Learning of Multi-valued Multithreshold Neural Units

Vladyslav Kotsovsky[1]

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

Abstract

The issues related to the use of multithreshold neural units in multiclass classification are treated in the paper. Two models of multi-valued *k*-threshold neurons are considered. Online and offline modifications of the learning algorithm are designed to train multithreshold neuron to solve multiclass classification tasks using simple and fast learning techniques. The conditions are found ensuring the finiteness of the training. The experiment results demonstrate the performance of multithreshold multiclass classifier on real-world datasets compared to some popular classifiers.

Keywords

Multithreshold neuron, multi-valued neuron, machine learning, neural network, classification

1. Introduction

Neural-like networks and systems have numerous applications in artificial intelligence [1] and intelligent data analysis [2]. They are used in modern hardware [3] and software [4] tools and products [5, 6]. The amazing capacities of artificial neural networks (ANN) are provided by the appropriate use of the network architecture [7] and related learning techniques [8, 9].

The synergy between the network architecture, the kind of network nodes and the network learning (or synthesis) procedures is very important in the practice of neural computations [10]. Linear neural units with threshold activation functions [11], binary inputs and output were used in early models [12]. This kind of computation units was inspired by the models of biological neurons from the brain study [11]. But both the theoretical studies and practical applications showed the strong limitations of the basic neuron model of McCulloch and Pitts [12, 13] as well as difficulties related to the learning of threshold ANN [14, 15]. In order to overcome above-mentioned limitations and difficulties, many more complicated models of neural devices were proposed [11, 12]. The overall majority of these models employed two ways to increase the network capacities by enhancing the power of network neurons [10]. The first is based on the use of more sophisticated models of the aggregation of the input signals of the neural unit instead of the classical weighted sum of inputs [12], e.g., polynomial threshold units [12, 13]. The second approach consists in the use of more complicated activation functions instead of the step function [12] from the Rosenblatt model [16, 17]. Both approaches have their pros and cons discussed in [10–14]

The multithreshold models were developed under the second approach [18]. One of the earliest among them was the multithreshold threshold element [19]. Binary multithreshold neuron with weight vector $\mathbf{w}=(w_1,\ldots,w_n)\in\mathbf{R}^n$ and threshold vector $\mathbf{t}=(t_1,\ldots,t_k)\in\mathbf{R}^k$ is the computation unit with *n* inputs $x_1,\ldots,x_n$ whose single binary output *y* is calculated by the following rule:

$$y=\begin{cases}1, & \text{if } t_{2j-1}\le \mathbf{w}\cdot\mathbf{x}<t_{2j},\ j\in\{1,\ldots\lceil k/2\rceil\},\\ 0, & \text{if } t_{2j}\le \mathbf{w}\cdot\mathbf{x}<t_{2j+1},\ j\in\{0,1,\ldots\lceil k/2\rceil\},\end{cases} \tag{1}$$

COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems, April 12–13, 2024, Lviv, Ukraine

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

0000-0002-7045-7933 (V. Kotsovsky)

where $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$ is an input vector, $\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + \ldots + w_n x_n$ is the dot product of vectors **w** and **x** (weighted sum of inputs), $[k/2]$ denotes the integer part of number $k/2$, $t_1 < t_2 < \ldots < t_k$, $t_0 = -\infty$ and $t_{k+1} = +\infty$ are additional thresholds used for convenience only. Multithreshold elements outperform single-threshold ones [18, 20], because they are activated when the sum of weighted inputs is within the one if given disjoint half-open intervals, which are specified by the ordered sequence of their thresholds [21].

But the increase in the recognition capability of multithreshold is not gratuitous. One must pay a high price for this, which consists in the difficulty of the learning of such units [7, 22], because the respective learning task is NP-hard even in the case of a unit with two thresholds. The research has *two main goals*:

- The study of the model of multi-valued multithreshold neuron that should effectively use the advantages of multiple thresholds, be suitable for the multiclass classification and admits fairly simple training techniques.
- The development of the learning algorithm for such units and the study of its fitness for intended applications in classification.

The paper has the following structure. First, the works related to the topic of the study will be reviewed. Then, two models of multithreshold neural units will be considered: binary-valued and multi-valued, respectively. We will discuss its advantages and consider some downsides related to the complexity of their learning. In the next section two learning algorithms will be described, which are designed for the learning of a single *k*-threshold neuron. For both algorithms the conditions on the learning rate will be stated, which satisfy the finiteness of the learning in the case of their application to the learning of strongly *k*-separable sets. Next, the simulation results will be treated of the performance of trained multiclass *k*-threshold neural classifiers in the comparison with some other popular classifiers provided by Sklearn library [11]. Finally, two last sections contain the discussion of obtained results and conclusions.

