Hyperparameter tuning in the learning of multithreshold neurons

Vladyslav Kotsovsky*

State University “Uzhhorod National University”, Narodna Square 3, Uzhhorod, 88000, Ukraine

Abstract
The modification of the online learning algorithm for multi-valued multithreshold neurons is proposed in the paper. Conditions are stated and proved that ensure the finite successful learning. The influence of the algorithm hyperparameters on the learning process is analyzed on the base of simulation results. The recommendations are formulated concerning the choice of values of these hyperparameters, which may significantly reduce the learning time. The experiment results prove that the proposed algorithm overperforms ‘incremental’ algorithm suggested by Obradović and Parberry. Obtained results can be useful in the design of artificial neural network classifiers employing multithreshold activation functions in network nodes.

Keywords
Multithreshold neural unit, classification, machine learning

1. Introduction

Neural networks (NN) became mainstream in modern artificial intelligence (AI) systems [1] and smart data proceeding [2, 3]. Both hardware [4] and software infrastructure of AI [1] widely employs concepts and solutions based on neural-like approaches [5, 6]. Different network architectures [7] as well as appropriate learning and synthesis techniques [8, 9] provide powerful capacities of artificial NNs in the solving numerous real-time problems. Modern NN-based AI systems depend on billions of parameters [2] and their behavior is influenced by many hyperparameters [9], which are used in the learning of the underlying machine learning (ML) model [10]. This implies the importance of the proper choice of these hyperparameters during the training process in order to adopt AI system to the solution of the given ML problem [11].

The tremendous power of latest AI systems is provided by the capability of underling NN [12]. Therefore, the main efforts in neural computation are devoted to the improvement of the network capacities [2]. It can be achieved in many ways [13]. The most popular one consists in the increasing of the network size by using deeper models with many neurons in every hidden layer [9], as well as the application of new hybrid network architectures, e.g., as in [2, 5, 6]. This approach can be extremely successful, but usually it requires considerable computation resources and may be very chip and inappropriate in many cases [2, 9].

The second approach consists in the use of a relatively small NN enhanced by the application of modified network nodes, which are more powerful than usual linear neural units with RELU- or sigmoid-like activation functions [14]. In simplest cases a single such unit is sufficient to solve a classification task on small- or medium-sized dataset [11, 15].

In order to overcome the limitation of classical neural units, many modified models were proposed, e.g., in [2, 9, 16]. They all were intended to increase the recognition capacity of a single neuron. As mentioned in [16], they can be divided into at least two classes.

The first class contains models using the modified modes of the aggregation of the input signals of the neural unit instead of the usual weighted sum of inputs. This approach includes different

ICST-2024: Information Control Systems & Technologies, September, 23 – 25, 2024, Odesa, Ukraine

* *Corresponding author.*

vladyslav.kotsovsky@uzhnu.edu.ua (V. Kotsovsky)
0000-0002-7045-7933 (V. Kotsovsky)

kernel models, which make the shape of decision region of the neural unit more complicated and more appropriate to the distribution of data patterns [2, 9].

The second class consists of models that benefit of the use of a modified activation function [17]. This class is sometimes more useful than the first one, because its representatives adopt the kind of activation to the particular task without adding many new parameters to the ML model [16]. Note that this approach requires the development of special learning techniques adapted to the chosen modification of the activation function [9, 18].

The current research is devoted to the study of the one kind of neural models belonging to the second class—the multi-valued multithreshold neural unit [19, 20]. The goal of the research is the design of the learning algorithm for such multithreshold units and the investigation what values of algorithm hyperparameters would be used in order to speed-up the training process and improve the capacity of resulting neuron.

2. Related works

Multithreshold approach was proposed in the early studies in threshold logic [17–19, 21]. The first models employed the multithreshold binary-valued activation in order to enhance the capacity of the classical threshold gate based on the famous McCulloch and Pitts model [22]. This enhancement was theoretically confirmed in [23, 24], where it was shown that a linear threshold unit strengthened by additional thresholds considerably overperforms single-threshold gate. The explanations and quantitative expressions of the increase of the unit capacity can be found in [17, 24]. Despite the strict confirmation and justification of the advantage of multithreshold models in pattern classification, the practical benefits of this approach were almost missing, because few synthesis (as well as learning) algorithms were proposed for such multithreshold systems. And this, in order, implies the decline of the interest in the development and the use of multithreshold models and systems [7].

Hardness results for multithreshold units stated in [16, 20] explain that the learning task for a multithreshold unit is considerably harder in the sense of complexity theory than similar task for a single-threshold unit. This conclusion was also confirmed for general multithreshold neural units with an arbitrary number of thresholds in [25]. Paper [20] also contains the result concerning the connection between multithreshold neurons and single-threshold neural networks with a single hidden layer.

