Synthesis of a 3-Layer Neural Network Classifier with the Bithreshold First Hidden Layer

Vladyslav Kotsovsky[1,*]

[1] State University "Uzhhorod National University", Narodna Square 3, 88000, Uzhhorod, Ukraine

Abstract

The paper treats the application of the bithreshold approach in the design of neural network classifiers. A novel hybrid 3-layer neural network model is proposed whose first hidden layer consists of bithreshold neurons, and the second hidden layer employs the softmax activation function. This model is intended to solve multiclass classification tasks. A supervised synthesis algorithm is designed for this neural network architecture. It consists of two stages. During the first stage a given training pattern is separated from representatives of other classes using single-threshold neurons, which are gradually converted into bithreshold neural units. In the second stage, the network design procedure reduces the size of network hidden layers in order to simplify the network and enhance the classifier's recognition ability. The performance of the proposed model is compared with that of several popular machine learning classifiers on a real-world dataset. Simulation results on "optical recognition of handwritten digits" benchmark demonstrate that the developed neural network model is suitable for multiclass classification tasks.

Keywords

bithreshold neuron, multithreshold neuron, neural network, classification, machine learning

1. Introduction

Neural networks play a leading role in modern machine learning due to their ability to model complex, non-linear relationships [1, 2]. They have become the foundation of many state-of-the-art systems in image recognition [3], natural language processing [4], game playing [5] and forecasting [6]. Their flexible architectures [1, 7] and capacity to learn from large datasets [8] have made them essential tools in advancing artificial intelligence [9, 10].

Activation functions are crucial for introducing non-linearity into neural networks [1, 11], allowing them to solve complex tasks [12]. They determine how signals are transformed and propagated through the network [1]. The choice of activation function can significantly impact the performance and convergence of a model [13, 14]. Without activation functions, neural networks would behave like simple linear models [15].

Multithreshold approach in neural computation arose as one of the first attempt to enhance the ability of classical activation functions (such as the Heaviside or sign functions [1] by using two or more thresholds [16, 17]. Modern applications concern smoothed continuous modification of multithreshold activations [18], which can outperform modern activations such as ReLU [3], Swish [14, 19] etc.

Multithreshold approach was developed for the binary classification of multidimensional patterns [17, 20, 21]. Binary valued bithreshold neurons [22] as well as their multithreshold generalizations [21] are capable to increase the recognition ability of a neural network [21] by the proper use of additional thresholds [23]. Moreover, the application of multithreshold neural networks in pattern classification may significantly reduce the network complexity [3, 12].

ICST-2025: Information Control Systems & Technologies, September 24-26, 2025, Odesa, Ukraine.
* Corresponding author.
vladyslav.kotsovsky@uzhnu.edu.ua (V. Kotsovsky)
0000-0002-7045-7933 (V. Kotsovsky)

It should be noted that all known approaches to the learning or synthesis multithreshold multilayer neural models have relied on offline or batch learning modes. The *objective of the present research* is the design of the model of bithreshold neural network (NN) suitable for the online learning.

2. Related works

First studies on binary-valued multithreshold neural units were conducted by D. R. Haring [16] and lead to the development of the so-called multithreshold logic (see [24–26] for further references).

The initial motivation for multithreshold units stemmed from the belief that multiple thresholds could considerably enhance neural models' computational power [16], a hypothesis later confirmed by counting arguments in [20] and [17]. However, early research lacked practical training methods, and only few heuristics were proposed. Learning binary multithreshold models proved difficult due to NP-hardness, even for bithreshold systems [23, 26].

Interest in multithreshold approaches re-emerged two decades later, driven by the development of multi-valued neurons and formal definitions of multithreshold functions [17, 21, 25]. Key contributions by Z. Obradović, I. Parberry, A. Ngom, and M. Anthony gave theoretical justification of online learning algorithms based on incremental correction [21], followed by improved both online and offline methods using relaxation techniques [23]. Note that these algorithms were designed for the learning of a single multithreshold neural unit.

Recent works [11, 27] explored network architectures with multithreshold hidden layers, showing strong performance in classification tasks thanks to offline synthesis algorithms [22, 27]. Hybrid models combining multithreshold, bithreshold, WTA, and single-threshold units further boosted accuracy [11]. Using two thresholds offers a balance between expressive power and synthesis simplicity [22], though more recent studies expanded to models with multiple thresholds and multi-valued outputs [23]. Generalized versions of these models were applied to pattern classification [12].

The multithreshold paradigm also includes hardware implementations, such as those by T. Gowda [33] and M. Nikodem [28, 29]. Applications in regression are relatively rare, as discrete-valued activation are better suited to classification tasks. However, both binary- and continuous-valued multithreshold-based regressors have recently been proposed. The model of neural network regressor that uses binary-valued bithreshold units was designed in [30], whereas its continuous-valued generalization within the gradient-based learning framework was proposed in [18].

3. Models and methods

3.1. Explanation of the bithreshold approach

3.1.1. Model of a bipolar-valued bithreshold neural unit

The key feature of the proposed classifier model is its hidden layer consisting of binary-valued bithreshold neurons. Let us consider a model of such a neuron. It is a binary-valued computational unit [22] with a *weight vector* $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbf{R}^n$ and two thresholds t_1, t_2 $(t_1 < t_2)$, whose single bipolar output *y* is obtained by applying the following activation function

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

to the weighted sum $\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + \ldots + w_n x_n$. Note that in the case $t_2 = +\infty$ function (1) behaves like a sign function (applied to the $w \cdot x - t_1$). Therefore, the single-threshold linear neural unit is a particular case of the bithreshold neural unit. Equation (1) gives the simplest kind of multithreshold activation function, which is a particular case of the general model of multithreshold activation function considered in [26]. The short notation $\mathrm{BN}(\mathbf{w}, t_1, t_2)$ will be used for such a neural unit. It should be mentioned that the use of bipolar range of function {–1, +1} instead of more usual binary

outputs is intentional, because it allows us to avoid the need for an additional normalization level in the networks used in [11].

