A Modified Error-Correction Learning Rule for Multilayer Neural Network with Multi-Valued Neurons

Igor Aizenberg, Senior Member, IEEE

Abstract—In this paper, we consider a modified error-correction learning rule for the multilayer neural network with multi-valued neurons (MLMVN). MLMVN is a neural network with a standard feedforward organization, but based on the multi-valued neuron (MVN). MVN is a neuron with complex-valued weights and inputs/output, which are located on the unit circle. MLMVN has a derivative-free learning algorithm based on the error-correction learning rule. The discrete \( k \)-valued MVN activation function divides a complex plane into \( k \) equal sectors. To be able to get more reliable and efficient solutions for various classification problems, it is possible to modify the MLMVN error-correction learning rule in such a way that the learning samples belonging to different classes (clusters) will be concentrated along the bisector of a desired sector (the cluster center) and at the same time will be located as far as possible from each other. Such a modification based on soft margins learning, which is reduced to the minimization of the angular distance between the bisector of a desired sector and a weighted sum, is considered in this paper.

I. INTRODUCTION

In this paper, we consider a modified error-correction learning rule for the multilayer neural network based on multi-valued neurons (MLMVN). This neural network is a valuable member of the rapidly developing complex-valued neural networks (CVNN) family. A good observation of the state of the art in the CVNN area can be found, for example, in [1], [2] where all the modern trends are deeply observed. MLMVN is a feedforward neural network, which has a traditional feedforward topology where neurons are grouped in layers, and outputs of all neurons from a current layer are connected to the corresponding inputs of all neurons from a following layer. The main distinction of MLMVN from a classical feedforward neural network with sigmoidal neurons is the multi-valued neuron (MVN), which is its basic neuron. MVN is the first historically known complex-valued neural element. It was initially called "an element of multiple-valued threshold logic", which was introduced by N.N.Aizenberg et al in 1973 in [3]. Then it was re-introduced as a discrete MVN in [4]. While MVN weights can be arbitrary complex numbers, its inputs and output are located on the unit circle. Discrete MVN outputs are the exact \( k \)th roots of unity (where \( k \) is a positive integer). The discrete MVN activation function was proposed by N.N.Aizenberg et al in 1971 in the paper [5], and it is the first historically known complex-valued activation function. The discrete MVN was considered in detail in [6].

A continuous MVN was introduced in [7]. In the same paper, it was suggested to use the MVN as a basic neuron in a multilayer neural network with multi-valued neurons (MLMVN). This network, which may consist of both continuous and discrete MVNs and its derivative-free learning algorithm, were presented in detail in [8]. The MLMVN learning algorithm was generalized in [9] for a network with an arbitrary amount of output neurons.

In [10], MVN with a periodic activation function (MVN-P) was introduced. On the one hand, a single MVN-P may learn non-linearly separable input/output mappings and on the other hand, MVN-P can be used as an output neuron in MLMVN.

A comprehensive study of MVN and MLMVN, their error-correction learning and other properties are presented in detail in [11].

Both a single MVN and MLMVN have a learning algorithm based on the same error-correction learning rule. This algorithm is derivative-free. MLMVN is one of the brightest applications of MVN. MLMVN significantly outperforms a classical feedforward network with sigmoidal neurons in terms of learning speed and generalization capability [8], [9], [11]. It performs better than SVM when solving, for example, such a challenging real-world multi-class classification problem as type of blur and blur parameters identification [9]. Recently MLMVN was successfully used for signals decoding in an EEG-based brain-computer interface where it also outperforms other techniques [12]. Apart MLMVN, we would like to distinguish MVN-based associative memories among other MVN applications: the cellular memory [4], the Hopfield-like memories [13], [14], [15], the memory for storing images (invariant to any image rotation) [16], and the memory with random connections [6], [11], [17].

Although MLMVN has shown its high efficiency in different applications, its performance when solving classification problems can be improved.

To solve a classification problem using MLMVN, a discrete MVN (or MVN-P) is used as its output neuron (neurons). A discrete MVN output is always one of the \( k \)th roots of unity. The discrete MVN activation function separates the complex plane into \( k \) equal sectors, adjusting an MVN output onto a lower border of the sector where the weighted sum has fallen. A learning process consists of fitting the weighted sums into the corresponding desired sectors. Regardless of the particular location of the weighted...
sum in the sector number \( j = 0, 1, \ldots, k - 1 \), the neuron's output equals \( \varepsilon_k^j = e^{j2\pi j/k} \) (\( i \) is an imaginary unity). This means that the MVN (and MLMVN) learning algorithm does not care of a particular location of a weighted sum inside a desired sector. As a result, a weighted sum for any learning sample can be located very close to the sector borders (determined by \( \varepsilon_k^j \) and \( \varepsilon_k^{j+1} \)) in terms of the angular distance. This may lead to a potentially problematic situation where some test samples, which are quite similar to those ones generated the weighted sums located close to the borders during the learning process, can be misplaced in one of the adjacent sectors and misclassified accordingly.