Nevertheless, in [6, 26, 27] some recent advances were observed in the application of bithreshold and multithreshold neural units and networks, respectively. In the bithreshold case it was caused by new approaches in the synthesis of NN by employing bithreshold neurons in hidden layers of networks [16]. This approach can be combined with the reducing of drawbacks of bithreshold activations [23] by making network deeper using hybrid blocks, which consist of group of heterogeneous neurons preserving the information concerning the location of training patterns [16]. The similar approach was proposed in [23], where the smoothed modification of activation function was used as well as neuron center defined by the portion of training patterns, which activate this neuron.

In the multithreshold case the progress is related to the use of multi-valued outputs instead of binary ones [28]. This leads to lesser complexity of the learning task compared to the case of the application of binary-valued neurons, because the complexity of the learning of multi-valued multithreshold neurons proved to be equal to the complexity of the learning of linear single-threshold units [29, 30].

3. Models and methods

The multithreshold multi-valued model of the neuron will be considered in this section as well as issues related to its learning.

3.1. Model of multi-valued multithreshold neural unit

Consider a model of multithreshold neuron. It is a computation unit provided with weight vector $\mathbf{w}=(w_1,\ldots,w_n)\in\mathbf{R}^n$ and ordered threshold vector $\mathbf{t}=(t_1,\ldots,t_k)\in\mathbf{R}^k$. Each weight w_i is associated with corresponding input x_i, $i=1,\ldots,n$. The use of multiple thresholds allows the neuron to operate in two modes: binary-valued and multi-valued, respectively [30]. Further only multi-valued neurons will be considered. The unit output is denoted by *y* and is defined in the following way:

$$y = \begin{cases} 0, & \text{if } \mathbf{w}\cdot\mathbf{x} < t_1, \\ 1, & \text{if } t_1 \le \mathbf{w}\cdot\mathbf{x} < t_2, \\ \dots\dots\dots\dots\dots\dots & \\ k-1, & \text{if } t_{k-1} \le \mathbf{w}\cdot\mathbf{x} < t_k, \\ k, & \text{if } t_k \le \mathbf{w}\cdot\mathbf{x}, \end{cases} \tag{1}$$

where $\mathbf{w}\cdot\mathbf{x}$ denotes the inner products of the weight vector w and the input vector x.

It is evident that the neural unit describing by equation (1) has $k+1$ different values. Therefore, we can use it as a single output node of NN classifier in the case when the number of classes is greater by 1 than the number of thresholds.

The pair (w, t) completely defines the multi-valued multithreshold neuron. Further, this pair will be used as the short name for "multi-valued multithreshold neuron with weight vector w and threshold vector t".

Let *A* be an arbitrary set of patterns in *n*-dimensional real space. Every multi-valued *k*-threshold neuron $(\mathbf{w},\mathbf{t})$ induces the ordered partition $(A_0, A_1, \dots, A_k)$ of the set A, where the set A_i contains all elements of the set *A* such that $t_i \le \mathbf{w}\cdot\mathbf{x} < t_{i+1}, (i = 0,\dots,k)$. Note that two additional pseudo-thresholds $t_0 = -\infty$ and $t_{k+1} = +\infty$ were used in the previous equation for convenience.

This partition is called an ordered *k*-threshold partition of the set *A* by strongly *k-separable* sets $A_0, A_1, \dots, A_k$ [30]. Notice that the order matters for such partitions.

3.2. Learning of multi-valued *k*-threshold neural unit

Two algorithms for single multi-valued multithreshold neuron were proposed in [30]. This subsection contains a brief description of the modification of the first one.

Consider the search for a multi-valued *k*-threshold neuron $(\mathbf{w},\mathbf{t})$ that performs the desired ordered partition $(A_0, A_1, \dots, A_k)$ of the finite set *A*. We can consider the elements of the set *A* as members of our training set. It is evident that without loss of generality one can replace all non-strict inequalities in (1) by strict one (this is true, because *A* is finite). Furthermore, it is also easy to show that the learning task is equivalent to the solution of the following system of linear inequalities:

$$\begin{cases} t_0 < \mathbf{w}\cdot\mathbf{x} < t_1, & \text{if } \mathbf{x} \in A_0, \\ t_1 < \mathbf{w}\cdot\mathbf{x} < t_2, & \text{if } \mathbf{x} \in A_1, \\ \vdots & \\ t_{k-1} < \mathbf{w}\cdot\mathbf{x} < t_k, & \text{if } \mathbf{x} \in A_{k-1}, \\ t_k < \mathbf{w}\cdot\mathbf{x} < t_{k+1}, & \text{if } \mathbf{x} \in A_k. \end{cases} \tag{2}$$

Note that similar as in the definition of ordered partition, two additional sentinel thresholds $t_0 = -\infty$ and $t_{k+1} = +\infty$ were used in (2) in order to simplify notations. There exists, in addition, ML-like interpretation of the solution of (2). We can consider it as the task of supervised learning on the dataset consisting of training pairs $(\mathbf{x}, y_\mathbf{x})$, where $\mathbf{x} \in A$, $y_\mathbf{x} = i$ if and only if $\mathbf{x} \in A_i$.

3.2.1. Data preprocessing

Consider the method of the transformation of the task (2) to the solution of the homogenous system of linear inequalities in $n+k$ variables $w_1, \dots w_m, t_1, \dots, t_k$, which was proposed in [30].