Consider geometrical concepts related to the performance of bithreshold neural unit. The pair of two parallel hyperplanes $w_1x_1+\ldots+w_nx_n=t_1$ and $w_1x_1+\ldots+w_nx_n=t_2$ divide n-dimensional space R^n by three parts. Thus, the activation function (1) allows us to distinguish all points located in the middle region (i.e., between hyperplanes) from all other points of the space. This induces so-called bithreshold separable sets in the space R^n. Therefore, a bithreshold neural unit can act as a simple binary classifier. It is evident that the classification capacity of a single bithreshold neuron is very limited. It is unable to solve relatively simple classification task related to dichotomies of small finite set of n-dimensional patterns. Furthermore, the general multiclass classification tasks are more frequent and usual than binary ones. This implies that binary-valued bithreshold neuron must be combined within a neural network in order to solve real-world problems.

Let $S=\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)\}$ denote a training sample containing m training pairs (x_i, y_i), where x_i is an n-dimensional real vector ($x_i \in R^n$)—feature vector, y_i is a non-negative integer label belonging to the set $\{1, \ldots, K\}$, where $i = 1, \ldots, m$, and K is the number of classes. In practice, S contains the part of an available dataset, because its remainder can be reserved for special purpose, e.g., for the test set.

3.1.2. Illustration of the idea behind the performance of a bithreshold classifier

Let us consider an example how bithreshold neurons can be useful for multiclass classification. Consider a simple example of ternary classification in two dimensions ($K = 3$, $n = 2$). Figure 1 shows three pairwise linearly non-separable classes of 2-dimensional patterns C_1, C_2 and C_3 as well as 12 representative patterns, which form training sample along with their labels $1, 2, \ldots, 12$ and are consecutively fed on the learner input (in Figure 1, for brevity, patterns are referred only by their labels $1, \ldots, 12$, instead of full notation $x_1, \ldots, x_{12}$).

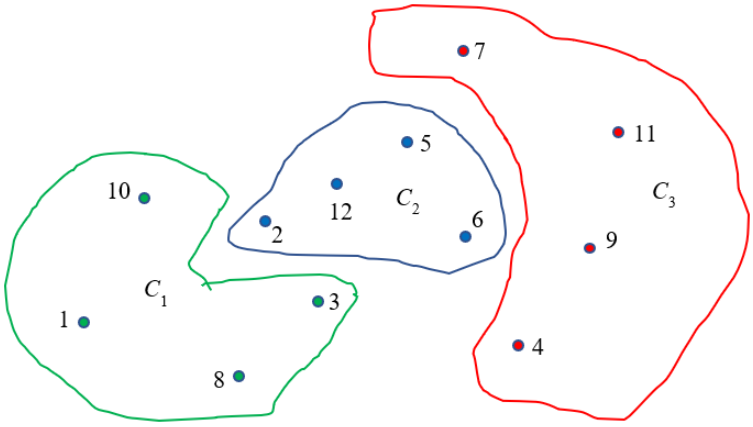

Figure 1: Example of ternary classification

Let us consider how these patterns can be separated using bithreshold neurons. Note that in two dimensions the notion of a hyperplane coincides with a line. It is evident that patterns 1 and 2 are member of different classes and are linearly separable. Third pattern belongs to class C_1, as does the first one. We can properly separate these three patterns using a single BN_1 as shown in Figure 2.

The bithreshold unit BN_1 is defined by a pair of parallel lines l_{11} and l_{12}. It is evident that pattern 4 cannot be correctly classified using BN_1, because it lies in the same region as pattern 3 (actually, the pattern 1 lies in the same region with respect to the output of BN_1). Thus, an additional line is necessary to separate it from patterns 1 and 3. A possible solution is illustrated in Figure 3. Note that a single (dotted) line l_2 is used, corresponding to a single-threshold neuron TN_2.

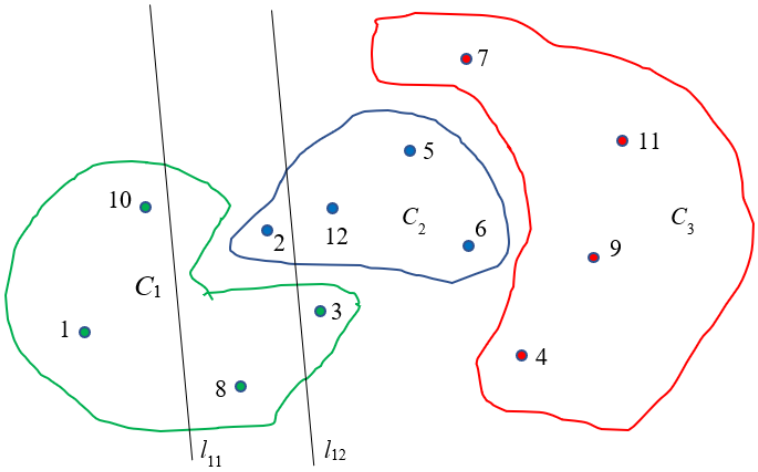

Figure 2: Proper separation of first three points using a single BN_1

Let us assume that a new point always falls within the positive half-plane defining by a single separating line. Assume that single lines, like l_2, can be later complemented with another parallel line in order to form a full bithreshold neuron instead of single-threshold one. This strategy provides benefits in terms of memory capacity in the general n-dimensional case for large n, as it requires a single additional scalar parameter for another threshold, rather than of $n + 1$ new parameters in the case when a new complete hyperplane is employed. Note that this approach enabled us to obtain the second line l_{12} with intention to extend a single-threshold neuron corresponding to the line l_{11} of the bithreshold neuron BN_1.

