Abstract—In this paper, we consider a new periodic activation function for the multivalued neuron (MVN). The MVN is a neuron with complex-valued weights and inputs/output, which are located on the unit circle. Although the MVN outperforms many other neurons and MVN-based neural networks have shown their high potential, the MVN still has a limited capability of learning highly nonlinear functions. A periodic activation function, which is introduced in this paper, makes it possible to learn nonlinearly separable problems and non-threshold multiple-valued functions using a single multivalued neuron. We call this neuron a multivalued neuron with a periodic activation function (MVN-P). The MVN-Ps functionality is much higher than that of the regular MVN. The MVN-P is more efficient in solving various classification problems. A learning algorithm based on the error-correction rule for the MVN-P is also presented. It is shown that a single MVN-P can easily learn and solve those benchmark classification problems that were considered unsolvable using a single neuron. It is also shown that a universal binary neuron, which can learn nonlinearly separable Boolean functions, and a regular MVN are particular cases of the MVN-P.

Index Terms—Classification, complex-valued neural networks, derivative-free learning, mod-k addition, multivalued neuron.

I. INTRODUCTION

The discrete multi-valued neuron (MVN) was introduced in [1]. This neuron operates with complex-valued weights. Its inputs and output are located on the unit circle. Its outputs are exactly k-th roots of unity (where k is a positive integer). The MVNs activation function was proposed in the seminal paper [2] as a key tool of the theory of multiple-valued (k-valued) threshold functions over the field of complex numbers. This theory was introduced in [3] and then developed and elaborated in [4]. The most important element of this theory is that the values of k-valued logic are encoded by the k-th roots of unity. The MVNs activation function depends only on the argument (phase) of the weighted sum and maps the entire complex plane on the unit circle. The discrete MVN may learn multiple-valued threshold functions.

A comprehensive study of the discrete-valued MVN, its properties, and learning is presented in [4]. The discrete MVN has a learning algorithm based on the error-correction rule. It is derivative-free, which makes it highly efficient. This property and the MVNs high functionality made this neuron attractive for the development of different applications. We have to mention among others several associative memories with a different topology, the cellular memory [1], the Hopfield-like memories [5]–[7], the memory for storing medical images [8], the memory with random connections [4], and a cellular network [9].

A continuous MVN was introduced in [10]. In the same paper, it was suggested to use the MVN as a basic neuron in a multilayer neural network with multivalued neurons (MLMVNs). This network, which can consist of both continuous and discrete MVNs, and its derivative-free backpropagation learning algorithm were explicitly presented in [11]. Its learning algorithm was generalized in [12] for a network with an arbitrary amount of output neurons.

Although the MVN has shown its high efficiency in different applications, there is still a very interesting open problem. Suppose we have to solve some n-dimensional k-class classification problem and the corresponding classes are nonlinearly separable. The commonly used approach for solving such a problem is its consideration in the larger dimensional space. Actually, one of the ways to utilize this approach is a neural network, where hidden neurons form a new space, and a problem becomes linearly separable [13], [14]. An even more popular (and often more efficient) way is the support vector machine (SVM) introduced in [15], where a larger dimensional space is formed using the support vectors and a problem becomes linearly separable in this new space. Let us approach the same problem from a different angle, i.e., to consider an n-dimensional k-class classification problem as an n-dimensional m-class classification problem (where m > k and each of k initial classes is a union of some of t disjoint subclasses (clusters) of an initial class)

\[ C_j = \bigcup_{i=1}^{t_j} \tilde{C}_i^j, \quad j = 1, \ldots, k; 1 \leq t_j < m; \tilde{C}_i^j \cap \tilde{C}_i^s = \emptyset, t \neq s \]

where \( C_j, j = 1, \ldots, k \) is an initial class and each \( \tilde{C}_i^j, i = 1, \ldots, m \) is a new subclass. Thus, we would like to modify the formation of a decision rule instead of increasing the dimensionality. In terms of neurons and neural networks, this means increasing the functionality of a single neuron by modification of its activation function.

In this paper, we consider a new discrete activation function and a modified learning algorithm for the discrete MVN. As mentioned above, the discrete MVN can learn the k-valued threshold functions (the threshold functions of k-valued logic) [3], [4]. However, it is clear that the k-valued
threshold functions form just a small subset of the entire set of $k$-valued functions. Those functions that are not threshold cannot be learned using a single MVN. The question is, if some $k$-valued function $f$ is not a $k$-valued threshold function, can it be a partially defined $m$-valued threshold function for some $m > k$? If so, it is possible to learn this function using a single MVN, but with an $m$-valued activation function instead of a $k$-valued activation function.