Let us search for solution vector in the form $\mathbf{v} = (w_1, \dots, w_n, -t_1, \dots, -t_k)$, which contains all sough weights as well as all negated thresholds. Consider the sequence of transformations $f_j : \mathbf{R}^n \to \mathbf{R}^{n+k}$
:

$$f_j\left(x_1,...,x_n\right) = (x_1,...,x_n,\underbrace{0,...,0}_{j-1},1,\underbrace{0,...,0}_{k-j}),\ \left(j=1,\ldots,k\right). \tag{3}$$

It follows from (3) that every chained inequality $t_j < \mathbf{w}\cdot\mathbf{x} < t_{j+1}$ in (2) is equivalent to the following system:

$$\begin{cases} f_j\left(\mathbf{x}\right)\cdot\mathbf{v} > 0, \\ -f_{j+1}\left(\mathbf{x}\right)\cdot\mathbf{v} > 0. \end{cases} \tag{4}$$

Thus, it is possible to reduce (2) to the solution of homogenous system:

$$\begin{cases} \mathbf{b}_1\cdot\mathbf{v} > 0, \\ \ldots\ldots\ldots\ldots\ldots \\ \mathbf{b}_m\cdot\mathbf{v} > 0 \end{cases} \tag{5}$$

where vectors b$_i$ are obtained using (3) and (4), (i = 1, ..., m). Note that in the case of the use of pseudo-thresholds $t_0 = -\infty$ and $t_{k+1} = +\infty$ (5) consists of exactly 2|A| inequalities, where |A| is a cardinality of the set A, but we can drop "dummy" inequalities involving pseudo-thresholds. Thus, the actual value of m is $2|A| - |A_0| - |A_k|$. Let $V(B)$ denotes the set of all solution of (5).

The reduction process was described in detail in [30], where the corresponding function ReduceSet$\left(A_0, A_1, \ldots, A_k\right)$ was defined, which returns the set $\left\{\mathbf{b}_1,\ldots,\mathbf{b}_m\right\}$.

3.2.2. Online algorithm with shift

Consider the online-version of the learning algorithm for a multi-valued k-threshold neural unit. The idea of this algorithm is from [30] and it actually derives many steps of Motzkin' relaxation algorithm for systems of linear inequalities [9]. The pseudocode of this algorithm is shown below:

```
ShiftedMultithreshold$(A_0, A_1, \ldots, A_k, r, \sigma, \mathbf{v}^0, \eta, d)$
1  $B \leftarrow$ NormalizedSet$\left(A_0, A_1, \ldots, A_k\right)$
2  $\mathbf{v} \leftarrow \mathbf{v}^0$
3  $\left(i, j, err\right) \leftarrow \left(0, 0, 1\right)$
4  while $i < r$ and $err > 0$:
5      $err \leftarrow 0$
6      shuffle $B$
7      for $\mathbf{b}$ in $B$:
8          $s \leftarrow \mathbf{b}\cdot\mathbf{v}$
9          if $s > 0$:
10             continue
11         $j \leftarrow j+1$
12         $err \leftarrow err+1$
13         $\mathbf{v} \leftarrow \mathbf{v} + \eta\left(j\right)\left(d - \sigma s\right)\mathbf{b}$
14     $i \leftarrow i+1$
15 $\mathbf{w} \leftarrow \left(v_1,...,v_n\right)$
16 $\mathbf{t} \leftarrow \left(-v_{n+1},...,-v_{n+k}\right)$
17 return $\mathbf{w},\mathbf{t}$
```

Above algorithm has the single parameters $(A_0, A_1, ..., A_k)$—an ordered partition consisting of strongly k-separable sets. Algorithm has also five hyperparameters: r—the upper bound on the number of learning epochs, σ—a binary value defining the learning mode, $\mathbf{v}^0 \in \mathbf{R}^{n+k}$—an initial approximation, η —the schedule function defining the value of the learning rate, and *d*—non-negative real value, which is a measure of the shift used during each correction. They all are used in crucial step 13, where the correction of the vector v is performed. The learning process continue until we find such vector $\mathbf{v} \in \mathbf{R}^{n+k}$ that all inequalities in (5) are satisfied. If it is not true, then there exists a vector b such that $\mathbf{b} \cdot \mathbf{v} < 0$. This vector b is used in step 13 in order to “improve” the current value of the vector v by nudging it in the direction of b. Note that inner product is used in this step to define the correction step as well as shift *d* and the current value of the learning rate.

It should be mentioned that considered algorithm differs from the similar algorithm from [30] only in steps 1 and 13, respectively. In step 1 and additional preprocessing transformation is performed, which consists in the normalization of the elements of the set *B* in order to obtain the set of vectors with unit Euclidean norm. The proposed modification of learning algorithm uses also an additional hyperparameter *d* in step 13, which should be non-negative. This allows us to avoid the possible convergence to the point lying on the bounding surface of the set *V*(*B*). Notice also that the correction is performed only in the case $s \le 0$. Hence, during every iteration performed in steps 7-13 by all elements of the set *B* the value of $d - \sigma s$ is always equal to $d + \sigma|s|$, if correction step 13 was reached.