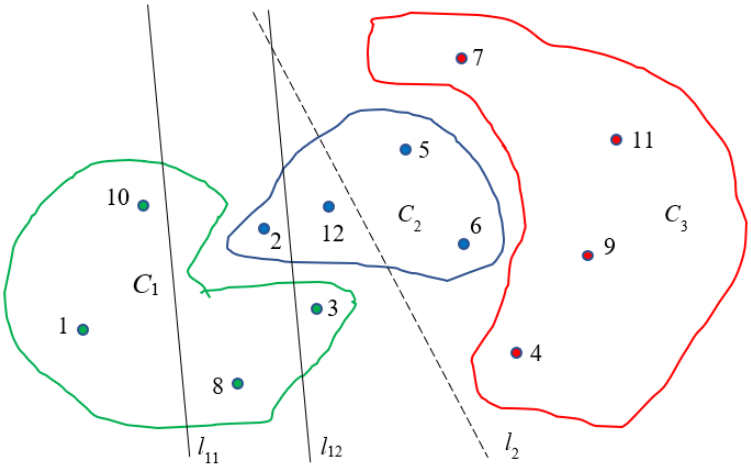

Figure 3: Separation of first four points using BN_1 and line l_2

Consider pattern 5. It belongs to the same plane region as pattern 4. Thus, it must be separated by an additional line. This cannot be done by using a line parallel to l_2, i.e., by extending TN_2 to some bithreshold unit. Let this line be l_3, as shown in Figure 4. This line corresponds to some single-threshold unit TN_3. Pattern 6 falls in the same region as pattern 5. Thus, no additional separation is required. Pattern 7 belongs to class C_3 but lies in the same region as the previous two patterns. Therefore, we need to separate pattern 7 from them. A possible solution involves a new line, l_4, as shown in Figure 5.

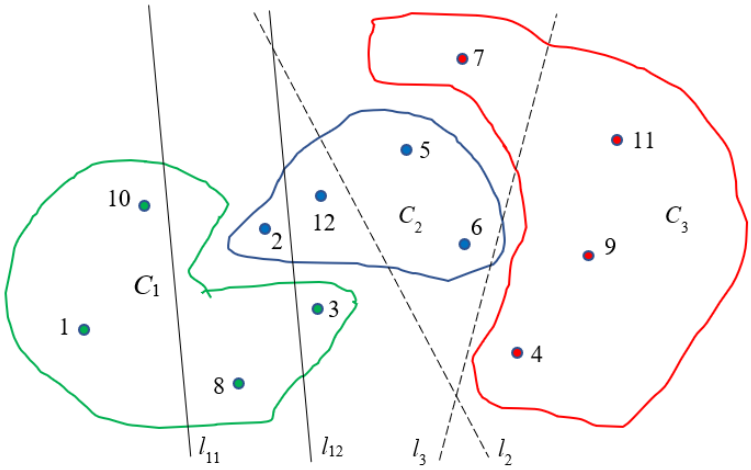

Figure 4: Separation of first six points using BN_1 and two lines l_2 and l_3

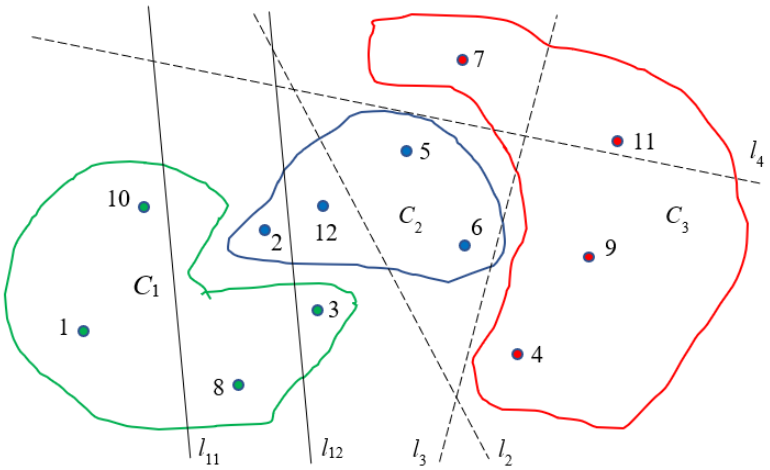

Figure 5: Separation of first seven points using BN_1 and lines l_2, l_3 and l_4

Pattern 8 conflicts with pattern 2. They can be separated by drawing the line l_{42}, which is parallel to l_4, as shown in Figure 6. This does not affect the separability of patterns 5 and 6 from patterns 7. Thus, the pair of parallel lines l_{41} (l_4 in Figure 5) and l_{42} defines the bithreshold neuron BN_4.