We would like to modify here the MLMVN error-correction learning rule in such a way that the number of the test errors caused by the just described problem will be minimized. Thus we would like to try to maximize the angular distance between the learning samples belonging to the distinct classes (clusters). This problem is similar to the one successfully solved in support vector machines (SVM) [18], [19] where the soft margin method, which was suggested in [19], chooses a hyperplane that separates the learning samples as cleanly as possible while maximizing the distance to the nearest cleanly split samples.

In this paper, we use the soft margin idea to modify the MLMVN learning rule and to improve the MLMVN performance. The maximization of the angular distance (in terms of arguments of complex numbers) between the learning samples belonging to the distinct classes (clusters) can be reduced to the maximization of the angular distance between the learning samples and the borders corresponding to their desired sectors. In [20], it was suggested to use the hard margin method to achieve this maximization. This method consists of the reservation of a fixed size small subsector around a bisector of a desired sector as an "allowable" area for the weighted sums, while the rest of the desired sector is a "prohibitive" area where the weighted sums are not allowed to appear during the learning process. This approach makes it possible to improve the classification results (which is shown in [20]), but it may suffer from overfitting, and the learning process takes significantly longer time.

Here we suggest to "move" the weighted sums closer to the bisectors of the corresponding sectors by minimizing the angular distance between them in terms of the root mean square error. So instead of using hard margins, we suggest to use the soft margins for the weighted sums, approaching them to the bisectors. As it will be shown below, this approach improves the classification results when MLMVN is used as a classifier.

II. MLMVN AND ITS ERROR-CORRECTION LEARNING

A. MVN

Let us recall very briefly some MVN fundamentals for the reader's convenience.

MVN is a neuron with \( n \) inputs and one output, all located on the unit circle. The discrete MVN was proposed in [4]. A discrete MVN input/output mapping is described by a function of \( n \) variables \( f(x_1, \ldots, x_n) \), which is either a function \( f : E^n_k \rightarrow E_k \) or a function \( f : O^n \rightarrow E_k \) (where \( E_k = \{1, \varepsilon_k, \varepsilon_k^2, \ldots, \varepsilon_k^{k-1}\} \) is the set of the \( k^{th} \) roots of unity, \( \varepsilon_k = e^{j2\pi/k} \) is the primitive \( k^{th} \) root of unity, \( i \) is an imaginary unity, \( k \) is a positive integer, and \( O \) is a set of points located on the unit circle). This function is a threshold function of \( k \)-valued logic [11] and therefore it can be represented using \( n+1 \) complex-valued weights as follows [11]

\[
f(x_1, \ldots, x_n) = P(w_0 + w_1 x_1 + \ldots + w_n x_n)
\]

where \( x_1, \ldots, x_n \) \((x_j \in E_k, j = 1, \ldots, n)\) are neuron inputs (the variables, which a performed function depends on) and \( w_0, w_1, \ldots, w_n \) are the weights. \( P \) is the activation function of the neuron:

\[
P(z) = \varepsilon_k^j = e^{j2\pi j/k}, \text{ if } 2\pi j/k \leq \arg z < 2\pi (j+1)/k, \quad (2)
\]

where \( \varepsilon_k^j, j=0, 1, \ldots, k-1 \) are values of the \( k \)-valued logic, \( z = w_0 + w_1 x_1 + \ldots + w_n x_n \) is the weighted sum, \( \arg z \) is the argument of the complex number \( z \).

![Fig. 1. Geometrical interpretation of the discrete MVN activation function](image)

Function (2), which was suggested in the seminal paper [5], divides a complex plane into \( k \) equal sectors and maps the whole complex plane into a set \( E_k \) of the \( k^{th} \) roots of unity (see Fig. 1). If the weighted sum is located in sector \( j \) then the neuron’s output is \( \varepsilon_k^j \).

If an MVN input/output mapping is described by a function \( f : O^n \rightarrow E_k \), then the neuron inputs are not necessarily exactly \( k^{th} \) roots of unity, but arbitrary points
located on the unit circle. It is important to mention that if there is a real-valued input/output mapping defined on the bounded subdomain \( D^n \subset \mathbb{R}^n \) and 
\[
f(y_1, \ldots, y_n): D^n \to K = \{0,1,\ldots,k-1\},
\]
where 
\[
y_j \in \left[ a_j, b_j \right], a_j, b_j \in \mathbb{R}, j = 1,\ldots,n,
\]
then it can be easily transformed to \( f: O^n \to E_k \) by a linear transformation applied to each variable (input):
\[
y_j \in \left[ a_j, b_j \right] \Rightarrow \varphi_j = \frac{y_j - a_j}{b_j - a_j} \alpha \in [0,2\pi],
\]
\( j = 1,\ldots,n; 0 < \alpha < 2\pi, \)
and then \( x_j = e^{i \varphi_j} \in O, j = 1,2,\ldots,n \) is the complex number located on the unit circle.