The issues related to the convergence of the above algorithm will be considered in the next subsection.

3.3. Convergence conditions for learning algorithms

Let us consider theoretical foundation of the above algorithm ensuring its convergence and even finiteness.

Proposition. If finite sets $A_0, A_1, ..., A_k$ are strongly *k*-separable,

$$\eta(j) = \eta_1(j) + \frac{\eta_2(j)}{d + \left|\mathbf{b}^j \cdot \mathbf{v}^{j-1}\right|}, \tag{6}$$

where $\mathbf{b}^j$ is a train vector used in *j*th correction, $\mathbf{v}^{j-1}$ is the value of sought vector v after previous correction and

$$0 \le \eta_1(j) \le 2,\ 0 \le \eta_2(j) \le \eta_{\max},\ \eta_1(j) + \eta_2(j) \ge \eta_{\min}, \tag{7}$$

where $\eta_{\min}$ and $\eta_{\max}$ are arbitrary positive constants, then there exists r such that after at most *r* corrections ShiftedMultithreshold yields a multi-valued *k*-threshold neuron $(\mathbf{w}, \mathbf{t})$, which produces the partition $(A_0, A_1, ..., A_k)$.

Proof. Consider two possible cases for the value of hyperparameter σ:

1. $\sigma = 0$;
2. $\sigma = 1$.

In the first case the learning process is similar to the classical perceptron learning with the learning rates $d\eta(j)$ used in *j*th correction or its extension in the case of multi-valued multithreshold functions proposed by Obradović and Parberry (see [11]). It is well known [11] that the equality

$$\lim_{r \to \infty} \frac{\sum_{j=1}^{r} \eta^2(j)}{\left(\sum_{j=1}^{r} \eta(j)\right)^2} = 0 \tag{8}$$

is the sufficient condition of the finiteness of the learning. Let us prove that (8) follows from the correction rule in step 13 and conditions (6), (7).

Prove first that the sequence $(S_r)_{r\in\mathbf{N}}$ is divergent, where $S_r = \sum_{j=1}^{r} \eta(j)$ (note that the denominator of the fraction in (8) contains squared value of S_r). Suppose the contrary. Then

$$\left|\mathbf{b}^j \cdot \mathbf{v}^{j-1}\right| = \left|\mathbf{b}^j\left(\mathbf{v}^0 + d\eta(1)\mathbf{b}^1 + \ldots + d\eta(j)\mathbf{b}^{j-2}\right)\right| \le \left|\mathbf{b}^j \cdot \mathbf{v}^0\right| + d\max\left\{\left|\mathbf{b}^j \cdot \mathbf{b}\right| : \mathbf{b}\in B\right\}\sum_{r=1}^{\infty}\eta(r) \le D$$

for some positive constant *D*, because dot product of unit vectors does not exceed 1.

Therefore, $\eta(j) \ge \eta_1(j) + \dfrac{\eta_2(j)}{d+D} \ge \eta_{\min}\min\left(1,(d+D)^{-1}\right)$. This implies that (S_r) is divergent. Thus, our assumption about the convergence of the sequence (S_r) was wrong. Therefore, in the conditions of proposition this sequence always diverges.

Consider the numerator in (8). We can split the corresponding sum into two parts:

$$\sum_{j=1}^{r}\eta^2(j) = \sum_{j:\eta(j)<1}\eta^2(j) + \sum_{j:\eta(j)\ge 1}\eta^2(j).$$

Let S_r' be the first sum in the previous equation. It is evident that

$$S_r' = \sum_{j:\eta(j)<1}\eta^2(j) < \sum_{j:\eta(j)<1}\eta(j) \le \sum_{j=1}^{r}\eta(j) = S_r.$$

Therefore,

$$\frac{S_r'}{S_r^2} \le \frac{S_r}{S_r^2} = \frac{1}{S_r} \xrightarrow[r\to\infty]{} 0\,.$$

Consider S_r''—the second sum in the corresponding equation.

$$S_r'' = \sum_{j:\eta(j)\ge 1}\left(\eta_1(j) + \frac{\eta_2(j)}{d + \left|\mathbf{b}^j\cdot\mathbf{v}^{j-1}\right|}\right)^2 \le \sum_{j:\eta(j)\ge 1}\left(2 + \frac{\eta_{\max}}{d}\right)^2 = n_r\left(2 + \frac{\eta_{\max}}{d}\right)^2,$$

where n_r is the number of terms in S_r''. If for all *r* numbers n_r are bounded by $n_{\max}\in\mathbf{N}$, then

$$\lim_{r\to\infty}\frac{S_r''}{S_r^2} = \left(2 + \frac{\eta_{\max}}{d}\right)^2\frac{n_{\max}}{\lim\limits_{r\to\infty}S_r^2} = 0\,.$$