2. Related works

The study of multithreshold neural units has a long history [19, 23, 24]. Multithreshold neural elements were introduced in the early studies in threshold logic [19, 25]. As mentioned above, the additional thresholds were proposed with intention to increase the capacities of basic single-threshold element [19, 26]. Some properties of multithreshold neurons were stated in [22, 25, 26]. These works mostly dealt with the recognition capacity of multithreshold elements [2]. Issues related to the synthesis of multithreshold devices remained almost untouched, because few algorithms for training such multithreshold units and networks had been developed [18, 24]. Therefore, the applications of devices using multithreshold approach were almost unknown [27] despite the better capabilities of multithreshold units compared to the classical linear threshold units [20, 26]. The hardness results from [15, 22] can explain these difficulties for the practical application of bithreshold systems to some extent. Nevertheless, as stated in [8, 10, 28], the lack of learning techniques for multithreshold systems caused the decline of interest in their study.

But recent advances in multithreshold logic changed the situation [7, 14]. One of the reasons were new approaches in the synthesis ANN with hidden layers consisting of neurons with bithreshold activation functions [14, 20]. They were developed on the base of the generalization of the Baum's synthesis algorithm [29] for threshold networks in the case of bithreshold nodes [14, 28].

The advance in the application of so-called bithreshold networks was stated in [1, 10], where such networks were considered as the effective tools, which are capable to solve typical problems of intellectual data processing and computational intelligence. The limitations and downsides of the basic bithreshold ANN from [7, 14] were stated in [28]. Hybrid models of the multiclass classifier with heterogenous hidden layers were proposed in [28], where other kinds of neural units (e.g., WTA and single-threshold) units were used in order to enhance network performance and reduce its drawbacks. It should be noted that bithreshold ANN can be useful not only in classifiers. Their potential applications are considerably wider [2, 6, 8, 9]. E.g., they were mentioned

in design of powerful deep ANN providing the exponential improvement of the memorization capacity [16]. The bithreshold approach primary was employed for the solution of real-valued problems [10]. But it admits the generalization to the complex domain [14]. The complex analogs of bithreshold activation could be proposed [30] that extend the capacity of complex-valued threshold neural units. This allows the multithreshold approach in the proceeding of data in the complex domain [17, 28].

It should be noted that the above-mentioned advance in the application of multithreshold systems is actually related to only bithreshold models [7]. The examples of successful application of general multithreshold models with an arbitrary number of thresholds are unknown [14, 30]. It became evident that the additional study is necessary before such models can be employed in machine learning systems [10]. One of them was the paper [22], where general *k*-threshold neural units were treated in the case $k \geq 2$. As was observed in [22], the parity of *k* has the great influence to the properties of multithreshold neurons. Moreover, every multithreshold unit can be realized using a small threshold circuit, and, consequently, every multithreshold network can be replaced by the equivalent networks consisting solely of bithreshold and threshold nodes [30]. Notice also that unlike the learning of a single threshold linear unit, the learning of a multithreshold unit proved to be NP-hard [22] confirming the similar result of the intractability of the learning of a single bithreshold unit [14].

Notice that all mentioned applications of bithreshold and *k*-threshold neurons have the binary outputs [28]. Thus, their employment in the classifiers requires the special shape of the network output layer with a separate neuron for every class and the using of "one versus all" approach in the learning or synthesis [11]. In some cases, a single output multi-valued neuron is preferable [12], because its application results in the network having fewer nodes and weight coefficients.

3. Models and methods

3.1. Two models of multithreshold neural units

3.1.1. Model of binary-valued *k*-threshold neuron

Let us consider again a model of *k*-threshold binary-valued neuron with the *weight vector* **w** and (ordered) *threshold vector* **t**, which output is given by (1). Note that its performance can be described as follows:

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} < t_1, \\ 1, & \text{if } t_1 \leq \mathbf{w} \cdot \mathbf{x} < t_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ (1+(-1)^k)/2, & \text{if } t_{k-1} \leq \mathbf{w} \cdot \mathbf{x} < t_k, \\ (1+(-1)^k)/2, & \text{if } t_k \leq \mathbf{w} \cdot \mathbf{x}. \end{cases} \tag{2}$$

Model (2) has a simple geometrical interpretation [22, 26]. The family of parallel hyperplanes $H_j : \mathbf{w} \cdot \mathbf{x} = t_j, \ j \in \{1, \ldots, k\}$ divides the space $\mathbf{R}^n$ by $k+1$ parts, which can be successively labeled by numbers 0, 1, ..., *k*. All points belonging to "even" parts are attributed as "negative" ones. Remaining parts are considered as "positive" [22]. The illustration is shown in Figure 1, where the case $n = 2, k = 3$ is considered.

