for its best performance compared to basic version of regressor.
7. The synthesis of centered modification of bithreshold regressor yielded a NNs whose size of hidden layer is less by 8–23% compared to basic model of bithreshold regressor with the same value of discretization levels.
8. Regressor whose hidden layer consists of centered bithreshold neurons had better generalization ability that its prototype, because it showed lower difference between prediction accuracy measured on training and validation set, respectively.

It should be also mentioned that a centered bithreshold neuron with n inputs requires additional memory for storing its center vector $\mathbf{c}$. Therefore, the memory requirement for modified version of regressor is roughly twice as large as for basic binary-valued model with the same size of the hidden layer. But in practice centered bithreshold regressor having the performance compared with basic model can be synthesized using much lower value of discretization levels as well as a smaller number of hidden nodes due to shape of the decision regions. Thus, the application of a centered bithreshold network instead of a basic NN generally results in a more compact and, notably, more robust and precise model.

6. Conclusions

The research is devoted to the study and application of multithreshold model and methods in neural computation. First, the basic bithreshold approach of the design of hybrid 2-layer NN was analyzed. The four main weakness of the performance of bithreshold NN regressor were found and explained. Two of them are caused by the nature bithreshold binary-valued activation as well as the operation mode of the hidden layer of the network along with general approaches to the design of regressor. The model of centered bithreshold neuron was proposed in order to improve the performance. It uses new activation rule that is a distant analogue of activations used in bithreshold-based classifiers in [8] and [39]. The advantage of the model of CBN consists in the extension of the information that is memorized by the neuron. This model keeps in memory not only parameters of hyperplane containing some n instances from the training set with close targets, but also the centroid of these instances and the parameter D describing the width of a compact region defined by all point with similar targets. The hidden layer produces continuous output and is embedded in the previous model of regressor by replacing WTA activation mode to more flexible softmax method.

The above-mentioned approach results in the model of the multilevel 2-layer centered neural network with a single output node with linear activation. The proposed model can be applied to regression tasks. The iterative algorithm has been proposed for the synthesis of such networks. The hidden layer size is not the parameter of algorithm, but directly depends on the desired number of the discretization levels. The distribution of training pairs in the training set also plays important role during the synthesis.

The fifth section of the paper contains experimental results obtained on synthetic as well real-world benchmark datasets [3]. The analysis of these results confirms that synthesized NN regressor is concurrent enough compared to different classical regression models and can even overperform it. Moreover, the application of new network architecture and new design algorithm produces a smaller NN with better performance and generalization ability compared to basic bithreshold NN regressor.

Of course, the designed model of regressor is not flawless. It reduces only first two drawbacks of the bithreshold regressor from [25]. Author hopes that it is possible to improve the performance of the regressor using deeper network with additional hidden layers. The introducing of such layers in the network architecture as well as modification of the synthesis algorithm can extend the range of possible applications of multithreshold-based regressors. Another promising strategy consists in the replacing of the synthesis approach by the training one based on the backpropagation.

Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] J. Kaur, J. Verma, Past, present and future of computational intelligence: a bibliometric analysis, in: AIP Conference Proceedings 2916(1), 2023, 020001.

[2] V. Teslyuk, A. Kazarian, N. Kryvinska, I. Tsmots, Optimal artificial neural network type selection method for usage in smart house systems, Sensors 21.1 (2021): 47.

[3] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 3rd ed., O'Reilly Media, Sebastopol, 2022.

[4] E.H. Houssein et al., Soft computing techniques for biomedical data analysis: open issues and challenges, Artificial Intelligence Review 56 (2023): 2599–2649.

[5] K. Yemets, I. Izonin, I. Dronyuk, Time series forecasting model based on the adapted Transformer neural network and FFT-based features extraction, Sensors 25.3 (2025): 652.

[6] M. Lupei, A. Mitsa, V. Repariuk, V. Sharkan, Identification of authorship of Ukrainian-language texts of journalistic style using neural networks, Eastern-European Journal of Enterprise Technologies 1.2 (103) (2020): 30–36.

[7] F. Geche et al., Synthesis of time series forecasting scheme based on forecasting models system, in: CEUR Workshop Proceedings, volume 1356, 2015, pp. 121–136.

[8] O. Vyshnevskyy, I. Zhuravchak, V. Yakovyna, Improving energy efficiency in smart building using deep reinforcement learning control strategy, in CEUR Workshop Proceedings Open source preview, volume 4013, 2025, pp. 1–14.

[9] I. Izonin et al., Regression-based model for predicting simulated vs actual building performance discrepancies, in: Procedia Computer Science, volume 251, 2024, pp. 633–638.

[10] S. Moon, Y. Kim, Mounting angle prediction for automotive radar using complex-valued convolutional neural network, Sensors 25.2 (2025): 353.

[11] V. Kotsovsky, F. Geche, A. Batyuk, Artificial complex neurons with half-plane-like and angle-like activation function, in: Proceedings of 10th International Conference on Computer Sciences and Information Technologies, CSIT 2015, Lviv, Ukraine, 2015, pp. 57–59.

[12] F. Geche et al., Synthesis of the integer neural elements, in: Proceedings of the International Conference on Computer Sciences and Information Technologies, CSIT 2015, Lviv, Ukraine, 2015, pp. 121–136.