Otherwise, let us estimate S_r^2:

$$S_r^2 = \left(\sum_{j=1}^{r}\eta(j)\right)^2 \ge \left(\sum_{j:\eta(r)\ge 1}^{r}\eta(j)\right)^2 \ge n_r^2.$$

Hence,

$$\lim_{r\to\infty}\frac{S_r''}{S_r^2} \le \lim_{r\to\infty}\frac{n_r\left(2+\dfrac{\eta_{\max}}{d}\right)^2}{n_r^2} = \left(2 + \frac{\eta_{\max}}{d}\right)\lim_{r\to\infty}\frac{1}{n_r} = 0.$$

Therefore,

$$\lim_{r\to\infty}\frac{1}{S_r^2}\sum_{j=1}^{r}\eta^2(j) = \lim_{r\to\infty}\left(\frac{S_p'}{S_r^2} + \frac{S''}{S_r^2}\right) = 0\,,$$

and (7) holds.

Consider now the case $\sigma = 1$. Let us prove that the sequence $\left(\mathbf{v}^j\right)_{j=0,\ldots,r}$ satisfies Fejér condition

$$\left\|\mathbf{v}^j - \mathbf{v}\right\| < \left\|\mathbf{v}^{j-1} - \mathbf{v}\right\|, \tag{9}$$

for all $\mathbf{v} \in V(B)$. It is evident that (8) is equivalent to $\|\mathbf{v}^j - \mathbf{v}\|^2 < \|\mathbf{v}^{j-1} - \mathbf{v}\|^2$. Since

$$\|\mathbf{v}^j - \mathbf{v}\|^2 = \|\mathbf{v}^j - \mathbf{v}^{j-1}\|^2 + 2(\mathbf{v}^j - \mathbf{v}^{j-1}) \cdot (\mathbf{v}^{j-1} - \mathbf{v}) + \|\mathbf{v}^{j-1} - \mathbf{v}\|^2,$$

Fejér condition (9) is satisfied if

$$\|\mathbf{v}^j - \mathbf{v}^{j-1}\|^2 + 2(\mathbf{v}^j - \mathbf{v}^{j-1}) \cdot (\mathbf{v}^{j-1} - \mathbf{v}) < 0$$

for all $\mathbf{v} \in V(B)$.

We can rewrite the step 13 of the learning algorithm in the following way:

$$\mathbf{v}^j = \mathbf{v}^{j-1} + \eta(j)(d - \mathbf{b}^j \cdot \mathbf{v}^{j-1})\mathbf{b}^j. \quad (10)$$

Therefore, it is possible to rewrite the last inequality as follows:

$$\eta^2(j)(d - \mathbf{b}^j \cdot \mathbf{v}^{j-1})^2 \|\mathbf{b}^j\|^2 + 2\eta(j)(d - \mathbf{b}^j \cdot \mathbf{v}^{j-1})\mathbf{b}^j \cdot (\mathbf{v}^{j-1} - \mathbf{v}) < 0.$$

Remember that $\mathbf{b}^j \cdot \mathbf{v}^{j-1} \le 0$ in every correction. Thus, $d - \mathbf{b}^j \cdot \mathbf{v}^{j-1} = d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|$ and the last quadratic inequality holds only if

$$0 < \eta(j) < \frac{2(\mathbf{v} - \mathbf{v}^{j-1}) \cdot \mathbf{b}^j}{d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|}.$$

We can rewrite this inequality in a following form:

$$0 < \eta(j) < 2\left(1 + \frac{\mathbf{v} \cdot \mathbf{b}^j - d}{d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|}\right). \quad (11)$$

Let us slightly relax Fejér condition from the whole set *V*(*B*) to its own subsets. By using the techniques describing in [5], it is easy to verify that for every $d \ge 0$ and every $\delta > 0$ the cone *V*(*B*) contains such point $\mathbf{v} = \mathbf{v}(d, \delta)$ that, the unit closed ball $B_1[\mathbf{v}] = \{\mathbf{x} \in \mathbf{R}^{n+k} : \|\mathbf{x} - \mathbf{v}\| \le 1\}$ is the subset of *V*(*B*) and for all $\mathbf{x} \in B_1[\mathbf{v}]$ $\mathbf{v} \cdot \mathbf{x} > d + \delta$. Therefore, it follows from (11) that sequence $(\mathbf{v}^j)$ satisfies Fejér condition (9) for the ball $B_1[\mathbf{v}]$ if

$$0 < \eta(j) < 2\left(1 + \frac{\delta}{d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|}\right). \quad (12)$$

Let $\delta = \eta_{\max} / 2$. If (6) and (7) are satisfied, then (12) holds.

Suppose that the learning process is infinite, i.e., for all *r* ShiftedMultithreshold is unable to produce $\mathbf{v}^j \in V(B)$ for some $j \le r$. Then the sequence $(\mathbf{v}^j)_{j=0,\dots,r}$ satisfies is Fejér condition (9) for the ball $B_1[\mathbf{v}]$ and, hence, is convergent by well-known fact from the theory of linear normed spaces [9].