Figure 1 can also illustrate the nature of difficulties related to the application of binary-valued multithreshold neuron. Its value can alternate many times. It ensures the great capability of the multithreshold unit on the one hand, but results in the hardness of its training on the other hand. Note that the strict proof of the NP-hardness of the learning of a single binary-valued multithreshold neuron can be found in [22].

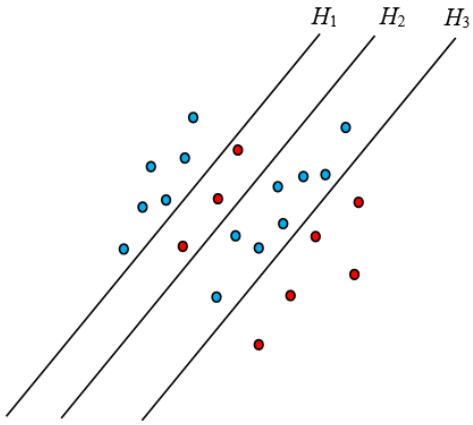

Figure 1: Illustration of the performance of binary-valued 3-threshold neuron

3.1.2. Model of multi-valued *k*-threshold neuron

The multi-valued modification of the model (2) can be considered [18, 23] that keeps the capacity of the base model and is easier in the training [24]. This multithreshold model uses the same weight vector **w** and threshold vector **t,** but differs in the output range of the neuron. To be more precise, the range set of *k*-threshold multi-valued neuron is $\mathbf{Z}_{k+1} = \{0,1,,...,k\}$, and the neuron output *y* satisfies the following condition:

$$y = f_{\mathbf{t}}(\mathbf{w} \cdot \mathbf{x}), \tag{3}$$

where

$$f_{\mathbf{t}}(x) = \begin{cases} 0, & \text{if } x < t_1, \\ 1, & \text{if } t_1 \le x < t_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ k-1, & \text{if } t_{k-1} \le x < t_k, \\ k, & \text{if } t_k \le x. \end{cases} \tag{4}$$

Consider again the geometrical illustration, now, for the *k*-threshold multi-valued neuron (3), (4). As it is shown in Figure 2, the performance of the neuron is also defined by parallel hyper-planes $H_j : \mathbf{w} \cdot \mathbf{x} = t_j$, $j \in \{1, ..., k\}$, which make partition of the space $\mathbf{R}^n$ by $k+1$ parts.

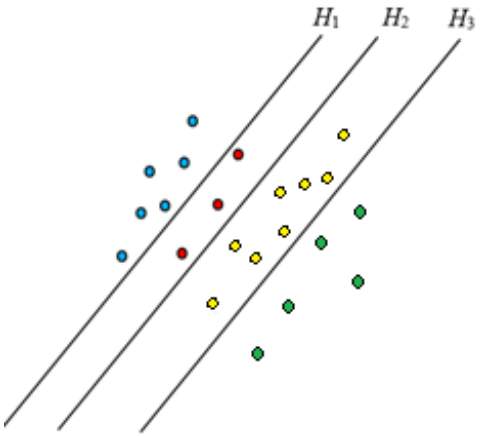

Figure 2: Illustration of the performance of multi-valued 3-threshold neuron

These parts also are labeled by indices 0, 1, ..., k corresponding to the output value of the (multi-valued) neuron whose activation is given by (4). Notice that same points are used in both Figure 1 and Figure 2, but their partition by classes differs, because there are only two classes for binary-valued k-threshold neuron and $k+1$—for its many-valued counterpart [22].

The pair (**w**, **t**) completely defines the multi-valued multithreshold neuron and is called its *structure pair*. Let A be an arbitrary set in $\mathbf{R}^n$. Then every multi-valued k-threshold neuron with structure pair $(\mathbf{w},\mathbf{t})$ performs the (ordered) partition $(A_0, A_1, ..., A_k)$ of the set A, where:

$$A_i = \{\mathbf{x} \in A \mid f_{\mathbf{t}}(\mathbf{w} \cdot \mathbf{x}) = i\},\ i = 0,1,...,k \tag{5}$$

This partition is called an ordered k-threshold partition of the set A, whereas sets $A_0, A_1, ..., A_k$ are called strongly *k-separable* (compare with [22]). Note that the order matters for the strongly separated sets. Sets $A_0, A_1, ..., A_k$ are called *k-separable*, if there exists a permutation $\pi : \mathbf{Z}_{k+1} \to \mathbf{Z}_{k+1}$ such that sets $A_{\pi(0)}, A_{\pi(1)}, ..., A_{\pi(k)}$ are strongly k-separable [22].