In what follows, we will show one of the possible ways of finding such $m > k$ that a $k$-valued function, which is not a $k$-valued threshold function, becomes a partially defined $m$-valued threshold function.

A new activation function, which we will consider here, is a $k$-periodic function (a periodic function with a period $k$). The idea behind our approach is similar to the idea, on which the universal binary neuron (UBN) is based. The UBN was introduced in [16] and then developed in [4]. The mathematical idea behind the UBN is similar to the one behind the MVN. The UBN is a neuron with complex-valued weights and an activation function, which separates the complex plane into $m$ equal sectors determining the output by the alternating sequence of $1, -1, 1, -1, \ldots$ depending on the parity of the sector's number. When $m = 2$, the functionality of the UBN coincides with the functionality of a neuron with a threshold activation function [4]. However, if $m > 2$, the functionality of the UBN is always higher than that of a classical threshold neuron [4]. Thus, when $m > 2$, the UBN can learn nonlinearily separable Boolean functions. The UBN activation function leads to an $l$-multiple duplication of the sequence $\{1, -1\}$ and of the sectors into which the complex plane is divided, respectively. Hence $m = 2l$ is the total number of sectors in the UBN activation function. If $l > 2$, then a single UBN may learn nonlinearily separable Boolean functions.

In this paper, we suggest the use of a similar approach to increase a MVN's functionality. We suggest considering an input/output mapping described by some non threshold function $f(x_1, \ldots, x_n)$ of $k$-valued logic in $m$-valued logic, where $m = kl$. By analogy with the UBN, the complex plane will be divided into $m = kl$ sectors and the MVN's activation function in this case becomes $k$-periodic and $l$-multiple. This will lead us to the multivalued neuron with a periodic activation function (MVN-P).

We will also consider a learning algorithm for the MVN-P, which employs specific properties of the new activation function. This learning algorithm is based on the same linear error-correction learning rule as the standard error-correction MVN learning algorithm [4]. Its utilization is based on two original strategies, which we will present here.

Finally, we will show how a single MVN with a periodic activation function and a modified learning algorithm can learn and solve a number of benchmark problems, which were considered unsolvable using a single neuron.

Preliminary results [17], [18] obtained using the approach presented in this paper show that the modification of the MVN activation function and learning algorithm results in significant improvement of its functionality.

It is important to mention that this paper contributes to the further development of a complex-valued neural networks paradigm. Complex-valued neural networks have become increasingly popular. There are different specific types of complex-valued neurons and complex-valued activation functions. A good observation can be found, for example, in [19] and [20]. It is also reasonable to mention other recent works where some important aspects of a complex-valued neural networks paradigm were developed [21]–[23].

This paper has the following structure. In Section II, MVN and UBN will be described for the reader's convenience. In Section III, a $k$-periodic activation function for the MVN will be presented and the MVN-P will be defined. A modified learning algorithm for the MVN, which is in fact a learning algorithm for the MVN-P, will be described in Section IV. Simulation results will be presented in Section V, where we will show how a number of benchmark problems can be solved using a single MVN-P and its learning algorithm. Concluding remarks will be given and directions for future research will be discussed in Section VI. A proof of the convergence of the learning algorithm will be given in the Appendix.

II. MVN AND UBN

A. MVN

The discrete MVN was proposed in [1]. This neuron utilizes the principles of multiple-valued threshold logic over the field of complex numbers [3], [4]. Let us recall the following.

Let $E_k = \{1, e_k, e_k^2, \ldots, e_k^{k-1}\}$, where $e_k = e^{2\pi i/k}$ is the primitive $k$th root of unity ($i$ is an imaginary unity and $k$ is some positive integer). Let $K = \{0, 1, \ldots, k - 1\}$ be the set of the values of $k$-valued logic. A one-to-one correspondence between $K$ and $E_k$ can be easily established. Let $O$ be the continuous set of the points located on the unit circle. Let either $X = E_k$ or $X = O$.

Definition 1 [3], [4]: A function $f(x_1, \ldots, x_n) : X^n \to E_k$ is called a function of $k$-valued logic over the field of complex numbers (or simply $k$-valued function).