The MVN error-correction learning rule, which was proposed in [17], generalizes the classical Rosenblat’s error-correction learning rule [21]. This rule and the MVN learning algorithm based on it are presented and analyzed in detail in [11] where a modified proof of the convergence of the learning algorithm has also been presented.

The weights adjustment for both discrete and continuous MVN is completely determined by the neuron’s error, which is the arithmetic difference \( \delta = D - Y \) between the complex numbers \( D \) (a desired output) and \( Y \) (an actual output) located on the unit circle.

The error-correction learning rule is [11]
\[
W_{r+1} = W_r + \frac{C_r}{(n+1)} \delta \overline{X},
\]
and the rule with a self-adaptive learning rate is [11]:
\[
W_{r+1} = W_r + \frac{C_r}{(n+1)} |z_j| \delta \overline{X},
\]
where \( \overline{X} \) is the input vector with the components complex-conjugated, \( n \) is the number of neuron inputs, \( r \) is the number of the learning step, \( W_r \) is the current weighting vector (to be corrected), \( W_{r+1} \) is the following weighting vector (after correction), \( C_r \) is the constant part of the learning rate, and \( |z_j| \) is the absolute value of the weighted sum obtained on the \( r \)th learning step (self-adaptive part of the learning rate).

It should also be mentioned that to learn highly nonlinear input/output mappings using a discrete MVN, it is reasonable to calculate the error not as the difference between the desired \( (D = e^{i\varphi_j}) \) and actual \( (Y = e^{i\varphi_j}) \) outputs, but as the difference between the desired output \( D \) and the projection \( z/|z| \) of the current weighted sum \( z \) on the unit circle. Thus the error in such a case is \( \delta = D - z/|z| \).

**B. MLMVN, its Error Backpropagation and Learning**

The idea of MLMVN was first suggested in [7]. Then MLMVN and its derivative-free learning algorithm based on the error-correction learning rule with the error backpropagation were presented in detail in [8]. In [9], the MLMVN learning algorithm was generalized for an arbitrary amount of output neurons. The most detailed observation of MLMVN with mathematical proofs of its error-correction learning convergence and justification of its error backpropagation is done in [11]. A slight modification the MLMVN learning algorithm was done in [20] where the hard margin learning rule was also introduced. Since we will modify the MLMVN learning rule here, let us recall some basic points.

Let MLMVN contain one input layer, \( m-1 \) hidden layers and one output layer. Let us use the following notations. Let \( D_{jm} \) be a desired output of the \( j \)th neuron from the \( m \)th (output) layer (here and further an output layer index is \( m \)); \( Y_{jm} \) be an actual output of the \( j \)th neuron from the output layer. Then the global error of the network taken from the \( j \)th neuron of the output layer is calculated as follows:
\[
\delta_{jm} = D_{jm} - Y_{jm}; j = 1,\ldots,N_m.
\]

The MLMVN learning algorithm is derived [8], [9], [11] from the consideration that the global error of the network depends on the local errors of all the neurons and therefore it must be shared among all the neurons because all of them contribute to this error by their local errors.

The backpropagation of the global errors \( \delta_{jm}^* \) through the network is used to express the error of each neuron \( \delta_{jr}, r = 1,\ldots,m \) by means of the global errors \( \delta_{jm}^* \) of the entire network.

Let us use the following notations. Let \( w_{jr}^m \) be the weight corresponding to the \( r \)th input of the \( jr \)th neuron \( (jr \)th neuron of the \( r \)th layer), \( Y_{jr} \) be the actual output of the \( r \)th neuron from the \( r \)th layer \( (r=1,\ldots,m) \), \( \tilde{Y}_{jr} \) be the updated (after the weights are corrected) output of the \( r \)th neuron from the \( r \)th layer \( (r=1,\ldots,m) \), and \( N_r \) be the number of the neurons in the \( r \)th layer. Evidently, the neurons from the \( r+1 \)th layer have exactly \( N_r \) inputs \( (r=1,\ldots,m) \). Let \( x_1,\ldots,x_n \) be the network inputs.