[13] Mitsa et al., On computer modeling of quadratic surface intersections, in: Proceedings of IEEE 5th International Conference on Smart Information Systems and Technologies, SIST 2025, Astana, Kazakhstan, 2025, pp. 1-6.

[14] P. Ramachandran, B. Zoph, Q. V. Le, Swish: a self-gated activation function, arXiv: Neural and Evolutionary Computing, 2017.

[15] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: a comprehensive survey and benchmark, Neurocomputing 503 (2022): 92–108.

[16] A. Apicella, F. Donnarumma, F. Isgrò, R. Prevete, A survey on modern trainable activation functions, Neural Networks 138 (2021): 14–32.

[17] I. Jahan, M.F. Ahmed, M.O. Ali, Y.M. Jang Self-gated rectified linear unit for performance improvement of deep neural networks, ICT Express 9.3 (2023): 320–325.

[18] D. Misra, Mish: a self regularized non-monotonic neural activation function, arXiv: Machine Learning, 2019. URL: https://arxiv.org/vc/arxiv/papers/1908/1908.08681v1.pdf.
[19] V. Kotsovsky, Multithreshold neurons with smoothed activation functions, in: CEUR Workshop Proceedings, volume 3983, 2025, pp. 93–102.
[20] N. Jiang, Y. X. Yang, X. M. Ma, and Z. Z. Zhang, Using three layer neural network to compute multi-valued functions, in 2007 Fourth International Symposium on Neural Networks, June 3-7, 2007, Nanjing, P.R. China, Part III, LNCS 4493, 2007, pp. 1-8.
[21] Z. Obradovic, I. Parberry, Learning with discrete multivalued neurons, Journal of Computer and System Sciences 49 (1994): 375–390.
[22] V. Kotsovsky, Learning of multi-valued multithreshold neural units, in: CEUR Workshop Proceedings, volume 3688, 2024, pp. 39–49.
[23] Scikit-learn: Machine learning in Python, 2025. URL: https://scikit-learn.org/stable/.
[24] T. Szandała, Review and comparison of commonly used activation functions for deep neural networks, Studies in Computational Intelligence, volume 903, pp. 203-224, 2021.
[25] V. Kotsovsky, A. Batyuk, Towards the design of bithreshold ANN regressor, in: 19th IEEE International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT 2024. Lviv, October 16–19, pp. 1–4, 2024.
[26] D. R. Haring, Multi-threshold threshold elements, IEEE Transactions on Electronic Computers EC-15.1 (1966): 45–65.
[27] Ghosh S., Choudhury A., Partition of Boolean functions for realization with multithreshold threshold logic elements, IEEE Transactions on Computers 22.2 (1973): 204–215.
[28] S. Olafsson, Y. S. Abu-Mostafa, The capacity of multilevel threshold function, IEEE Transactions on Pattern Analysis and Machine Intelligence 10.2 (1988): 277–281.
[29] M. Anthony, Learning multivalued multithreshold functions, CDMA Research Report No. LSE-CDMA-2003-03, London School of Economics, 2003.
[30] I. Prokíc, Characterization of multiple-valued threshold functions in the Vilenkin-Chrestenson basis, J.of Multiple-Valued Logic and Soft Computing 34 (2020): 223–238.
[31] M. Lupei et al., Analyzing Ukrainian media texts by means of support vector machines: aspects of language and copyright, in: Z. Hu., I. Dychka, M. He (Eds.), Advances in Computer Science for Engineering and Education VI. ICCSEEA 2023, Lecture Notes on Data Engineering and Communications Technologies, volume 181, Springer, Cham, 2023, pp. 173–182.
[32] V. Kotsovsky, Synthesis of multithreshold neural network classifier, in: CEUR Workshop Proceedings, volume 3711, 2024, pp. 75–88.
[33] T. Gowda et al., Identification of threshold functions and synthesis of threshold networks, IEEE Transaction on Computer-Aided Design, 30.5 (2011): 665-677.
[34] M. Nikodem, Synthesis of multithreshold threshold gates based on negative differential resistance devices, IET Circuits Devices Syst. 7.5 (2013): 232–242.
[35] O. Vyshnevskyy, L. Zhuravchak, Forecasting the electricity consumption for energy management software using an ensemble model, in: 19th IEEE International Conference on Computer Science and Information Technologies, CSIT 2024, Lviv, Ukraine, pp. 1-5, 2024.
[36] J. Li et al., Multithreshold change plane model: Estimation theory and applications in subgroup identification. Statistics in Medicine 40.15 (2021): 3440–3459.
[37] J. Wang et al., A model-based multithreshold method for subgroup identification. Statistics in Medicine 38.14 (2019): 2605–2631.
[38] A. Reinke et al., Common Limitations of Image Processing Metrics: A Picture Story (2023). URL: https://arxiv.org/abs/2104.05642v8.
[39] V. Kotsovsky, A. Batyuk, Representational capabilities and learning of bithreshold neural networks, in: S. Babichev et al. (Eds), Advances in Intelligent Systems and Computing, volume 1246, Springer, Cham, 2021, pp. 499–514.
[40] OpenML: A worldwide machine learning lab, 2025. URL: https://www.openml.org.