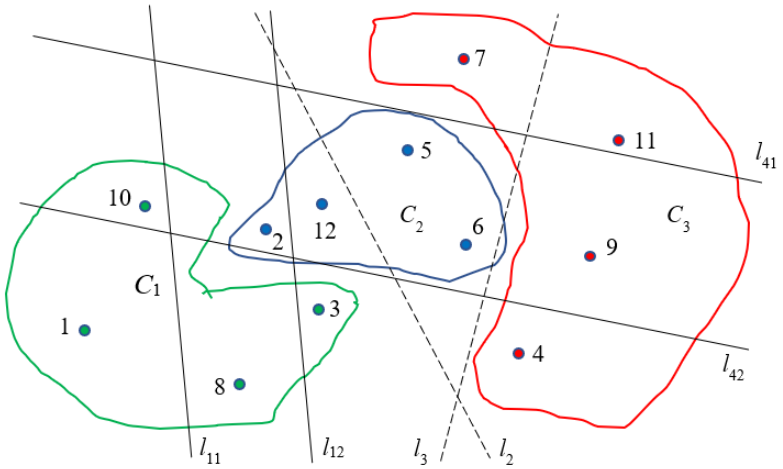

Figure 6: Separation of first eight points using BN_1, BN_4 and lines l_2, l_3

Consider the last three patterns. Notice that when dealing with the separation of many patterns, it is not always easy to determine whether these patterns are properly separated using bithreshold neurons. This problem can be solved by analyzing the output of all neurons involved in the separation process. The outputs of the neurons from Figure 6 are presented in Table 1.

Table 1
Outputs of neurons

Pattern	Class	Outputs			
		BN_1	TN_2	TN_3	BN_4
1	C_1	−1	−1	−1	−1
2	C_2	1	−1	−1	1
3	C_1	−1	−1	−1	−1
4	C_3	−1	1	1	-1
5	C_2	−1	1	−1	1
6	C_2	−1	1	−1	1
7	C_3	−1	1	−1	−1
8	C_1	1	−1	−1	−1
9	C_3	−1	1	1	1
10	C_1	−1	−1	−1	1
11	C_3	−1	1	1	−1
12	C_2	−1	−1	−1	1

In the case of proper pattern separation, no two identical output rows correspond to patterns from different classes. It is true for the first eight patterns. The output row for pattern 9 is unique. Hence, this pattern is correctly separated. The same is true for pattern 10. Consider pattern 11. Its row of outputs is identical to the row corresponding to pattern 4. However, it is acceptable because both patterns belong to the same class C_3. Note again time that these patterns fall into different regions of the plane in Figure 6, but these regions are identical with respect to the outputs of neurons. This apparent ambiguity arises from the fact that both the first and third region, induced by a single BN, produce the same output −1.

Consider the last pattern. It belongs to class C_2 and shares identical outputs with pattern 10, which represents class C_1. Thus, an additional line is required to separate these two patterns. This can be done by extending one of single-threshold neurons, TN_2 or TN_3. Let us extend TN_3 into BN_3 as shown in Figure 7.

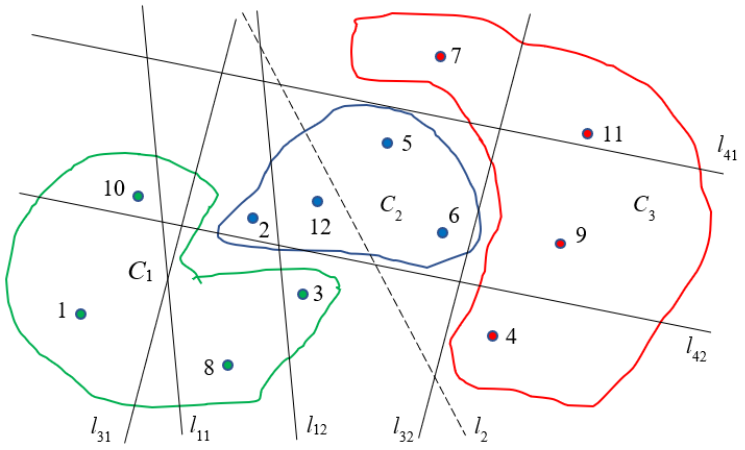

Figure 7: Separation of all patterns using BN_1, TN_2, BN_3 and BN_4

Note that the replacing of TN_3 with BN_3 results in the change of the output table. The values in the column BN_3 are negated for patterns 2, 4, 5, 6, 7, 8, 9, and 11. The updated outputs are presented in Table 2.

Table 2
Updated outputs of neurons

Pattern	Class	Outputs			
		BN_1	TN_2	BN_3	BN_4
1	C_1	−1	−1	−1	−1
2	C_2	1	−1	1	1
3	C_1	−1	−1	1	−1
4	C_3	−1	1	−1	−1
5	C_2	−1	1	1	1
6	C_2	−1	1	1	1
7	C_3	−1	1	1	−1
8	C_1	1	−1	1	−1
9	C_3	−1	1	−1	1
10	C_1	−1	−1	−1	1
11	C_3	−1	1	−1	−1
12	C_2	−1	−1	1	1

It is evident from Table 2 that the above changes do not cause any new conflicts, whereas patterns 10 and 12 have different rows of outputs.

All neurons obtained during the separation phase are potential candidates for use in a future classifier (as we will see later, they are useful in the first hidden layer of the corresponding neural network). However, the set of neurons obtained during the separation may be redundant. This means that some subset of neurons may performs the same separation as the full set does. Let us return to our example. Suppose that BN_1 is excluded from the separation process. This results in the removal of the corresponding column from Table 2. The result is shown in Table 3.

Table 3
Output table after removal of BN_1

Pattern	Class	Outputs		
		TN_2	BN_3	BN_4
1	C_1	−1	−1	−1
2	C_2	−1	1	1
3	C_1	−1	1	−1
4	C_3	1	−1	−1
5	C_2	1	1	1
6	C_2	1	1	1
7	C_3	1	1	−1
8	C_1	−1	1	−1
9	C_3	1	−1	1
10	C_1	−1	−1	1
11	C_3	1	−1	−1
12	C_2	−1	1	1

It is easy to verify that there are no equal rows for patterns belonging to different classes. Thus, it is possible to remove BN_1 without the loss of separability of all patterns presented in the training sample (the redundancy of this neuron follows from the fact that it separates only patterns 1 and 3

from pattern 2, but BN_4 also does it). The other three neurons are significant because their removal breaks the valid separability.