The global errors of the entire network are determined by (6). The goal of the learning process is to correct the errors. To obtain the local errors for all neurons, the global error must be shared with these neurons through the backpropagation process. This process works as follows. The errors of the output layer neurons are:
\[
\delta_{jm} = \frac{1}{t_m} \delta_{jm}^*; j = 1,\ldots,N_m,
\]
where \( jm \) specifies the \( j \)th neuron of the \( m \)th (output) layer; \( t_m = N_{m-1} + 1 \), i.e. the number of all neurons in the preceding layer (layer \( m-1 \) where the error (7) will be then
backpropagated to) incremented by 1.

Then the errors of the hidden layers neurons are

$$\delta_{jr} = \frac{1}{t_r} \sum_{j=1}^{N_r} \delta_{j,r+1} (w_{jr}^{r+1})^{-1},$$  \hspace{1cm} (8)

where \(jr\) specifies the \(j^\text{th}\) neuron of the \(r^\text{th}\) layer \((j=1,\ldots,m-1)\); \(t_r = N_{r-1} + 1,\ r = 2,\ldots,m, t_n = n + 1\) \((n\) is the number of network inputs) is the number of all neurons in the layer \(r-1\) (the layer where the error from the layer \(r\) will be then backpropagated to, or in other words the number of inputs of the \(j^\text{th}\) neuron from the \(i^\text{th}\) layer, incremented by 1. Equations (7)-(8) determine the MLMVN error backpropagation.

The weights for all neurons of the network can be corrected after calculation of the errors. This correction can be done using the error-correction learning rules (5) and (6) adapted to MLMVN. These adapted rules are initially presented in [8] and slightly modified in [20]. They are as follows.

For the neurons from the \(m^\text{th}\) (output) layer \((j^\text{th}\) neuron of the \(m^\text{th}\) layer),

$$\tilde{w}_{jm} = w_{jm}^{m-1} + \delta_{jm} \tilde{Y}_{m-1}, \ i = 1,\ldots,N_{m-1},$$

$$\tilde{w}_{0m} = w_{0m}^{m-1} + \delta_{0m},$$ \hspace{1cm} (9)

for the neurons from the \(2^\text{nd}\) hidden layer till \(m-1^\text{st}\) hidden layer \((j^\text{th}\) neuron of the \(r^\text{th}\) layer \((r=2,\ldots,m-1)\),

$$\tilde{w}_{jr} = w_{jr}^{r-1} + \frac{1}{|z_{jr}|} \delta_{jr} \tilde{Y}_{r-1}, \ i = 1,\ldots,N_{r-1},$$

$$\tilde{w}_{0r} = w_{0r}^{r-1} + \frac{1}{|z_{jr}|} \delta_{jr},$$ \hspace{1cm} (10)

and for the neurons of the \(1^\text{st}\) hidden layer

$$\tilde{w}_{j1} = w_{j1}^{1} + \frac{1}{|z_{j1}|} \delta_{j1} \tilde{X}_{1}, \ i = 1,\ldots,n,$$

$$\tilde{w}_{01} = w_{01}^{1} + \frac{1}{|z_{j1}|} \delta_{j1},$$ \hspace{1cm} (11)

The MLMVN learning algorithm consists of the sequential checking for all learning samples whether an actual output of the network coincides with a desired output. If for some sample there is no coincidence, then the errors (6) must be backpropagated according to (7)-(8), and the weights must be then adjusted according to (9)-(11). The learning process continues either until the zero-error is reached or one of the mean square error (MSE) or the root mean square error (RMSE) criteria is satisfied.

We consider here only MLMVN with the discrete output neurons. This means that both MSE and RMSE should be applied to the errors in terms of the numbers of sectors (see Fig. 1), thus not to the elements of the set \(E_k = \{\epsilon_k^0, \epsilon_k^1, \ldots, \epsilon_k^{k-1}\}\), but to the elements of the set \(K = \{0,1,\ldots,K-1\}\). The local errors for the \(r^\text{th}\) learning sample in these terms are calculated as

$$\gamma_r = (\alpha_{j_r} - \alpha_r) \mod k; \alpha_r \in \{0,1,\ldots,k-1\}$$  \hspace{1cm} (12)

\((\epsilon_k^0\) is the desired output and \(\epsilon_k^{\gamma_r}\) is the actual output).

Let \(\Delta_r\) be the square error of the network for the \(r^\text{th}\) learning sample. For MLMVN with a single output neuron it is

$$\Delta_r = \gamma_r^2, r = 1,\ldots,N$$ \hspace{1cm} (13)

\((\gamma_r = \text{the local error taken from (12)}\) and, for MLMVN with \(N_m\) output neurons \((m\) is the output layer index\) it is

$$\Delta_r = \frac{1}{N_m} \sum_{j=1}^{N_m} (\gamma_{jr})^2; r = 1,\ldots,N.$$ \hspace{1cm} (14)

The MLMVN learning process in such a case continues until RMSE drops below some pre-determined acceptable minimal value \(\bar{\lambda}\):

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N} \sum_{r=1}^{N} \Delta_r} \leq \bar{\lambda}.$$ \hspace{1cm} (15)

The convergence of this learning algorithm is proven in [11].