Consider the increment vectors $\Delta\mathbf{v}^j = \mathbf{v}^j - \mathbf{v}^{j-1}$. It follows from (6), (7), (10) and (12) that

$$\Delta\mathbf{v}^j = \left(\eta_1(j) + \frac{\eta_2(j)}{d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|}\right)(d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|)\mathbf{b}^j = \left(\eta_1(j)(d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|) + \eta_2(j)\right)\mathbf{b}^j.$$

It implies

$$\|\Delta\mathbf{v}^j\| = \left(\eta_1(j)(d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|) + \eta_2(j)\right)\|\mathbf{b}^j\| \ge \eta_{\min} \min\left((d + |\mathbf{b}^j \cdot \mathbf{v}^{j-1}|), 1\right) \cdot 1 \ge \eta_{\min} \min(d, 1).$$

It follows from the last equation that increment vectors do not go to zero as *j* goes to infinity. Therefore, the sequence $\left(\mathbf{v}^j\right)$ is not convergent. This apparent contradiction completes the proof in the case $\sigma = 1$.

Note that in the case $\sigma = 1$ convergence conditions (without the proof) appeared for the first time in [30].

4. Experiment

In the above theoretical study of the issues related to the algorithm convergence and finiteness the range of feasible values for the learning rate hyperparameter was found, but proved Proposition does not suggest what values are preferable in order to ensure the faster convergence.

This question can be clarified by empirical study of the dependence of ShiftedMultithreshold on different strategies to the choice of the values of hyperparameters, which are used in this algorithm.

During simulation *k*-threshold neurons were trained for different $2 \le k \le 10$. This range of values was chosen in accordance with recommendation from the paper [30]. Randomly generated *k*-threshold neuron was used to produce a partition $\left(A_0, A_1, \dots, A_k\right)$ of the set *A* containing *M* uniformly distributed point from *n*-dimensional hypercube $[-1,1]^n$, where cartesian product is used for the power in the previous formula. Two series of experiments were performed. In the first series whole *A* was used as training set. In the second—*A* was randomly split into training set and test set, where test set contained 20% of all points. The first series of experiment was more intensive. Only it was used to determine values of last four hyperparameters of algorithm, which then was used in the second series. For this reason, the most part of this section is devoted to the description of the experiments of the first type.

Note that the value of *r* was not studied in the first series of experiments and constant upper bound 100,000 was used for the number of learning epochs in the first experiment. The reaching of this bound during learning considered as signal that algorithm failed to learn neuron to solve a given task. In the next experiments *r* was reduced to 1,000.

The final value of the counter *j*, which corresponds to the total number of corrections performed during the learning process, was considered as the performance metric. Therefore, the further conclusions like "*X* performed better than *Y* by 30%" literally mean "the number of corrections in the case of *X* was lesser by 30% than the number of corrections in the case of *Y*".

The general tendence during the first series of experiments remained the same for every value of *k* from the above-mentioned range. For this reason, results will be presented only for a single value of *k*, namely, $k = 3$. This implies that 4-valued units will be considered.

The dimension of the feature space *n* was chosen to be 50. Different sizes of the training set were tried. In the next section results for *M* from {256, 512, 1024, 2048, 4096} will be presented. Random sampling was used. Each experiment was repeated for 110 times (more precisely, 11 random partitioning were performed for every of 10 randomly chosen sets *A*) and 5 best and 5 worst results were rejected in order to avoid outliers. The remaining 100 results were averaged. The obtained means will be analyzed in the next section.