3.2. Learning algorithms for multithreshold neurons

3.2.1. Initial reduction of the task

Let $A_0, A_1, ..., A_k$ be strongly k-separable finite sets. Consider the task of the search of a multi-valued k-threshold neuron with structure pair $(\mathbf{w},\mathbf{t})$ that performs the desired partition $(A_0, A_1, ..., A_k)$ of the set A that is the union of (disjoint) sets $A_0, A_1, ..., A_k$, which satisfies (5).

Consider how one can reduce the above task to the solution of the homogenous system of linear inequalities in $n+k$ variables $w_1, ...w_m, t_1, ..., t_k$. It is possible to rewrite (3)-(5) as follows:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x} < t_1, & \text{if } \mathbf{x} \in A_0, \\ t_j \le \mathbf{w} \cdot \mathbf{x} < t_{j+1}, & \text{if } \mathbf{x} \in A_j\ (1 < j < k), \\ \mathbf{w} \cdot \mathbf{x} \ge t_k, & \text{if } \mathbf{x} \in A_k. \end{cases} \tag{6}$$

Since sets $A_0, A_1, ..., A_k$ are finite and strongly k-separable, system (6) has solutions, which compose n-dimensional convex set. If all non-strict inequalities in (6) were replaced by strict ones, then resulting system would also have solutions. Let $\mathbf{v} = (w_1, ..., w_n, -t_1, ..., -t_k)$,

$$\mathbf{a}_j(x_1, ..., x_n) = (x_1, ..., x_n, \underbrace{0, ..., 0}_{j-1}, 1, \underbrace{0, ..., 0}_{k-j}). \tag{7}$$

The chained inequality $t_j < \mathbf{w} \cdot \mathbf{x} < t_{j+1}$ is equal to the system

$$\begin{cases} \mathbf{w} \cdot \mathbf{x} - t_j > 0, \\ -\mathbf{w} \cdot \mathbf{x} + t_{j+1} > 0. \end{cases}$$

The last system can be rewritten in the following way:

$$\begin{cases} \mathbf{a}_j(\mathbf{x}) \cdot \mathbf{v} > 0, \\ -\mathbf{a}_{j+1}(\mathbf{x}) \cdot \mathbf{v} > 0. \end{cases} \tag{8}$$

Thus, we can reduce system (6) to the following system:

$$\begin{cases} \mathbf{b}_1 \cdot \mathbf{v} > 0, \\ \ldots\ldots\ldots\ldots \\ \mathbf{b}_m \cdot \mathbf{v} > 0 \end{cases} \tag{9}$$

where $m = 2|A| - |A_0| - |A_k|$ ($|X|$ denotes the cardinality of the set X) and vectors $\mathbf{b}_i$ are obtained using (7) and (8). Note that there are algorithms solving (9) in polynomial time [13]. Thus, the task of the learning of k-threshold multi-valued neuron (3)-(4) is not NP-complete.

The reduction process can be described using the following pseudocode:

ReduceSet$(A_0, A_1, \ldots, A_k)$

```
1  B ← ∅
2  for x in A_0:
3      add −a_1(x) into B
4  for i in {1,...,k−1}:
5      for x in A_i:
6          add a_i(x) into B
7          add −a_{i+1}(x) into B
8  for x in A_k:
9      add a_k(x) into B
10 return B
```

Notice that the transformation (7) is used in steps 3, 6, 7, 9 ensuring the filling of the output set B.

3.2.2. Online learning algorithm

Consider the training of the multi-valued k-threshold neural unit to separate finite strongly k-separable sets $A_0, A_1, \ldots, A_k$.