III. SOFT MARGINS LEARNING FOR MLMVN

A. Why soft margins?

Whenever MLMVN output neurons are discrete, an MLMVN output is one of the \(k^\text{th}\) roots of unity. The learning process consists of fitting the weighted sums for all samples from a learning set into the corresponding desired sectors. If \(\epsilon_k^q\) is a desired MLMVN output and the weighted sum \(z\) has fallen in the desired sector number \(q\), then regardless of the particular location of the weighted sum in the sector \(q\), \(\arg \epsilon_k^q \leq \arg z < \arg \epsilon_k^{q+1}\). Since, the MLMVN learning algorithm does not care of a particular location of a weighted sum inside a desired sector, then a weighted sum for any learning sample may fall very close to the sector borders (determined by \(\epsilon_k^q\) and \(\epsilon_k^{q+1}\)) in terms of the angular distance. This may lead to such a situation where some test samples, which are quite similar to those ones generated the training set, may fall very close to the sector borders during the learning process, can be misplaced in one of the adjacent sectors and misclassified accordingly. This situation is illustrated in Fig. 2.

Suppose, during the learning process the weighted sum \(z_t\) corresponding to the \(t^\text{th}\) learning sample was fitted too close to the desired sector \(q\) lower border determined by \(\epsilon_k^q\). Suppose that our test set contains a test sample (the \(t^\text{th}\) one), which is quite similar to the \(t^\text{th}\) learning sample and belongs to the same class \(q\). This similarity usually means that the weighted sum \(z_t\) corresponding to the \(t^\text{th}\) test sample falls close to the weighted sum \(z_t\) (in terms of the angular distance) corresponding to the \(t^\text{th}\) learning sample. A
problem is that if \( Z_i \) is located in the proper sector, but too close to this sector border, then \( Z_i \) may fall close to \( Z_l \), but unfortunately from the opposite side of the sector border, that is in the adjacent sector. As a result of that the membership of the \( l^{th} \) test sample will be misidentified.

![Fig. 2](image-url)  
**Fig. 2.** Potentially non-reliable results of the learning process. The weighted sum \( Z_l \) corresponding to the learning sample \( l \) was fitted too close to the desired sector \( q \) lower border determined by \( E_k^q \). As a result, the weighted sum \( Z_t \) corresponding to the test sample \( t \) belonging to the same class \( q \) has fallen in the adjacent sector \( q-1 \).

Let us take a closer look at Fig. 2. Our desired sector is the sector number \( q \) whose borders are determined by \( E_k^q \) and \( E_k^{q+1} \). The weighted sum \( Z_l \) corresponding to the \( l^{th} \) learning sample is located in the desired sector number \( q \), but too close to its lower border determined by \( E_k^q \). As a result, the test weighted sum \( Z_t \) corresponding to the \( t^{th} \) test sample falls in the adjacent sector number \( q-1 \) whose borders are determined by \( E_k^{q-1} \) and \( E_k^q \). Let us take care of this problem.

Our goal is to modify the MLMVN learning algorithm in such a way that the number of the test (classification) errors caused by the just described problem will be minimized. Since our problem is the closeness of a weighted sum to the one of a sector borders in terms of the angular distance, then this problem can be solved by moving weighted sums for all learning samples to the middle of the corresponding sector that is, every weighted sum must be moved closer to the bisection of a desired sector. The bisection of the sector \( q \) is determined by the complex number \( E_{2k}^{2q+1} = e^{i(2\pi(2q+1)/2k)} \) (evidently, \( \arg E_{2k}^{2q+1} = (\arg E_k^q + \arg E_k^{q+1}) \mod 2\pi / 2 \)).

Thus during the learning process we have to minimize the angular distance \( \phi = |\arg E_{2k}^{2q+1} - \arg z_l| \) (see Fig. 2), to ensure that every weighted sum is located as close as possible to the bisection of a corresponding desired sector for every learning sample. If this goal will be achieved, we should expect that instead of the situation presented in Fig. 2 and just described we will have another situation, which is presented in Fig. 3. Here the learning weighted sum \( Z_l \) is located in the desired sector \( q \) close not to its border, but to its bisector determined by \( E_{2k}^{2q+1} \) (so the angular distance \( \phi \) is minimized). As a result, the test weighted sum \( Z_t \) falls in the same desired sector \( q \) and the membership of the \( t^{th} \) test sample is identified correctly.