The range of $f$ is discrete, while its domain can be either discrete or continuous. In general, its domain may be even hybrid. It should be mentioned that if some function is defined on the bounded subdomain $D^n \subset \mathbb{R}^n$, then it can be easily transformed to $f : O^n \to K$ by a linear transformation applied to each variable

$$y_j \in [a_j, b_j], \quad y_j = \frac{y_j - a_j}{b_j - a_j}, \quad a \in [0, 2\pi],$$

$$j = 1, \ldots, n; 0 < a < 2\pi$$

and then $x_j = e^{i\varphi_j} \in O$, $j = 1, 2, \ldots, n$ is the complex number located on the unit circle. Since, there exists a one-to-one correspondence between $K$ and $E_k$, then we obtain a function $f(x_1, \ldots, x_n) : O^n \to E_k$.

1The interval $[0, 2\pi]$ in (1) is open from the right side. This is important to avoid a collision, which follows from the fact that arguments $0$ and $2\pi$ determine the same point on the unit circle. Since $0 < \varphi < 2\pi$ in (1) this guarantees that $0 \leq \varphi_j < 2\pi$, $j = 1, \ldots, n$, and a point on the unit circle corresponding to the maximal value of $y_j$, $j = 1, \ldots, n$ does not coincide with a point corresponding to its minimal value.
\[ P(z) = e^{i \frac{2\pi j}{k}}, \text{ if } \frac{2\pi j}{k} \leq \arg z < \frac{2\pi (j+1)}{k} \]  

(3)

where \( j = 0, 1, \ldots, k-1 \) are values of \( k \)-valued logic, \( i \) is an imaginary unity, and \( \arg z \) is the argument of the complex number \( z \). Vector \( (w_0, w_1, \ldots, w_n) \) is called a weighting vector of the threshold function \( f \).

**Definition 3** [3, 4]: A function \( f(x_1, \ldots, x_n) : G^n \to E_k \), where \( G^n \subset \mathbb{X}^n \) is called a partially defined function of \( k \)-valued logic.

The discrete MVN [1] is a neuron with activation function (3). It performs a mapping between \( n \) inputs and a single output according to (2). Its inputs and output are located on the unit circle. The discrete MVN’s outputs are always the \( k \)th roots of unity \( e^{i} = e^{i2\pi j/k}, j \in \{0, 1, \ldots, k-1\} \), \( i \) is an imaginary unity. It follows from Definition 2 that an MVN input/output mapping is always described by some \( k \)-valued threshold function of \( n \) variables \( f(x_1, \ldots, x_n) \). Thus, if \( z = w_0 + w_1 x_1 + \cdots + w_n x_n \) is the weighted sum of the MVN inputs, \( P(z) \) is the MVN’s output. This means that any \( k \)-valued threshold function can be implemented (learned) by a single MVN, while a non-threshold function cannot [4].

The activation function (3) divides the complex plane into \( k \) equal sectors (see Fig. 1) and maps the whole complex plane into a set of the \( k \)th roots of unity. If the weighted sum is located in sector \( j \), then the neuron’s output is \( e^{i} \). Hence, the MVNs output is determined by the argument (phase) of the weighted sum and does not depend on its magnitude.

The MVN error-correction learning is reduced to the movement along the unit circle. It is derivative-free [4]. The weights adjustment is completely determined by the neuron’s error, which is the difference between the desired and actual outputs.

The error-correction learning rule is [4]

\[ W_{r+1} = W_r + \frac{C_r}{(n+1)} (e^{q} - e^{i}) X \]  

(4)

and with the modification suggested in [11]

\[ W_{r+1} = W_r + \frac{C_r}{(n+1)|z_r|} (e^{q} - e^{i}) \bar{X} \]  

(5)

where \( \bar{X} \) is the input vector with the components complex-conjugated, \( n \) is the number of neuron inputs, \( e^{q} \) is the desired output of the neuron, \( e^{i} = P(z) \) is the actual output of the neuron (see Fig. 2), \( r \) is the number of the learning iteration, \( W_r \) 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 (it may always be equal to 1), and \( |z_r| \) is the absolute value of the weighted sum obtained on the \( r \)th iteration. A factor \( 1/|z_r| \) in (5) is a variable part of the learning rate. The use of it can be important for learning highly nonlinear input/output mappings with a number of high irregular jumps. The error-correction rule (4) and its modification (5) ensure such a correction of the weights that the weighted sum moves from sector \( s \) to sector \( q \) (see Fig. 3). The direction of this movement is determined by the error \( \delta = e^{q} - e^{i} \). The convergence of the learning algorithm, which is based on the rule (4), is proven in [4]. It should also be mentioned that to learn highly nonlinear functions, it might be reasonable to calculate the error not as the difference between the desired and actual outputs, but as the difference between the desired output and the projection of the current weighted sum on the unit circle \( \delta = e^{i} - z/|z| \). This leads to the following modification of the learning rules (4) and (5), respectively:

\[ W_{r+1} = W_r + \frac{C_r}{(n+1)} (e^{q} - \frac{z_r}{|z_r|}) \bar{X} \]  