Let us describe the online learning algorithm for a k-threshold multi-valued neural unit that uses ReduceSet$(A_0, A_1, \ldots, A_k)$ from the previous subsection and an adopted version of the relaxation algorithm from [31, 32]. The pseudocode of the algorithm is shown in the function Online-Multithreshold:

OnlineMultithreshold$(A_0, A_1, \ldots, A_k, r, \mathbf{v}^0, \eta)$

```
1  B ← ReduceSet(A_0, A_1, ..., A_k)
2  v ← v^0
3  (i, j, err) ← (0,0,1)
4  while i < r and err > 0:
5      err ← 0
6      shuffle B
7      for b in B:
8          s ← b · v
9          if s > 0:
10             continue
11         j ← j + 1
12         err ← err + 1
```

13 $\mathbf{v} \leftarrow \mathbf{v} - \dfrac{\eta(j)s}{\|\mathbf{b}\|^2}\mathbf{b}$

14 $i \leftarrow i+1$

15 $\mathbf{w} \leftarrow (v_1,...,v_n)$

16 $\mathbf{t} \leftarrow (-v_{n+1},...,-v_{n+k})$

17 `return` $\mathbf{w},\mathbf{t}$

Previous algorithm has four main parameters: $(A_0, A_1,..., A_k)$ — an ordered partition corresponding to strongly *k*-separable sets, *r*—the number of learning epochs, $\mathbf{v}^0$ — initial approximation, η —the schedule function that defines the behavior of the learning rate. The above algorithm uses three internal counters: *i* that is responsible for learning epochs, *j*—responsible for learning corrections, and *err*—responsible for the unit errors during the current epoch of learning. The goal of algorithm is the search of a vector $\mathbf{v} \in \mathbf{R}^{n+k}$ such that for all $\mathbf{b} \in B$ the inequality $\mathbf{v} \cdot \mathbf{b} > 0$ holds. If such vector is already found, then the learning process terminates. Otherwise, the weight correction occurs in step 13 at least once per epoch. Note that this correction is successful only in the case $s \neq 0$. Thus, a random initial approximation should be used for $\mathbf{v}^0$ to avoid the situation $s = 0$ during the learning. The following proposition states conditions ensuring the successful completion of the online learning using above algorithm.

Proposition 1. If $A = A_0 \cup A_1 \cup ... \cup A_k$, sets $A_0, A_1,..., A_k$ are finite and strongly *k*-separable,

$$\eta(j) = \eta'(j) + \frac{\eta''(j)}{s(j)},$$

where *j* is a correction step, *s*(*j*) is the dot product obtained in step 8 before *j*th correction,

$$0 \le \eta'(j) \le 2,\ 0 < \eta''_{\min} \le \eta''(j) \le \eta''_{\max}, \tag{10}$$

then there exists r such that OnlineMultithreshold produces a structure pair $(\mathbf{w},\mathbf{t})$ of multi-valued *k*-threshold neuron, which satisfies (6) and performs desired partition of the set *A*.

3.2.3. Offline learning algorithm

Let us describe the offline approach to the learning of *k*-threshold multi-valued neural unit. It is designed using the modification of offline spectral algorithm from [32] adopted to solving the system (9). Let $B = \{\mathbf{b}_1,...,\mathbf{b}_m\}$ be a finite subset of $\mathbf{R}^{n+k}$, and $\mathbf{v} \in \mathbf{R}^{n+k}$. We will need the following notations:

$$\mathbf{s}(B) = \sum_{i=1}^{m} \mathbf{b}_i,\ g_{\mathbf{v}}(\mathbf{b}) = \operatorname{sgn}(\mathbf{v} \cdot \mathbf{b}),\ \mathbf{g}_{\mathbf{v}}(B) = \left(g_{\mathbf{v}}(\mathbf{b}_1),...,g_{\mathbf{v}}(\mathbf{b}_m)\right),\ \mathbf{s}_{\mathbf{v}}(B) = \sum_{i=1}^{m} g_{\mathbf{v}}(\mathbf{b}_i)\mathbf{b}_i,\ \mathbf{1} = \underset{n+k}{(1,...,1)}.$$

Note that both $\mathbf{s}(B)$ and $\mathbf{s}_{\mathbf{v}}(B)$ belong to $\mathbf{R}^{n+k}$ and vector $\mathbf{s}_{\mathbf{v}}(B)$ can be considered as an analogs of Fourier coefficients of the function $\mathbf{g}_{\mathbf{v}} : B \to \{-1,0,1\}$. Consider the following algorithm:

OfflineMultithreshold $(A_0, A_1,\ldots, A_k, r, \mathbf{v}^0, \eta)$

1 $B \leftarrow \text{ReduceSet}(A_0, A_1,\ldots, A_k)$

2 $\mathbf{v} \leftarrow \mathbf{v}^0$

3 compute $\mathbf{s}(B)$

4 compute $\mathbf{g}_{\mathbf{v}}(B)$

5 $j \leftarrow 0$

6 while $j<r$ and $\mathbf{g}_{\mathbf{v}}(B)\neq\mathbf{1}$:

7 $j\leftarrow j+1$

8 compute $\mathbf{s}_{\mathbf{v}}(B)$

9 $\mathbf{v}\leftarrow\mathbf{v}-\dfrac{\eta(i)\left(\mathbf{s}(B)-\mathbf{s}_{\mathbf{v}}(B)\right)\cdot\mathbf{v}}{\left\|\mathbf{s}(B)-\mathbf{s}_{\mathbf{v}}(B)\right\|^2}\left(\mathbf{s}(B)-\mathbf{s}_{\mathbf{v}}(B)\right)$

10 compute $\mathbf{g}_{\mathbf{v}}(B)$

11 $\mathbf{w}\leftarrow(v_1,\ldots,v_n)$

12 $\mathbf{t}\leftarrow(-v_{n+1},\ldots,-v_{n+k})$

13 return $\mathbf{w},\mathbf{t}$

Note that OfflineMultithreshold$(A_0,A_1,\ldots,A_k,r,\mathbf{v}^0,\eta)$ has identical input parameters as its online counterpart from the previous subsection.

The following proposition states conditions ensuring the successful completion of the offline learning using above algorithm.

Proposition 2. If $A=A_0\cup A_1\cup\ldots\cup A_k$, sets $A_0,A_1,\ldots,A_k$ are finite and strongly *k*-separable,

$$\eta(j)=\eta'(j)+\frac{\eta''(j)}{\mathbf{v}(j-1)\cdot\left(\mathbf{s}_{\mathbf{v}(j)}(B)-\mathbf{s}(B)\right)},$$

where *j* is a correction step, $\eta'(j)$ and $\eta''(j)$ satisfy (10), $\mathbf{v}(j)$ is a value of vector $\mathbf{v}$ after *j*th correction, then there exists r such that OfflineMultithreshold$(A_0,A_1,\ldots,A_k,r,\mathbf{v}^0,\eta)$ produces the structure pair $(\mathbf{w},\mathbf{t})$ of a multi-valued *k*-threshold neuron, which performs desired *k*-threshold partition of the set *A*.

Proofs of both propositions are omitted. They can be obtained using reasons similar to [32].

4. Experiment and results

Consider the capability of our learning algorithms from the previous section to train a multi-valued multithreshold-based classifier to solve the classification problems on some benchmarks. Let us compare their performance with well-known classification methods, such as classical perceptron, nearest neighbor classifier, random forest and feed-forward ANN (multilayer perceptron). Classifiers were compared on the following two real-world datasets: “balance-scale” (Balance Scale Weight & Distance Database) and “dry-bean” (Dry Bean Dataset) [33, 34] provided by UC Irvine Machine Learning Repository [35]. The datasets contain 625 and 13611 learning instances from 3 and 7 classes, respectively [33, 35]. The first dataset has 5 features, the second one—16 [33]. 25% instances of every dataset were used as the test set, and the rest 75%—as the training set. In order to obtain consistent results [12], the repeated random subsampling validation [11, 36] was used. The learning experiments were repeated 500 times for every dataset and then obtained results were averaged concerning the accuracy on the training and test sets.

Default values of parameters recommended by Scikit-Learn library were used during training experiments for first four classical classifiers: 5 neighbors for nearest neighbor classifier, 1000 iterations for linear perceptron classifier, unbounded depth for random forest, one hidden layer with 100 nodes and 200 iterations for multilayer perceptron [36]. The constant learning rate $\eta=2$ was used for both MultiThreshold algorithm as well as random initial approximations $\mathbf{w}^0,\mathbf{t}^0$. Datasets are not provided with an ordered partition into classes [35]. So, the classes were ordered using the alphabetical order induced by their labels. The following table contains results of experiments.

Table 1
Simulation results on two real-world datasets

Classifier	Accuracy on training set (in %)		Accuracy on test set (in %)	
	balance-scale	dry-bean	balance-scale	dry-bean
Perceptron	84.25	20.91	82.23	16.59
5-Nearest Neighbor	88.07	81.07	81.85	72.34
Random Forest	100	99.98	82.93	90.03
MLP Classifier	95.28	53.51	92.74	49.60
OnlineMultithreshold	88.84	57.23	83.89	51.22
OfflineMultithreshold	88.03	66.02	82.16	59.83

By analyzing data from Table 1, we can conclude that:

- Both multithreshold algorithms performed well on the relatively easy small 3-class classification task on balance-scale dataset and the online modification had the second-best accuracy on the test set.
- Classification on the dry-bean dataset was more difficult task for almost all classifiers considered during simulation. Learning for both linear perceptron and multilayer perceptron failed completely. Multi-valued multithreshold neuron yielded by OfflineMultithreshold performed better than neuron produced by online algorithm and had the best accuracy among all neural-like models, which were considered. But its accuracy was considerably worse than in the case of the use of random forest classifier.