The table of outputs can be used in order to produce a classifier, but it requires simplification to reduce its size.

Let us remove the duplicate rows of outputs (in the table corresponding to a valid separation, such rows are possible only for patterns that are members of the same class). There are four pairs of identical rows highlighted in Table 3: namely, 3, 8 for C_1; 4, 11 for C_3; 2, 12 and 5, 6 for C_2. Thus, it is possible to safely remove rows 6, 8, 11, and 12 without loss of information. The result is presented in Table 4, where patterns are grouped by class.

Table 4

Output table after removal of 4 redundant rows

Pattern	Class	Outputs		
		TN_2	BN_3	BN_4
1	C_1	–1	–1	–1
3	C_1	–1	1	–1
10	C_1	–1	–1	1
2	C_2	–1	1	1
5	C_2	1	1	1
4	C_3	1	–1	–1
7	C_3	1	1	–1
9	C_3	1	–1	1

Note that Table 4 contains all eight 3-dimensional Boolean vectors. Therefore, it can be used to classify any new 2-dimensional pattern P. To do so, we simply compute the outputs of TN_2, BN_3 and BN_4, respectively, and assign the pattern to the class whose representative in Table 4 shares the same output as P. The plane partition performed by this classifier is shown in Figure 8. For the given P, outputs of neurons are $(1, 1, -1)$. This vector matches the penultimate row of Table 4, which correspond to the representative of class C_3. Therefore, our classifier would assign pattern P to class C_3.

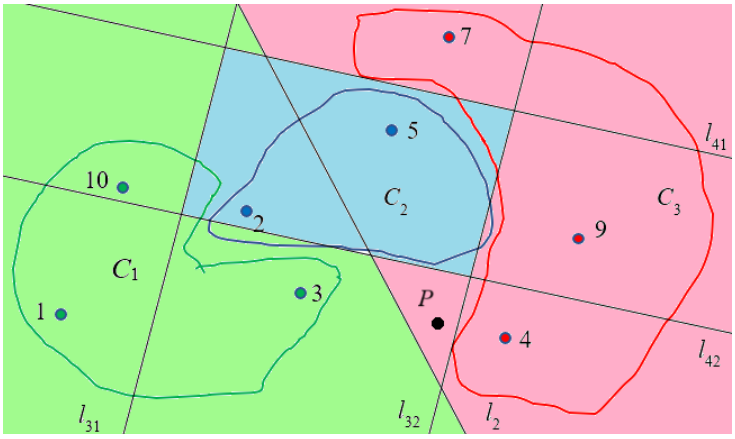

Figure 8. Separation of the plane using TN_2, BN_3 and BN_4 after simplification

It is clear from Figure 8 that the resulting class boundaries are only a rough approximation. Consequently, corresponding classifier may perform with low accuracy. This can be explained by the small size of training sample (only 12 patterns were used for illustration purposes) as well as certain properties of bithreshold activation function (1). These issues will be discussed later.

3.2. Hybrid 3-layer neural network classifier with hidden bithreshold layer

3.2.1. Architecture of classifier

Consider a neural-like multilayer feed-forward model of 3-layer classifier, whose principles of operation were described in the previous subsection. It is evident that the first hidden layer (i.e., the first network layer after its input layer, which is not taken into account) must consist of neurons that provide the desired separation of training patterns. This layer contains bipolar-valued bithreshold nodes as well as single-threshold ones. Let N be the number of these nodes. Then, the first layer performs a mapping from R^n to $\{-1, 1\}^N$.

The second hidden layer contains M nodes and serves as a bridge between the bithreshold and the output layers. It is constructed using the output table that is associated with the first hidden layer and can be considered as an $M \times N$ bipolar matrix V consisting of unique rows $v_1, \ldots, v_M$, where M is a number of rows each of which is an N-dimensional bipolar vector containing outputs of all first-layer neurons. It is evident that $M \le \min\{m, 2^N\}$. Let us assume that first M_1 rows of matrix B correspond to class C_1, next M_2 rows—to class C_2, ..., last M_K rows—to class C_K, where $M_1 + \ldots + M_K = M$, $M_i \ge 1$, $i = 1, \ldots, K$. Let I_k denote the set of indices of all rows corresponding to class C_k as, where $k = 1, \ldots, K$. For the sake of brevity, assume that all redundant patterns have already been removed from the training sample, so that $M = m$. Assume that ith unit in the second layer corresponds to the vector b_i and can recognize the input pattern x_i, with the class label y_i. Therefore, this unit may compare the output vector $z = z(x)$ from the first layer with the vector v_i and activate himself only in the case $z = v_i$ and transmit its activation to the next layer. However, this exact-match approach cannot be applied in practice. This is caused by two reasons: 1) there is no guaranty that for each of 2^N possible bipolar vectors z there exists a corresponding "prototype" in the training sample; 2) for large datasets the size of the first layer N may be so large that the use of the second layer consisting of 2^N units is not feasible. Thus, the relationship between z and v_i should use the proximity instead of the equality. Therefore, only a unit will be activated for which the distance between z and v_i is minimal. If the distance is defined as Euclidian distance, such behavior can be implemented using a layer of neural units with an appropriate activation. It follows from the fact that

$$\|\mathbf{z} - \mathbf{v}_i\|^2 = \|\mathbf{z}\|^2 - 2\mathbf{z} \cdot \mathbf{v}_i + \|\mathbf{v}_i\|^2 = 2M - 2\mathbf{v}_i \cdot \mathbf{z}.$$