(6)

\[ W_{r+1} = W_r + \frac{C_r}{(n+1)|z_r|} (e^{q} - \frac{z_r}{|z_r|}) \bar{X}. \]  

(7)

The activation function (3) is discrete. As proposed in [10] and [11], (3) can be modified in order to generalize it for the continuous case in the following way. If \( k \to \infty \) in (3), then
the angular size of the sector (see Fig. 1) approaches zero. Hence, (3) can be modified as follows:

\[ P(z) = \exp(i \arg z) = e^{i \arg z} = \frac{z}{|z|} \]  

(8)

where \( \arg z \) is the main value of the argument of the complex number \( z \) and \( |z| \) is its absolute value. The activation function (8) determines a continuous-valued MVN.

**B. UBN**

The UBN was introduced in [16] and then developed and considered in detail in [4]. In [24], a new learning algorithm was proposed for the UBN.

A key idea behind the UBN is the use of complex-valued weights and an original activation function for learning non-linearly separable Boolean functions. A classical threshold activation function (sgn) separates a real domain into two parts

\[ \text{sgn}(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0. \end{cases} \]  

(9)

If \( k = 2 \) in (3), then the activation function (3) separates the complex domain into two parts as well (the complex plane is separated into the top semiplane (“1”) and the bottom semiplane (“−1”))

\[ P(z) = \begin{cases} 1, & 0 \leq \arg(z) < \pi \\ -1, & \pi \leq \arg(z) < 2\pi. \end{cases} \]  

(10)

However, the activation function (9) does not increase the neuron’s functionality, although the weights are complex, the neuron can still learn only linearly separable functions [4]. In [16], the following \( l \)-multiple activation function was suggested:

\[ P_B(z) = (-1)^l, \quad \text{if} \quad \frac{2\pi j}{m} \leq \arg(z) < \frac{2\pi(j + 1)}{m}, \quad m = 2l, l \in \mathbb{N} \]  

(10)

where \( l \) is some positive integer, and \( j \) is a non-negative integer \( 0 \leq j \leq m \).

The activation function (10) is illustrated in Fig. 3. Function (10) separates the complex plane into \( m = 2l \) equal sectors. It determines the neuron’s output by the alternating periodic sequence of \( 1, -1, 1, -1, \ldots \) depending on the parity of the sector’s number. The activation function (10) equals to 1 for the complex numbers located in the even sectors \( 0, 2, 4, \ldots, m-2 \) and to −1 for the numbers located in the odd sectors \( 1, 3, 5, \ldots, m-1 \). Similarly to function (3), function (10) also depends on the argument (phase) of the weighted sum and does not depend on its magnitude.

The UBN learning algorithm is based on the same error-correction learning rule (4) as the MVN learning algorithm. Initially, this UBN learning algorithm was proposed in [4] and then it was modified and improved in [24], where the rule (5) was also used. The choice of the desired sector \( q \) in (4) and (5) is based on the closeness of the current weighted sum to the right or left adjacent sector. Indeed, if the UBN current output is incorrect, this means that it should become corrected if the weighted sum is moved to either left or right adjacent sector. This follows from the construction of the activation function (10) (see also Fig. 3). The adjacent sector, which is closer to the current value of the weighted sum in terms of angular distance, is chosen as the “correct” one. The number \( q \) of this sector determines the desired output in (4)–(7). The convergence of this learning algorithm is proven in [4].

Comparing (3) and (10) (see also Figs. 1 and 3, respectively), we may easily conclude that these activation functions are quite similar. Both are functions of the argument of the weighted sum and separate the entire complex plane on the equal sectors. It is shown in [24] that, if some Boolean function may be learned by a single UBN with the activation function (10), then there exists a partially defined (on the Boolean domain) \( m \)-valued threshold function which has the same weighting vector as the initial Boolean function. Actually, this fact determines the ability to use the learning algorithm based on either of the rules (4)–(7) for the UBN.

It is important to mention that the activation function (10) may be considered as an \( l \)-multiple extension of the activation function (9), which in turn is the MVN activation function (3) for \( k = 2 \). Indeed, (9) divides the entire complex plane into two equal sectors and (10) divides the entire complex plane into \( m = 2l \) equal sectors. Hence, the activation function (10) is 2-periodic (its period is 2) and \( l \)-multiple (a pair of outputs 1, −1 is repeated \( l \) times). While only threshold (linearly separable) Boolean functions may be learned by a neuron with activation function (9), a neuron with activation function (10) may learn non-threshold (nonlinearly separable) functions. Of course, such popular nonlinearly separable problems as XOR and parity \( n \) may also be learned using a single UBN [4], [24].