5. Discussions

Two versions of the learning algorithm for multi-valued multithreshold neurons have been proposed. The simulation results prove that both algorithms are capable to yield networks, which are suitable for the solution of classification problems in the case when the number of classes is relatively small. But the performance of both algorithms decreases in the case when the number of classes increases. It seems that it is due to at least two reasons.

The first one is the small number of parameters of the multithreshold model compared to other classifiers, which often use “one versus all” scheme [11, 36]. It seems that above drawback can be overcome by using multithreshold networks [29] or more powerful neuron models with multithreshold activation, e.g., polynomial neurons [23, 30, 32].

The second reason is caused by the nature of the datasets related to majority of classification problems. They contain training pairs, each of which consists of a pattern and its class label. In terms of the partition, we deal with an unordered partition while proposed learning algorithms are designed to process with strongly k-separable sets corresponding to an ordered partition. The question arises how to convert an unordered partition to an ordered one. The brute force is not effective due to fast growth of factorial. Numerous heuristics can be used in order to increase the performance of the multithreshold neurons. This is a problem that deserves a separate consideration.

6. Conclusions

The problem of the application of multithreshold multi-valued neural units has been considered. These units separate the sets of patterns in n-dimensional vector space using parallel hyperplanes. This ability allows them to become candidates for computational nodes of multiclass ANN classifiers. Thus, the development of learning methods for such networks is important.

The simplest case of this learning problem has been treated, namely, issues concerning the learning of a single multi-valued multithreshold neuron. Two approaches to the training of multithreshold neuron have been developed. Both of them require the simple preliminary patterns transformation in order to reduce a given multiclass task to corresponding binary classification

task. The online version of the learning algorithm is simpler and often faster. The offline modification performs single correction during each learning epoch, usually is more expensive but often yield the neuron having a somewhat better accuracy of classification. The conditions have been stated ensuring the finiteness of the learning process in the case of application of both algorithms to the training of *k*-separable sets.

References

[1] V.K. Venkatesan, M.T. Ramakrishna, A. Batyuk, A. Barna, B. Havrysh, High-Performance artificial intelligence recommendation of quality research papers using effective collaborative approach, Systems 11.2 (2023): 81. doi:10.3390/systems11020081.

[2] I. Izonin, R. Tkachenko, S. A. Mitoulis, A. Faramarzi, I. Tsmots, D. Mashtalir, Machine learning for predicting energy efficiency of buildings: a small data approach, in: Procedia Computer Science, volume 231, 2024, pp. 72–77. doi:10.1016/j.procs.2023.12.173.

[3] F. Geche, V. Kotsovsky, A. Batyuk, 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. doi:10.1109/STC-CSIT.2015.7325432.

[4] 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. doi:10.15587/1729-4061.2020.195041.

[5] M. Havryliuk, N. Hovdysh, Y. Tolstyak, V. Chopyak, N. Kustra, Investigation of PNN optimization methods to improve classification performance in transplantation medicine, in: CEUR Workshop Proceedings, volume 3609, 2023, pp. 338–345.

[6] O. Mitsa, V. Sharkan, V. Maksymchuk, S. Varha, H. Shkurko, Ethnocultural, educational and scientific potential of the interactive dialects map, in: Proceedings of 2023 IEEE International Conference on Smart Information Systems and Technologies (SIST), Astana, 2023, pp. 226-231. doi: 10.1109/SIST58284.2023.10223544.

[7] 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.

[8] R. Tkachenko, An integral software solution of the SGTM neural-like structures implementation for solving different Data Mining tasks, in: S. Babichev, V. Lytvynenko (Eds.), Lecture Notes on Data Engineering and Communications Technologies, volume 77, Springer, Cham, 2022, pp. 696–713.

[9] M. Lupei, O. Mitsa, V. Sharkan, S. Vargha, N. Lupei, 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.

[10] E.H. Houssein, M.E. Hosney, M.M. Emam, E.M. Younis, A.A. Ali, W.M. Mohamed, Soft computing techniques for biomedical data analysis: open issues and challenges, Artificial Intelligence Review 56 (2023): 2599–2649.

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

[12] P. Setoodeh, S. Habibi, S. Haykin, Nonlinear Filters: Theory and Applications, Wiley, New York, NY, 2022.