![Fig. 3](image-url)  
**Fig. 3.** Reliable results of the learning process. The weighted sum \( Z_l \) corresponding to the learning sample \( l \) was fitted close to the bisection of the desired sector \( q \). As a result, the weighted sum \( Z_t \) corresponding to the test sample \( t \) belonging to the same class \( q \) has also fallen in the desired sector \( q \).

### B. Soft Margins Criterion

The hard margin MVN/MLMVN learning suggested in [20] requires that for every learning sample a corresponding weighted sum must be strictly located in a pre-determined narrow subsector of a desired sector. This means that the learning process considered converged if for every learning sample and for every MLMVN output neuron the following condition holds along with (15)

\[
\phi_r = (\arg z_r - \arg E_{2k}^{2q+1}) \leq \theta; r = 1, \ldots, N.
\]  
(16)

Here \( r = 1, \ldots, N \) is the index of a learning sample, \( q \) is the number of the desired sector, \( \arg z_r \) is the argument of the weighted sum \( z_r \), \( \arg E_{2k}^{2q+1} \) is the argument of the root of unity corresponding to the bisection of the desired sector, \( \theta \) is a half of the angular size of the narrow subsector allowable for fitting there the weighted sums (hence \( 2\theta \) is the angular size of this narrow sector).

In fact, condition (16) establishes hard margins of the angular size of \( (2\pi / k - 2\theta) / 2 \) for all the weighted sums.
from both sides of a designated narrow subsector \((2\pi/k)\) is the angular size of one sector determined by activation function (2), see Fig. 1.

Let us consider another situation where some weighted sums are allowed to appear out of the \(\theta\)-interval \(\arg e_{2k}^{2q+1} - \theta, \arg e_{2k}^{2q+1} + \theta\) around the bisector of a desired sector, while a majority of the weighted sums are located within this \(\theta\)-interval. This softer requirement is nothing else than a soft margin requirement. It can be expressed in terms of the root mean square error (RMSE).

Let \(N_m\) be the number of output neurons in MLMVN, \(\Phi_{jr}\) is the angular distance between the weighted sum and the bisector of the desired sector for the \(r\)th learning sample and \(j\)th output neuron \((r = 1, ..., N_j, j = 1, ..., N_m)\). Then, averaging \(\Phi_{jr}; j = 1, ..., N_m\) over the number of output neurons for every learning sample, we obtain the local angular squared errors for all the learning samples

\[
\Phi_r = \frac{1}{N_m} \sum_{j=1}^{N_m} (\Phi_{jr})^2; \ r = 1, ..., N_j.
\]

(17)

Then the angular RMSE for the entire learning set can now easily be obtained from (17) and we obtain the following soft margins criterion for MLMVN

\[
\Phi = \sqrt{\frac{1}{N} \sum_{r=1}^{N} \Phi_r} \leq \theta.
\]

(18)

where \(\Phi_r\) is the angular error for the \(r\)th learning sample, which is determined by (17).

Thus according to (18) the root mean square angular error in terms of the angular distance from a weighted sum to the bisector of a corresponding desired sector shall not exceed the pre-determined threshold \(\theta\). The latter must be chosen such that \(\theta < 2\pi/2k\), which means that \(\theta\) shall not exceed a half-size of one sector determined by activation function (2) (see Fig. 1).

**C. Modified Learning Algorithm**

To employ a new MLMVN learning rule with the soft margins criterion, which was just presented, we have to introduce two modifications of the MLMVN learning algorithm [8], [9], [11] briefly recalled above.

The first modification is reduced to a different method of the MLMVN error (errors) calculation. If (18) is used as a criterion of the learning quality, this means that the quality is evaluated in terms of the angular distance between a weighted sum and the bisector of a desired sector. Thus the bisector becomes a target of our learning process.

The MLMVN errors are calculated according to (6). Since we consider here only MLMVN with discrete \(k\)-valued outputs, both \(D\) and \(Y\) are the \(k\)th roots of unity. Let \(D = e_k^q\) and \(Y = e_k^s\). Thus the sector number \(q\) is our desired (targeting) sector, while the sector number \(s\) is that actual sector where a current weighted sum is located. At the same time, our idea is not only to move the weighted sum \(z\) from the incorrect sector \(s\) to the correct sector \(q\), but also to move \(z\) to the \(\theta\)-interval around the bisector of the sector \(q\), which is determined by \(e_k^{2q+1}\). Hence it is reasonable to calculate the error as the difference \(\delta = e_k^{2q+1} - z/|z|\) where \(e_k^{2q+1}\) is a root of unity corresponding to the bisector of the desired sector \(q\) and \(z/|z|\) is a projection of the weighted sum on the unit circle.