Thus, the UBN may be considered as the 2-valued MVN with the \( l \)-multiple extension of the activation function (9) to the 2-periodic activation function (10). As we see, such an extension increases the neuron’s functionality dramatically. This means that if some Boolean function is non-threshold and cannot be learned by a neuron with the activation function (9), it is possible to find such a value of \( m = 2l \) that this function may be projected into \( m \)-valued logic, where it becomes a partially defined (on the Boolean domain) \( m \)-valued threshold function. Let us try to
increase the MVN's functionality for any \( k \geq 2 \) in the same manner.

III. MVN-P

Let us consider the MVN input/output mapping described by some \( k \)-valued function \( f(x_1, \ldots, x_n) \), which is either

\[
f : E_k^n \rightarrow E_k \text{ or } f : O^n \rightarrow E_k.
\]

It is important to mention that since there exists a one-to-one correspondence between the sets \( K \) and \( E_k \) (see above), our function \( f \) can also be easily redefined as \( f_K : E_k^n \rightarrow K \) or \( f_K : O^n \rightarrow K \). These both definitions are equivalent.

If this function \( f(x_1, \ldots, x_n) \) is not a \( k \)-valued threshold function, it cannot be learned by a single MVN with the activation function (3).

Let us project the \( k \)-valued function \( f(x_1, \ldots, x_n) \) into \( m \)-valued logic, where \( m = kl \) and \( l \geq 2 \). To do this, let us define the following new discrete activation function for the MVN:

\[
P_l(z) = j \mod k, \text{ if } \frac{2\pi j}{m} \leq \text{arg} z < \frac{2\pi (j + 1)}{m}, \quad j = 0, 1, \ldots, m - 1; m = kl, l \geq 2. \tag{11}
\]

This definition is illustrated in Fig. 4. The activation function (11) separates the complex plane into \( m \) equal sectors and \( \forall d \in K \) there are exactly \( l \) sectors, in which the activation function (11) equals to \( d \).

This means that the activation function (11) establishes mappings from \( K \) into \( M = \{0, 1, \ldots, k - 1, k + 1, \ldots, m - 1\} \), and from \( E_k \) into \( E_m = \{1, e_m, e_m^2, \ldots, e_m^{m - 1}\} \), respectively.

Since \( m = kl \), then each element from \( M \) and \( E_m \) has exactly \( l \) prototypes in \( K \) and \( E_k \), respectively. In turn, this means that the neuron’s output determined by (11) is equal to

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
\]

\[l = m\]  \( \tag{12}\)

depending on which one of the \( m \) sectors (whose ordinal numbers are determined by the elements of the set \( M \)) the weighted sum is located in.

Hence, the MVN’s activation function in this case becomes \( k \)-periodic and \( l \)-multiple.

It is important that, in terms of multiple-valued logic, the activation function (11) projects a \( k \)-valued function \( f(x_1, \ldots, x_n) \) into \( m \)-valued logic. Evidently, \( f(x_1, \ldots, x_n) \) is a partially defined function in \( m \)-valued logic because \( K \subset M \), \( E_k \subset E_m \), and \( E_k^n \subset E_m^n \). Since \( f(x_1, \ldots, x_n) \) is a \( k \)-valued function, it takes only \( k \) values out of \( m \) in \( m \)-valued logic. The projection established by (11) may have a great practical sense if \( f(x_1, \ldots, x_n) \), being a non-threshold function in \( k \)-valued logic, will be a partially defined threshold function in \( m \)-valued logic and therefore it will be possible to learn it using a single MVN with the activation function (11). It will be shown below that this is definitely the case.

Let us refer to the MVN with the activation function (11) as the MVN-P.

It is important to mention that if \( l = 1 \) in (11), then \( m = k \) and the activation function (11) coincides with the activation function (3) accurate within the interpretation of the neuron’s output (if the weighted sum is located in the \( j \)th sector), then according to (3) the neuron’s output is equal to \( e^{j2\pi k} = e^{j} \in E_k \), which is the \( j \)th of \( k \)th root of unity, while in (11) it is equal to \( j \in K \), and the MVN-P becomes the regular MVN. It is also important to mention that if \( k = 2 \) in (11), then the activation function (11) coincides with the UBN activation function (10), sequence (12) becomes an alternating sequence \( 1, -1, 1, -1, \ldots \), and the MVN-P becomes the UBN. Hence, the MVN-P is a neuron for which both the MVN and the UBN are its particular cases.