[13] M. Anthony, J. Ratsaby, Large-width machine learning algorithm, Progress in Artificial Intelligence 9.3 (2020): 275–285.

[14] V. Kotsovsky, A. Batyuk, Feed-forward neural network classifiers with bithreshold-like activations, in: Proceedings of IEEE 17th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT 2022, Lviv, Ukraine, 2022, pp. 9–12.

[15] A. Blum, R. Rivest, Training a 3-node neural network is NP-complete, Neural Networks 5.1 (1992): 117–127.

[16] S. Rajput, K. Sreenivasan, D. Papailiopoulos, A. Karbasi, An exponential improvement on the memorization capacity of deep threshold networks, in: Advances in Neural Information Processing Systems, volume 16, 2021, pp. 12674–12685.
[17] Z.-G. Zhang, Y.-L. Xiao, J. Zhong, Unitary learning in conditional models for deep optics neural networks, in: Proceedings of SPIE – The International Society for Optical Engineering, volume 12565, 2023, no. 1256543.
[18] N. Jiang, Z. Zhang, X. Ma, J. Wang, Y. Yang, Analysis of nonseparable property of multi-valued multi-threshold neuron, in: Proceedings of 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, China, 2008, pp. 413-419, doi: 10.1109/IJCNN.2008.4633825
[19] D. R. Haring, Multi-threshold threshold elements, IEEE Transactions on Electronic Computers EC-15.1 (1966): 45–65.
[20] I. Prokíc, Characterization of multiple-valued threshold functions in the Vilenkin-Chrestenson basis, Journal of Multiple-Valued Logic and Soft Computing 34.3-4 (2020): 223–238.
[21] R. Takiyama, The separating capacity of a multithreshold threshold element, IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-7.1 (1985): 112–116.
[22] 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, doi: 10.1109/CSIT61576.2023.10324129.
[23] 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.
[24] M. Anthony, Learning multivalued multithreshold functions, CDMA Research Report No. LSE-CDMA-2003-03, London School of Economics, 2003.
[25] V.K. Venkatesan, I. Izonin, J. Periyasamy, A. Indirajithu, A. Batyuk, M.T. Ramakrishna, Incorporation of energy efficient computational strategies for clustering and routing in heterogeneous networks of smart city, Energies 15.20 (2022): 7524.
[26] 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.
[27] I. Izonin, B. Ilchyshyn, R. Tkachenko, M. Gregus, N. Shakhovska, C. Strauss, Towards data normalization task for the efficient mining of medical data, in: Proceedings of 12th International Conference on Advanced Computer Information Technologies, ACIT 2022, Ruzomberok, Slovakia, 2022, pp. 480–484.
[28] V. Kotsovsky, "Hybrid 4-layer bithreshold neural network for multiclass classification," in CEUR Workshop Proceedings, volume 3387, 2023, pp. 212–223.
[29] E. B. Baum, On the capabilities of multilayer perceptrons, Journal of Complexity 4.3 (1988): 193–215.
[30] V. Kotsovsky, A. Batyuk, V. Voityshyn, On the size of weights for bithreshold neurons and networks, in: Proceedings of IEEE 16th International Conference on Computer Sciences and Information Technologies, CSIT 2021, Lviv, Ukrain, 2021, volume 1, pp. 13–16.
[31] S. Dasgupta, S. Sabato, Robust learning from discriminative feature feedback, in: Proceedings of Machine Learning Research, volume 108, 2020, pp. 973–982.
[32] V. Kotsovsky, A. Batyuk, On-line relaxation versus off-line spectral algorithm in the learning of polynomial neural units, in: S. Babichev et al., (Eds.), Communications in Computer and Information Science, volume 1158, Springer, Cham, 2020, pp. 3–21.
[33] OpenML: A worldwide machine learning lab, 2024. URL: https://openml.org.
[34] M. Lupei, M. Shlahta, O. Mitsa, Y. Horoshko, H. Tsybko, V. Gorbachuk, Development of an interactive map within the implementation of actual state and public directions, in: Proceedings of the 12th International Conference on Advanced Computer Information Technologies, ACIT 2022, Ruzomberok, Slovakia, 2022, pp. 384–387.
[35] M. Kelly, R. Longjohn, K. Nottingham, The UCI machine learning repository, 2023. URL: http://archive.ics.uci.edu.
[36] S. Pölsterl, Scikit-survival: A library for time-to-event analysis built on top of Scikit-learn, Journal of Machine Learning Research 21 (2020): 1–6.