Thus to calculate the global MLMVN error \(\delta_{jm}\) taken from the \(j\)th neuron of the output layer according to that modified rule, which we have just considered, we have to set

\[
D_{jm} = e_k^{2q+1}, \hat{Y}_{jm} = \frac{z_{jm}}{|z_{jm}|}
\]

and then

\[
\delta_{jm} = D_{jm} - \hat{Y}_{jm}; j = 1, ..., N_m.
\]

(19)

where \(m\) is the output layer index. The backpropagation of these modified global network errors is still determined by (7)-(8) without any changes. The network weights shall still be adjusted according to (9)-(11).

The second modification of the MLMVN learning algorithm is a modification of the stopping and correcting criteria. The regular MLMVN learning algorithm continues until the RMSE criterion (15) has been satisfied. To implement the soft margins MLMVN learning, (15) must be used in conjunction with (18), which is the soft margins criterion.

Also, in the regular MLMVN learning algorithm, the weights must be adjusted only if at least one of the errors (6) is non-zero. However, even if all these errors equal 0, which means that the weighted sums for all learning samples and all output neurons are fitted in the desired sectors, any of these weighted sums can still be located out of the \(\theta\)-interval around a corresponding bisector. Hence we have to modify a correcting criterion. The MLMVN weights must be adjusted if the following disjunction of two conditions is true for any one of the output neurons

\[
(D_{jm} - Y_{jm} \neq 0) \vee \arg z_{jm} - \arg e_{2k}^{2q+1} \leq \hat{\theta},
\]

(20)

where \(m\) is the index of the output layer, \(j\) is the index of neuron in the output layer; \(D_{jm}\) is a desired output of the \(jm\)th neuron; \(Y_{jm}\) be an actual output of the \(jm\)th neuron; \(z_{jm}\) is a weighted sum of the \(jm\)th neuron; \(q_{jm}\) is a desired sector of the \(jm\)th neuron; \(e_{2k}^{2q+1}\) determines the bisector of the sector \(q_{jm}\); \(\hat{\theta}\) is a tolerance parameter, which determines an allowable deviation of the weighted sum from the bisector of a desired sector. Evidently, the simplest choice for \(\hat{\theta}\) is \(\hat{\theta} = \theta\).

The convergence of the modified MLMVN learning algorithm can be proven in the same way as the convergence of MLMVN learning algorithm for MLMVN with continuous output neurons was proven in [11]. It is also
important to mention that the same approach with some adaptations presented in [10], [11] can be used for MLMVN with MVN-P as an output neuron (neurons).

IV. SIMULATION RESULTS
In this section, we consider simulations where the approach presented above is tested using some popular benchmarks. We used the MLMVN software simulator where the soft margins learning is utilized, written in Borland Delphi 5.0 environment, running on a PC with the Intel® Core™i7 CPU. Since MLMVN is more suitable (and interesting) for solving multi-class classification problems, exactly such problems are considered.

A. Glass Identification
This dataset was downloaded from the UC Irvine Machine Learning Repository [22]. The data set contains 6 classes, which are represented by 214 instances. Taking into account many controversial classification results reported for this data set (these controversies are caused by very different amounts of samples used for the learning and training purposes by different authors) and a high imbalance factor of this data set, we decided to perform our experimental testing using leave-one-out cross-validation. MLMVN with soft margins learning presented in this paper (MLMVN-SM) was compared to regular MLMVN and SVM with the Gaussian and second order polynomial kernels. Comparison with regular MLMVN is reasonable, since we want to improve the MLMVN performance. SVM was chosen as a competitive technique because it is commonly recognized as one of the most efficient machine learning tools.

For our experiments with SVM, we used the software simulator mcSVM v1.0.2, which was designed for solving multi-class classification problems and employs the "one against all" model.

In our experiments with MLMVN-SM and MLMVN we transformed the real-valued input features into the numbers located on the unit circle using (3) with \( \alpha = \frac{7\pi}{8} \).

Both MLMVN-SM and MLMVN, which we used in these experiments, had the same topology \( 9 \rightarrow 36 \rightarrow 1 \) (9 inputs, 36 neurons in a single hidden layer and a single output neuron). The continuous activation function was used in all hidden neurons. Our single output neuron employs discrete activation function (2) with \( k = 6 \). The choice of 36 hidden neurons was based only on the results of experimental testing. We used \( \theta = \bar{\theta} = 0.32 \) in (18) and (20), respectively. This choice was based on the following idea. The angular size of one sector determined by activation function (2) with \( k = 6 \) is \( 2\pi / 6 \). A subsector whose angular size is a "golden" part of the sector (that is, \( (2\pi / 6) / 1.618 \approx 0.64 \) ) was chosen as a desirable area for the corresponding weighted sums. This means that \( 2\theta = 0.64 \) and \( \theta = 0.32 \).