IV. LEARNING ALGORITHM FOR THE MVN-P

To make the approach proposed in Section III active, it is necessary to develop an efficient learning algorithm for the MVN-P. Such an algorithm will be presented here.

As mentioned above (Section II.A), the MVN learning algorithm is based on the error-correction learning rule (4). It can also be based on its modifications (5)–(7). Let us adapt this algorithm to the MVN-P. Let us assume for simplicity, but without loss of generality, that the learning rule (4) will be used. It is important to mention that the learning rules (5)–(7) can also be used and it will be shown below (see Appendix) that the learning algorithm for the MVN-P converges independently of the particular rule (4)–(7), on which it is based.

Let \( f(x_1, \ldots, x_n) \) be a non-threshold function of \( k \)-valued logic. Thus, there is no way to learn it using a single MVN with activation function (3). Let us try to learn \( f(x_1, \ldots, x_n) \) in \( m \)-valued logic using a single MVN-P with activation function (11). Thus, the expected result of this learning process is the representation of \( f(x_1, \ldots, x_n) \) according to (2), where the activation function \( P_l \) determined by (11) substitutes for the activation function \( P \) determined by (3).

To organize this learning process, we will use the same learning rule (4), but applied to \( f(x_1, \ldots, x_n) \) as to the
Let us employ the same approach here for the MVN-P. Let \( \langle q \rangle \) to always move a weighted sum to the sector that is closest to the actual output of the UBN is not correct, it was suggested that by using two self-adaptive learning strategies, the desired output of the MVN-P may be determined during the learning process every time when the neuron’s output is incorrect.

The first strategy is based on the same idea that was used for the UBN [4]. As we have already mentioned (Section-II.B), if the actual output of the UBN is not correct, it was suggested to always move a weighted sum to the sector that is closest to the current weighted sum in terms of the angular distance. Let us employ the same approach here for the MVN-P. Let \( l \geq 2 \) in (11) and \( d \in [0, 1, \ldots, k-1] \) be the desired output. The activation function (11) determines the \( k \)-periodic and \( l \)-multiple sequence (12) with respect to sectors on the complex plane. Suppose that the current MVN-Ps output is not correct and the current weighted sum is located in the sector \( s \in M = \{0, 1, \ldots, m-1\} \), where \( m = kl \).

Since \( l \geq 2 \) in (11), there are \( l \) sectors on the complex plane where function (11) takes the correct value (see also Fig. 4). Two of these \( l \) sectors are the closest ones to sector \( s \) (from right and left sides, respectively). From these two sectors, we choose sector \( q \) whose border is closer to the current weighted sum \( z \) in terms of the angular distance. Then we take \( e^q \) as the desired output and apply learning rule (4). Hence, the first learning strategy for the MVN-P with the activation function (11) is as follows. Let a learning set for the function \( f(x_1, \ldots, x_n) \) be learned contain \( N \) learning samples and \( j \in \{1, \ldots, N\} \) be the number of the current learning sample, \( r \) be the number of the learning iteration, and Learning is a flag, which is “True” if the weights adjustment is required and “False” otherwise.

The iterative learning process for the first strategy consists of the following steps.

**Learning Strategy 1.**

1. Set \( r = 1 \), \( j = 1 \), and Learning = “False.”
2. Check (2) with the activation function (11) for the learning sample \( j \).
3. If (2) holds, then set \( j = j + 1 \), otherwise set Learning = “True” and go to Step 5.
4. If \( j \leq N \), then go to Step 2; otherwise go to Step 9.
5. Let \( z \) be the current value of the weighted sum and \( P(z) = e^q, s \in M, P(z) \) is the activation function (3), where \( m \) is substituted for \( k \). Hence the MVN-Ps actual output is \( P(z) = s \mod k \). Find \( q_1 \in M \), which determines the closest sector to the \( s \)th one, where the output is correct, from the right, and find \( q_2 \in M \), which determines the closest sector to the \( s \)th one, where the output is correct, from the left (this means that \( q_1 \mod k = d \) and \( q_2 \mod k = d \).
6. If \( \arg(z - \arg(e^{(q_1+1)2\pi/m})) \mod 2\pi \leq \arg(e^{(q_2)2\pi/m}) - \arg(z) \mod 2\pi; \) then \( q = q_1 \) else \( q = q_2 \).
7. Apply the learning rule (4) to adjust the weights.
8. Set \( j = j + 1 \) and return to Step 4.
9. If Learning = “False,” then go to Step 10; otherwise set \( r = r + 1, j = 1 \), Learning = “False” and go to Step 2.
10. End.