In the last equation, we used that squared Euclidian norm of M-dimensional bipolar vector is equal to M. Therefore, the minimization of $\|\mathbf{z} - \mathbf{v}_i\|$ is equivalent to the maximization of the dot product $\mathbf{v}_i \cdot \mathbf{z}$. Thus, it is possible to use a layer with weight matrix V and WTA activation mode for this purpose, in which the maximum layer output is transformed to 1, and all others set to 0. An alternative approach consists in the use of the softmax activation mode instead of WTA, which provides a smoothed continuous version of WTA. This mode preserves the possibility of the membership to numerous classes for patterns lying near the decision boundaries of the classifiers. Let $s(z) = (s_1(z), \ldots, s_M(z))$ denote the output of second layer given the input z. Then,

$$s_i(\mathbf{z}) = \frac{\exp(\mathbf{v}_i \cdot \mathbf{z})}{S(\mathbf{z})}, \text{ where } S(\mathbf{z}) = \sum_{i=1}^{M} \exp(\mathbf{v}_i \cdot \mathbf{z}), \quad i = 1, \ldots, M. \tag{2}$$

The (third) output layer of classifier traditionally contains as many nodes as the number of distinct classes. It uses the linear neural units without biases and activation functions. The last layer weight matrix $U = (u_{ki})$, ($k = 1, \ldots, K$, $i = 1, \ldots, M$) is predefined as follows:

$$u_{ki} = \begin{cases} 1, & \text{if } i \in I_k, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Thus, from (2) and (3) we may conclude that

$$y_k(\mathbf{x}) = \sum_{i=1}^{M} u_{ki} s_i(\mathbf{z}(\mathbf{x})) = \sum_{i \in I_k} s_i(\mathbf{z}(\mathbf{x})), \ (k = 1, \ldots, K), \ \sum_{k=1}^{K} y_k = 1.$$

It follows from the last equation that *k*th output node indicates the predicted probability of the input x to be a member of class C_k for $k = 1, \ldots, K$.

3.2.2. Network synthesis algorithm

Consider how to synthesize the neural network classifier described in 3.2.1. The key question is the construction of the network first layer providing the desired separation of training patterns.

The synthesis algorithm consists of two stages. The first stage is most important and consists in the separation of the representatives of every class. During this stage, the current input pattern x_i ($i = 2, \ldots, m$) must be separated from the patterns that belong to other classes and have already been processed during the synthesis procedure. The conceptual scheme of this process is illustrated in Figure 9.

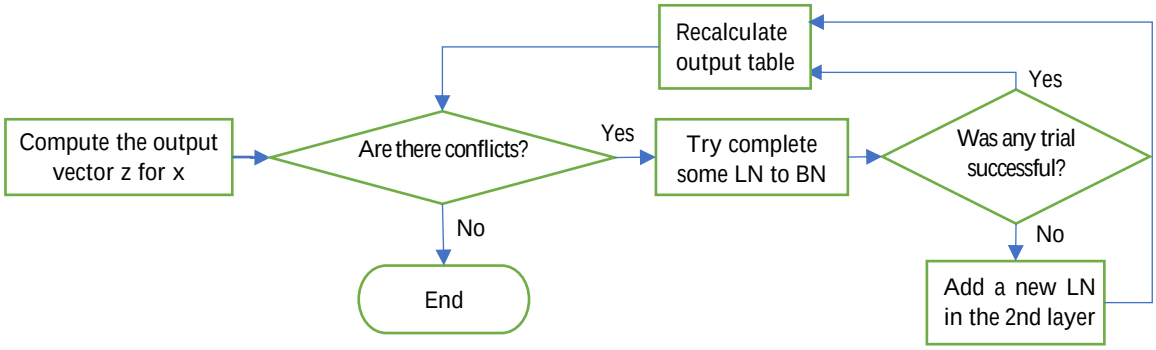

Figure 9: Flowchart of the main operation of the first stage of the synthesis

The process begins by computing the output vector z_i, which consists of the outputs of all neurons that have already been included in the first layer. If no conflict occurres (i.e., $z_i \neq z_j$ for all j such that $1 \leq j < i$ and $y_j \neq y_i$), then no changes are required. Otherwise, an additional action are necessary to resolve conflicts for some patterns x_i and x_j such that $z_i = z_j$ and $y_j \neq y_i$. First, the synthesis system attempts to resolve conflict between *i*th and *j*th patterns without inserting new nodes in the first layer. It checks whether there exists a single-threshold neural unit LN(w, *t*) that can be extend to a BN$(\mathbf{w}, t_1, t_2)$ with some new threshold t_1 and t_2, in such way that $f_{t_1, t_2}(\mathbf{w} \cdot \mathbf{x}_i) \neq f_{t_1, t_2}(\mathbf{w} \cdot \mathbf{x}_j)$ and the replacement of LN(w, *t*) with BN$(\mathbf{w}, t_1, t_2)$ does not cause any conflict among already separated patterns $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$. If it is impossible, then the algorithm proceeds by adding a new single-threshold neuron LN(w, *t*) in the layer, where.

$$\mathbf{w} = 2(\mathbf{x}_i - \mathbf{x}_j), \ t = \|\mathbf{x}_i\|^2 - \|\mathbf{x}_j\|^2 \tag{4}$$

Such choice ensures consistent separation, because $\mathbf{w} \cdot \mathbf{x}_i > t > \mathbf{w} \cdot \mathbf{x}_j$. The last inequalities follow from the fact that

$$\mathbf{w}\cdot\mathbf{x}_i-\mathbf{w}\cdot\mathbf{x}_j=2\left(\mathbf{x}_i-\mathbf{x}_j\right)^2=2\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2>0,\ t=\frac{1}{2}\left(\mathbf{w}\cdot\mathbf{x}_i+\mathbf{w}\cdot\mathbf{x}_j\right).$$

Note that if the input patterns have integer coordinates, then both w and t in (4) are also integer, which may be advantageous for certain implementation purposes [31].