The first experiment consists in the estimation of the influence of the value of binary hyperparameter σ to the performance of the learning algorithms with random initial approximation (more precisely, random uniformly distributed in (−1, 1) numbers were used as coordinates of $\mathbf{v}^0$). The constant learning rate $\eta = 2$ was used for both possible values of hyperparameter σ. This value is suggested by [9] as recommended in the case of relaxation-based algorithms. Note that application of any constant learning rate η means that for all *j* $\eta_1\left(j\right) = \eta$, $\eta_2\left(j\right) = 0$, because otherwise it follows from (6) that $\eta\left(j\right)$ depends on *j*. The case $\sigma = 0$ corresponds to the fixed increment used in classical perceptron-like models. The opposite case $\sigma = 1$ leads to relaxation-like learning in which the increment in the *j*th correction is adaptive and depends on the classification error on the current training pattern $\mathbf{b}^j$ measured by $\mathbf{b}^j \cdot \mathbf{v}^{j-1}$ in (10). It was observed that relaxation-like approach to the learning considerably overperformed the perceptron-like one in the online learning of 4-valued 3-threshold neurons. The grid search in segment [1, 2] was also performed with the step 0.01 in the case $\sigma = 0$, but it did not make significant impact to the difference of learning times for both above-mentioned types of the increment (actually, the change of η influenced the performance for relaxation-like mode much stronger than for perceptron-like one). Therefore, the perceptron-like approach to online learning was rejected, the value $\sigma = 1$ was fixed, and, consequently, only relaxation-like online learning studied in all next experiments. During the second simulation the choice of initial approximation was considered alongside with constant learning rate. The learning with randomly chosen $\mathbf{v}^0$ was compared with optimized initial approximation $\tilde{\mathbf{v}}^0 = (\mathbf{b}_1 + \ldots + \mathbf{b}_m)/m$, where $m = |B|$. Both the idea and justification of such approximation are from [30]. The idea of the use of $\tilde{\mathbf{v}}^0$ is suggested by the fact that its coordinates have signs and ordering similar to same characteristics of coordinates of feasible solution from the set *V*(*B*). For all considered *M* and η results for $\tilde{\mathbf{v}}^0$ were on average at least twice as good as for a random $\mathbf{v}^0$. For this reason, only $\tilde{\mathbf{v}}^0$ was used further. Next simulation was devoted to the search of the appropriate values for the first term $\eta_1(j)$ of the learning rate in (6). In order to reduce the impact of the second term in (6) $\eta_2(j) = 0$ was used here. The quite simple constant schedule strategy was used, i.e., η was assigned to $\eta_1(j)$ (and, consequently, to $\eta(j)$) in every correction step. None of tried η outside the segment [1.23, 2.31] was successful and only $\eta \in [1.5, 2.2]$ performed well. For this reason, the grid search on [1.5, 2.2] with the step 0.001 was used to determine $\eta^{(M)}$—empirically the best η for the given *M*. Further simulation was devoted to the search of the appropriate values for the second term $\eta_2(j)$ of the learning rate in (6). It was observed that only constant $\eta_2(j) = \eta_2 \in [0.1, 0.5]$ provided good performance. Another grid search was performed on two dimension to find $(\eta_1^{(M)}, \eta_2^{(M)})$ – empirically the best pair for the given *M*. In the next simulation the impact of the value of the shift hyperparameter *d* was studied. The learning rate was calculated by using (6) and pairs $(\eta_1^{(M)}, \eta_2^{(M)})$ from the previous experiment. The last simulation was the second series of experiments. Previously found values of hyperparameters were tried to solve the classification task on the split dataset in order to estimate the generalization ability of multi-valued *k*-threshold neuron in the case of different $2 \le k \le 10$.

5. Results and discussion

Consider results that were obtained in above-mentioned experiments. Table 1 contains comparative results of perceptron-like $(\sigma = 0)$ and relaxation-like $(\sigma = 1)$ learning mode, respectively, in the case of the learning of 3-threshold neuron with constant learning rate 2.

It is evident from Table 1 that the learning mode has great impact on the performance.

Table 1
Performance comparison for different values of hyperparameter σ

Dataset size	Average number of corrections	
	$\sigma = 0$	$\sigma = 1$
258	1413.51	157.14
512	3701.84	218.03
1024	8989.07	274.27
2048	15791.62	338.86
4096	20159.93	415.48

The single adaptive correction (10) allows to move vector v in the right half-space in accordance to the violated inequality $\mathbf{b}^j \cdot \mathbf{v}^{j-1}$ instead of numerous fixed increments in the direction b^j, which are necessary for perceptron. Thus, we obtained the empirical proof of the significant advantage of relaxation approach in the online learning of *k*-threshold neurons. Consider results concerning the impact of optimized initial approximation. They were presented in Table 2 also in the case of the learning of 3-threshold neuron with $\eta = 2$. It is evident from Table 1 and Table 2 that the optimized initial approximation can at least halve the number of corrections. Thus, it provides the important improvement of the learning process.

Table 2
Performance comparison for random and optimized initial approximation

Dataset size	Average number of corrections	
	Random approximation	Optimized approximation
258	157.14	46.52
512	218.03	90.99
1024	274.27	133.37
2048	338.86	167.3
4096	415.48	216.71

Consider performance results in the case of different constant values of learning rate. In Table 3 the best value of the learning rate for every dataset size is shown, which was found using the grid search, as well as the average number of corrections for it. Consider learning in more general case when constant pairs (η_1, η_2) were used. Corresponding results are presented in Table 4.

Table 3
Performance results for learning with constant learning rates

Dataset size	Best learning rate	Average number of corrections
258	1.949	44.01
512	1.986	88.24
1024	2.003	128.81
2048	1.928	162.08
4096	2.051	201.33

Table 4
Best learning rate coefficients (η_1, η_2) for learning with adaptive learning rates

Dataset size	η_1	η_2	Average number of corrections
258	1.935	0.009	43.22
512	1.989	0.12	85.01
1024	2.001	0.009	122.99
2048	1.997	0.124	153.17
4096	2.038	0.103	191.05

The final experiment of the first series consists in the study of role of hyperparameter *d* on the learning. It was observed that for all datasets the best performance was obtained using $d = 0$. Moreover, in the case $d > 0.1$ the learning became considerably slower. Consider the second series of experiments. Unlike the first series performed only for $k = 3$, the second series was consisted in the learning of multi-valued *k*-threshold neuron for all $2 \le k \le 10$ using $\sigma = 1$, optimized initial approximation calculated only on the proper training set, and values of (η_1, η_2) from Table 4. The shift was not performed. Table 5 contains the average percentage of accuracy of a trained *k*-threshold neuron that was measured on the test set for every combination of the dataset size and the number of thresholds.