Let us now consider the second learning strategy, which is somewhat different. The activation function (11) divides the complex plane into \( l \) domains, and each of them consists of \( k \) sectors (Fig. 4). Since the function \( f \) to be learned is a partially defined function of \( m \)-valued logic (\( m = \ell k \)) in fact a \( k \)-valued function, then each of \( l \) domains contains those \( k \) values that may be used as the desired outputs of the MVN-P. Suppose that the current MVN-Ps output is not correct and the current weighted sum is located in the sector \( s \in M = \{0, 1, \ldots, m-1\} \). This sector in turn is located in the \( t \)th \( l \)-domain (out of \( l, t = [s/k] \)). Since there are \( l \) \( l \)-domains and each of them contains a potential correct output, we have \( l \) options to choose the desired output. Let us choose it in the same \( l \)-domain where the current actual output is located. Hence, \( q = tk + f_k(x_1, \ldots, x_n) \), where \( f_k(x_1, \ldots, x_n) \) is a desired value of function to be learned in terms of traditional multiple-valued logic (\( f_k(x_1, \ldots, x_n) \in K = \{0, 1, \ldots, k-1\} \)), respectively, \( f(x_1, \ldots, x_n) \in \theta_k = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\} \). Then we take \( e^q \) as the desired output and apply learning rule (4).

Let again a learning set for the function \( f(x_1, \ldots, x_n) \) to be learned contain \( N \) learning samples and \( j \in \{1, \ldots, N\} \) be the number of the current learning sample, \( r \) be the number of the learning iteration, and Learning is a flag which is “True” if the weights adjustment is required and “False” otherwise. The iterative learning process for the second strategy consists of the following steps.

**Learning Strategy 2.**

1. Set \( r = 1, j = 1, \) and Learning = “False.”
2. Check (2) with the activation function (11) for the learning sample \( j \).
3. If (2) holds, then set \( j = j + 1 \); otherwise set Learning = “True” and go to Step 5.
4. If \( j \leq N \), then go to Step 2; otherwise go to Step 8.
5. Let the actual neuron’s output is located in sector \( s \in M = \{0, 1, \ldots, m-1\} \). Then \( t = [s/k] \in \{0, 1, \ldots, l-1\} \) is the number of that \( l \)-domain, where sector \( s \) is located. Set \( q = tk + f_k(x_1, \ldots, x_n) \).
6. Apply the learning rule (4) to adjust the weights.
7. Set \( j = j + 1 \) and return to Step 4.
8. If Learning = “False,” then go to Step 9; otherwise set \( r = r + 1, j = 1 \), Learning = “False” and go to Step 2.
9. End.

The Learning Strategies 1 and 2 determine two variants of the same MVN-P learning algorithm, which is based on either of the learning rules (4)–(7). The convergence of this learning algorithm follows from the next theorem.
Theorem 1: If a \( k \)-valued function \( f(x_1, \ldots, x_n) \) is a non-threshold function of \( k \)-valued logic, but it is a partially defined threshold function of \( m \)-valued logic (where \( m = kl, l \geq 2 \)), then the MVN-P learning algorithm based on any of learning rules (4)–(7) and either of learning strategies 1 or 2 converges to the weighting vector of this function (a weighting vector of this function can always be obtained after a finite number of learning iterations).

The proof of this theorem is given in Appendix.

Thus, in other words, if a non-threshold \( k \)-valued function \( f(x_1, \ldots, x_n) \) is a partially defined \( m \)-valued threshold function, then it can be learned as an \( m \)-valued function by a single MVN-P, and the learning algorithm based on any of learning rules (4)–(7) and either of learning strategies 1 or 2 converges.

It is interesting that in terms of learning a \( k \)-valued function, the learning algorithm presented here is supervised. However, in terms of learning an \( m \)-valued function, this learning algorithm is unsupervised. We do not have prior knowledge about those \( m \)-valued output values that will be assigned to the input samples. The process of this assignment is self-adaptive, and this adaptation is reached by the learning procedure if a corresponding function is a partially defined \( m \)-valued threshold function.