The second stage of the synthesis algorithm aims to simplify the first layer of the network by eliminating redundant neurons and duplicate rows in the output matrix *V*. This stage results in an $N \times n$ real matrix *W*—weight matrix of the first network layer, as well as in $M \times N$ bipolar matrix *V*—weight matrix of the second network layer. The binary $K \times M$ weight matrix *U* of the output layer is defined by (3).

4. Experiment

The performance of proposed 3-layer neural network classifier was compared with the performance of several popular classifiers in order to estimate its ability to solve the classification task on the opdigits dataset, which is considered as small-sized benchmark in pattern classification [32]. This dataset contains 1797 8 × 8 images of one of 10 handwritten digits, with no missing values. This is a copy of the test set of the UCI ML hand-written digits dataset [33] with reduced dimensionality and some invariance to small distortions. All 64 input attributes are integers in the range 0 ... 16. A more detailed description of the model is available in [32].

During the simulation the performance of following 4 popular and 3 bithreshold-like models was compared. Popular classifiers were: decision tree, random forest (averaging algorithms based on randomized decision trees) [32], LinearSVC (support vector machine with linear kernel) [32] and MLPClassifier (multilayer perceptron classifier) [1].

The Scikit Learn library [32] implementations of popular classifiers were used with recommended parameter settings. The 2-layer bithreshold NN classifier [22] was studied as well as the 3-layer NN classifier described in the third section. The 2-layer model was tested with different values of its hyperparameter α. Two versions of 3-layer classifier were studied. The first of them used only single-threshold neurons in first hidden layer. The second version used both single-threshold and bithreshold neurons in this layer.

Main classification metrics were used: accuracy, precision, recall, and F_1 score [1]. Last three metrics were calculated for each label, and their unweighted means were found. This strategy is quite reasonable, as there was no significant imbalance between class sizes in the training sample [33]. 5-fold cross-validation [1, 32] was applied in order to obtains representative results. The simulation results will be presented and discussed in the following section.

5. Results and discussion

Simulation results are shown in Table 5.

Table 5
Experiment results on "hand-written digits" dataset

Regressor	Mean metric value			
	accuracy	precision	recall	F1 score
Decision tree	68.1	72.92	66.53	68.13
Random forest	81.64	81.99	81.3	81.91
Linear SVC	98.33	98.39	98.32	98.35
MLP classifier	98.38	98.43	98.39	98.4
2-layer bithreshold NN	74.05	72.13	73.37	72.74
3-layer single-threshold NN	79.17	79.31	78.42	78.86
3-layer bithreshold NN	92.77	93.6	92.86	92.88

By analyzing simulation results, it is possible to conclude that:

1. All but one model demonstrated higher level of precision compared to other metrics, but, in general, the difference between different metric scores was not significant.
2. Neural-based models outperformed other classifiers. MLP classifier achieved the best results by all four main metrics.
3. 3-layer bithreshold NN was third best by all metrics.
4. The results of the random search techniques showed that $\alpha \approx 1$ is preferable. Larger values of α did not provide the improvement of the performance.
5. Both 3-layer networks overperformed 2-layer one.
6. The use of the bithreshold neurons in the second network layer resulted in the significant improvement of performance. Moreover, 3-layer bithreshold NN has in average 19% fewer neurons in the first layer as well as the size of its second layer was approximately 41% smaller compared 3-layer single-threshold NN.

The 3-layer NN classifier design employs the synthesis approach. As a result, the sizes of the hidden layers depend on the specific dataset used, and even on the order in which the patterns are selected during synthesis. Experiment results show that the second network layer is quite acceptable (e.g., few dozens of units for digits dataset, typically in the range 21..40).

It seems that the main drawback of the proposed model of classifier is the large size of its second layer, which was enormous for single-threshold version (over 1,000) and also excessively large in the case of bithreshold modification. This is due to the fact that the number of duplicate rows was not very large (between 109 and 816). Therefore, second layer may remain very large.

Note that second stage (network simplification) does not significantly impact the performance of classifier during the experiment (it caused decrease in accuracy by 1%–12%). The reasons for such a performance degradation are unclear and require further investigation.

The comparison of 2-layer bithreshold NN model proposed in [22] and the current 3-layer model is also noteworthy. The results of the above experiment showed that 3-layer model had better prediction accuracy on new data. Nevertheless, the 2-layer model is much more compact and can be used for larger datasets compared to 3-layer one. Unlike the 2-layer model, the computational complexity of the proposed synthesis algorithm has not yet been analyzed and remains an open question.

The experiment also showed that the second stage of the synthesis algorithm can be significantly more expensive than the first separation stage. This is due to the need to search for identical rows in the output table, which can be large enough. The simplest implementation uses two nested loops and can be very slow. This limitation can be partially bypassed by applying the hashing to the set matrix rows.

6. Conclusions

The applications of the neural systems based on the bithreshold approach in neural computation have been considered in the paper. The model of the hybrid 3-layer neural network has been designed whose first hidden layer consists of both single-threshold and bithreshold neural units. The second hidden layer of the network contains neural units, which serve as memory cells, and uses the softmax activation principle. The last layer consists of *K* neurons with predefined weights, where *K* is the number of classes of the particular classification task for which the neural network is designed.

The proposed classifier employs a model-based approach to synthesis, using the first layer as a compressed, encoded representation of the training sample. The synthesis algorithm can be considered as an online algorithm because during the one step of the synthesis process only the current pattern is analyzed. The simulation results obtained on “optical recognition of handwritten digits” dataset demonstrated that proposed NN model is concurrent compared to popular machine

learning classifier and outperforms the 2-layer bithreshold NN synthesized using the offline algorithm from [22].