Table 5
Accuracy of neurons in the second series of experiments

Data-set size	Number of thresholds *k*								
	2	3	4	5	6	7	8	9	10
258	97.07	98.21	95.02	94.85	90.34	88.19	84.31	76.55	71.82
512	99.02	98.75	94.93	95.12	91.8	90.12	86.46	81.73	80.99
1024	96.85	94.7	93.08	91.02	89.94	87.67	88.12	87.04	87.26
2048	94.54	94.63	90.9	90.15	87.77	87.04	85.47	86.01	84.13
4096	92.13	90.96	85.48	86.34	85.09	86.11	82.12	81.26	79.78

After the analysis of the result of the first series of experiments, which are presented in Tables 1–4, as well as results of the second series from Table 5, one can conclude that:

1. The learning mode defined by σ is extremely significant to the performance of online learning and its proper value 1 decreases the number of corrections in 10 times.
2. The initial approximation also matters. The use of the improved approximation requires additional calculations but this can reduce the number of corrections in 2–4 times compared to random initial approximation.
3. Constant learning rate in the case $1.93 < \eta < 2.05$ is good choice for the relaxation learning.
4. The best values of the second terms in (6) were quite low compared to the first term.
5. The variation of the values of η_1 and η_2 is not so important and could improve the performance by 6–12%.
6. The generalization ability of *k*-threshold neuron decreases with the growth of *k*.
7. The shift hyperparameter *d* has mainly the theoretical importance as a guarantee of the finite learning. Its practical application is limited by small values, whereas larger *d* can significantly decrease the learning process.

6. Conclusions

The modification of the online learning algorithm for multi-valued multithreshold neurons has been considered. It uses the additional preprocessing step as well as new shift parameter *d* ensuring the convergence. Conditions has been proved for the first time that guarantee the finiteness of the learning process. The influence of the algorithm hyperparameters on the behavior of the learning algorithm has been also studied. The suggestions were stated concerning the preferred values of hyperparameters, which provided better performance during experiments on synthetic datasets. Simulation results proved the advantage of relaxation learning mode over perceptron-like one and testified that the proposed algorithm is able to greatly overperform 'incremental' procedure of Obradović and Parberry [11]. The use of optimized initial solution has also great positive impact on the performance. Despite the fact that quantitative characteristics of the improvement presented in the fifth section are not absolute and may vary depending on the dimension of feature space, the content and the size of a dataset as well as other factors, proposed recommendation could be useful for ML projects employing NNs designed using multi-valued multithreshold neurons.

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] 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.

[3] 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.

[4] 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.

[5] F. Geche, V. Kotsovsky, A. Batyuk, S. Geche, M. Vashkeba, Synthesis of time series forecasting scheme based on forecasting models system, in: CEUR Workshop Proceedings, volume 1356, 2015, pp. 121–136.

[6] 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.

[7] 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.

[8] S. Vladov, R. Yakovliev, M. Bulakh, V. Vysotska, Neural network approximation of helicopter turboshaft engine parameters for improved efficiency, Energies 17.9 (2024): 2233.
[9] S. Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson Education, Upper Saddle River, NJ, 2009.
[10] O. Kuchanskyi et al., Gender-related differences in the citation impact of scientific publications and improving the authors' productivity, Publications 11.3 (2023): 37.
[11] M. Anthony, Learning multivalued multithreshold functions, CDAM Research Report no. LSE-CDAM-2003-03, London School of Economics, 2003.
[12] B. Amirgaliyev, O. Kuchanskyi, Y. Andrashko, Building a dynamic model of profit maximization for a carsharing system accounting for the region's geographical and economic features, Eastern-European Journal of Enterprise Technologies, 2.4-116 (2022): 22–29.
[13] V. Vysotska et al., Sentiment analysis of information space as feedback of target audience for regional e-business support in Ukraine, in: CEUR Workshop Proceedings, volume 3426, 2023, pp. 488–513.
[14] 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.
[15] 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.
[16] V. Kotsovsky, Hybrid 4-layer bithreshold neural network for multiclass classification, in: CEUR Workshop Proceedings, volume 3387, 2023, pp. 212–223.
[17] R. Takiyama, Multiple threshold perceptron, Pattern Recognition 10.1 (1978): 27–30.
[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, R. J. Diephuis, A realization procedure for multithreshold threshold elements, IEEE Transactions on Electronic Computers, EC-16.6 (1967): 828-835.
[20] 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.
[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] K. Ashenayi, J. Vogh, M.R. Sayeh, B. Karimi, T. Baradaran, Multiple threshold perceptron using sinusoidal function, International Journal of Modelling and Simulation 12.1 (1992): 22–26.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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 (121) (2023): 41–47.
[28] 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.
[29] E. Baum, On the capabilities of multilayer perceptrons, Journal of Complexity 4.3 (1988): 193–215.
[30] V. Kotsovsky, Learning of multi-valued multithreshold neural units, in: CEUR Workshop Proceedings, volume 3688, 2024, pp. 39–49.