The experimental results are presented in Table I.

<table>
<thead>
<tr>
<th>Tool</th>
<th># of correctly classified samples (out of 214)</th>
<th>Classification rate</th>
<th>Learning iterations (average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLMVN-SM</td>
<td>173</td>
<td>80.84%</td>
<td>36107</td>
</tr>
<tr>
<td>MLMVN</td>
<td>165</td>
<td>77.10%</td>
<td>3600</td>
</tr>
<tr>
<td>SVM (Gaussian)</td>
<td>132</td>
<td>61.68%</td>
<td>83</td>
</tr>
<tr>
<td>SVM (Poly 2)</td>
<td>137</td>
<td>64.02%</td>
<td>9</td>
</tr>
</tbody>
</table>

The test results show that MLMVN-SM outperforms MLMVN, showing a higher classification rate. In turn, even regular MLMVN significantly outperforms SVM with both Gaussian and polynomial kernels. It is important to mention that thousands learning iterations for MLMVN and the tens of thousands learning iterations for MLMVN-SM should not scare the user and should not be considered as a disadvantage. All the computations are performed very quickly. The average running time of a learning iteration for MLMVN is 0.0065 sec and the average running time of a learning iteration for MLMVN-SM is 0.0083 sec. This means that 3600 MLMVN learning iterations take 23.4 sec and 36107 learning iterations of MLMVN-SM take 299.7 sec = 5 minutes.

B. Car Evaluation
This dataset was also downloaded from the UC Irvine Machine Learning Repository [22]. The data set contains 4 classes, which are represented by 1728 instances. Many controversial classification results reported for this data set either. These controversies as well as for the previous data set are caused by very different amounts of samples used for the learning and training purposes by different authors and by a high imbalance factor of the data set. We have performed our experimental testing using 5-fold cross-validation. MLMVN with soft margins learning presented in this paper (MLMVN-SM) was compared to regular MLMVN and SVM with the Gaussian and second order polynomial kernels. The same software simulators as for the previous data set were used.

We encoded categorical features of this data set using integers 0,...,k-1 (k is the number of the features), which in our experiments with MLMVN-SM and MLMVN were then transformed into the numbers located on the unit circle using (3) with \( \alpha = \pi \).

Both MLMVN-SM and MLMVN, which we used in these experiments, had the same topology \( 9 \rightarrow 36 \rightarrow 1 \) (9 inputs, 36 neurons in a single hidden layer and a single output neuron). The continuous activation function was used in all hidden neurons. Our single output neuron employs discrete
activation function \((2)\) with \(k=6\). The choice of 36 hidden neurons was based only on the results of experimental testing. We used \(\theta = \hat{\theta} = 0.6\) in (18) and (20), respectively.

The experimental results are presented in Table 2.

<table>
<thead>
<tr>
<th>Tool</th>
<th># of correctly classified samples (out of 346)</th>
<th>Classification rate</th>
<th>Learning iterations (average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLMVN-SM</td>
<td>299</td>
<td>86.41%</td>
<td>677</td>
</tr>
<tr>
<td>MLMVN</td>
<td>292</td>
<td>84.32%</td>
<td>336</td>
</tr>
<tr>
<td>SVM (Gaussian)</td>
<td>300</td>
<td>86.98%</td>
<td>112</td>
</tr>
<tr>
<td>SVM (Poly 2)</td>
<td>250</td>
<td>72.3%</td>
<td>15</td>
</tr>
</tbody>
</table>

The test results show that MLMVN-SM performs as good as SVM with the Gaussian kernel (their classification results are almost identical) and outperforms MLMVN and SVM with the polynomial kernel.

It should be mentioned that the same data sets are tested in [23] for another complex-valued neural network – FLCNC, which showed 81.9% (Glass Identification) and 81.3% (Car Evaluation) classification rates, employing 80 and 90 hidden neurons, respectively. However, the cardinalities of the learning and testing sets were not exactly specified for these experiments.

V. CONCLUSIONS

In this paper, we have considered a modified learning rule for the multilayer neural network with multi-valued neurons (MLMVN). The soft margins were introduced for the weighted sums of the output MLMVN neurons, the error calculation was modified, and both correcting and stopping criteria were modified accordingly. These innovations brought us to the concept of MLMVN-SM and make it possible to improve the MLMVN generalization capability. The key idea behind MLMVN-SM is maximization of the distance between representatives of the distinct classes by creation the soft margins in the areas of borders between the classes.

REFERENCES