As it was mentioned in the discussion section, the proposed model of classifier is not flawless. Further studies are necessary to improve the structure of the second hidden layer, as well as reduce the impact of the order of training pairs on the size of the network and its performance.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] 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, CA, 2022.

[2] E.H. Houssein et al., Soft computing techniques for biomedical data analysis: open issues and challenges, Artificial Intelligence Review 56 (2023): 2599–2649. doi:10.1007/s10462-023-10585-2

[3] I. Izonin et al., Cascade-based input-doubling classifier for predicting survival in allogeneic bone marrow transplants: small data case, Computation 13.4 (2025): 80. doi:10.3390/computation13040080

[4] O. Mitsa et al., A comparative study of machine learning algorithms and the prompting approach using GPT-3.5 Turbo for text categorization, in: Z. Hu et al. (Eds.), Lecture Notes on Data Engineering and Communications Technologies, volume 242, Springer, Cham, 2025, pp. 156–167. doi:10.1007/978-3-031-84228-3_13

[5] Y. Andrashko et al., A method for assessing the productivity trends of collective scientific subjects based on the modified PageRank algorithm, Eastern-European Journal of Enterprise Technologies 1.4 (2023): 41–47. doi:10.15587/1729-4061.2023.273929

[6] 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.

[7] J. Kaur, J. Verma, Past, present and future of computational intelligence: a bibliometric analysis, in: AIP Conference Proceedings 2916(1), 2023, 020001. doi:10.1063/5.0177490

[8] V. K. Venkatesan et al., High-Performance artificial intelligence recommendation of quality research papers using effective collaborative approach, Systems 11.2 (2023): 81. doi:10.3390/systems11020081.

[9] S. Moon, Y. Kim, Mounting angle prediction for automotive radar using complex-valued convolutional neural network, Sensors 25.2 (2025): 353. doi:10.3390/s25020353

[10] O. Mitsa et al., Ethnocultural, educational and scientific potential of the interactive dialects map, in: IEEE International Conference on Smart Information Systems and Technologies, SIST 2023. Astana, May 4-6, pp. 226–231. doi:10.1109/SIST58284.2023.10223544

[11] V. Kotsovsky, Hybrid 4-layer bithreshold neural network for multiclass classification, in: CEUR Workshop Proceedings, volume 3387, 2023, pp. 212–223.

[12] I. Izonin et al., A hybrid two-ML-based classifier to predict the survival of kidney transplants one month after transplantation, in: CEUR Workshop Proceedings, volume 3609, 2023, pp. 322–331.

[13] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: a comprehensive survey and benchmark, Neurocomputing 503 (2022): 92–108. doi:10.1016/j.neucom.2022.06.111

[14] A. Apicella, F. Donnarumma, F. Isgrò, R. Prevete, A survey on modern trainable activation functions, Neural Networks 138 (2021): 14–32. doi: 10.1016/j.neunet.2021.01.026

[15] V. Kotsovsky, F. Geche, and A. Batyuk, Bithreshold neural network classifier, in: Proceedings of the IEEE 15th International Scientific and Technical Conference on Computer Sciences and

Information Technologies, CSIT 2020, vol. 1. Lviv, Ukraine, 2020, pp. 32–35. doi: 10.1109/CSIT49958.2020.9321883
[16] D. R. Haring, Multi-threshold building blocks, IEEE Transactions on Electronic Computers EC-15.4 (1966): 662–663.
[17] A. Ngom, I. Stojmenović, J. Zunić, On the number of multilinear partitions and the computing capacity of multiple-valued multiple-threshold perceptrons, IEEE Transactions on Neural Networks 14.3 (2003): 469–477.
[18] V. Kotsovsky, Multithreshold neurons with smoothed activation functions, in: CEUR Workshop Proceedings, volume 3983, 2025, pp. 93–102.
[19] T. Szandała, Review and comparison of commonly used activation functions for deep neural networks, Studies in Computational Intelligence, volume 903, 2021, pp. 203-224.
[20] 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.
[21] Z. Obradovic, I. Parberry, Learning with discrete multivalued neurons, Journal of Computer and System Sciences 49 (1994): 375–390.
[22] 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.
[23] V. Kotsovsky, Learning of multi-valued multithreshold neural units, in: CEUR Workshop Proceedings, volume 3688, 2024, pp. 39–49.
[24] 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.
[25] M. Anthony, J. Ratsaby, Large-width machine learning algorithm, Progress in Artificial Intelligence 9.3 (2020): 275–285.
[26] V. Kotsovsky, A. Batyuk, Multithreshold neural units and networks, in: Proceedings of IEEE 18th International Conference on Computer Sciences and Information Technologies, CSIT 2023, Lviv, Ukraine, 2023, pp. 1-5.
[27] V. Kotsovsky, Synthesis of multithreshold neural network classifier, in: CEUR Workshop Proceedings, volume 3711, 2024, pp. 75–88.
[28] T. Gowda et al., Identification of threshold functions and synthesis of threshold networks, IEEE Transaction on Computer-Aided Design, 30.5 (2011): 665-677.
[29] M. Nikodem, Synthesis of multithreshold threshold gates based on negative differential resistance devices, IET Circuits Devices Syst. 7.5 (2013): 232–242.
[30] 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.
[31] 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.
[32] F. Pedregosa et al., Scikit-learn: machine learning in Python, Journal of Machine Learning Research 12 (2011): 2825-2830.
[33] M. Kelly, R. Longjohn, K. Nottingham, The UCI machine learning repository, 2023. URL: http://archive.ics.uci.edu.