It should be mentioned that, for \( k = 2 \) in (11), the MVN-P learning algorithm (Strategy 1) coincides with the UBN learning algorithm based on the error-correction rule [4]. On the other hand, for \( k > 2 \) and \( l = 1 \) in (11), the MVN-P learning algorithm (both Strategy 1 and Strategy 2) coincides with the MVN learning algorithm based on the error-correction rule [4]. The important conclusion that follows from this is that the concept of the MVN-P completely generalizes and includes the corresponding MVN and UBN concepts.

V. Simulation Results

In this section, we consider a number of simulations where the MVN-P and its learning algorithm are tested using some popular benchmarks. We used the MVN-P software simulator written in Borland Delphi 5.0 environment, running on a PC with an Intel Core2 Duo CPU.

A. Iris

This famous benchmark database was downloaded from the UC Irvine Machine Learning Repository [25]. The dataset contains three classes of 50 instances each, where each class refers to a type of iris plant. Four real-valued (continuous) features are used to describe the data instances. Thus, we have here a four-dimensional three-class classification problem. It is known [25] that the first class is linearly separable from the other two but the latter are not linearly separable from each other. Thus, a regular single MVN with the activation function (3), as well as any other single artificial neuron, cannot learn this problem.

However, a single MVN-P with the activation function (11) \((l = 3, k = 3, m = 9)\) learns the Iris problem completely with no errors. To transform the input features into the numbers located on the unit circle, we use (1) with \( \alpha = 2\pi/3 \). It is to be pointed out that the problem is really complicated and is not so easy to learn it. For example, the learning algorithm based on the Learning Strategy 1 does not converge even after 55 000 000 iterations independently from the learning rule that is applied. However, the learning algorithm based on the Learning Strategy 2 and the learning rule (7) converges with zero error. Seven independent runs of the learning algorithm starting from the different random weights\(^2\) converged after 9 379 027–43 878 728 iterations. Every time the error decreases very quickly and after 50–100 iterations there are stably one or just a few more samples which still require the weights adjustment, but their final adjustment takes time (5–12 h). Nevertheless, this result is very interesting because, to our best knowledge, this is the first time when the “Iris” problem was learned using just a single neuron. It is interesting that, after the convergence, for the first class (known and referred to as “Iris Setosa” [25]), the weighted sums for all instances appear in the same single sector on the complex plane, for the second class (“Iris Versicolour”) about two-thirds of them appear in the one sector but about one-third appear in another one located in a different \( l \)-domain [see (11) and Fig. 4]. For the third class, the weighted sums for all the instances except one appear in the same single sector on the complex plane, but for the one instance (every time the same) it appears in the different sector belonging to the different \( l \)-domain. In this way, the second and the third classes, which initially are known as nonlinearly separated, become separated. This effect is achieved just by the self-adaptation of the learning algorithm.

Another important experiment with the “Iris” dataset was checking the MVN-Ps ability to solve a classification problem. We used a five-fold cross validation. The dataset was every time randomly separated into a learning set containing 75 samples (25 from each class) and a testing set also containing 75 samples. The best results are obtained for the activation function (11) with \( l = 2, k = 3, m = 6 \). The Learning Strategy 1 and the learning rule (4) were used. The learning algorithm requires for its convergence with the zero error 10–288 iterations (which takes just a few seconds). The classification results are absolutely stable, 73 of 75 instances are classified correctly (the classification rate is 97.33%). These results practically coincide with the best known results for this benchmark data set [26] (97.33 for the one-against-one SVM and 97.62 for the dendogram-based SVM). However, the one-against-one SVM for three classes contains three binary decision SVMs, the dendogram-based SVM for three classes contains five binary decision SVMs, while we use just a single MVN-P.

B. Two Spirals

The two-spirals problem is a well-known non linearly separable classification problem where the two spirals point (see Fig. 5) must be classified as belonging to the first or to the second spiral. Thus, this is 2-D two-class classification problem. The standard two spirals dataset usually consists of 194 points (97 belong to the first spiral and other 97

\(^2\)Here and further the initial weights (both real and imaginary parts) are random numbers from the interval \([0,1]\) generated using a standard generator.
are known as the best for this problem so far. The two-points belong to the second spiral). The following results
Fig. 5. Two spirals.
the learning algorithm for each of the learning rules. We
shows an accuracy of about 70% [11].
A single MVN-P with the activation function (11) \((l = 2, k = 2, m = 4)\) significantly outperforms all the mentioned
techniques. Just 2–6 learning iterations are required to learn
the two-spirals problem completely with no errors using
the Learning Strategy 1 and either of the learning rules (4) or
(5). These results are based on the 10 independent runs of
the learning algorithm for each of the learning rules. We
also used 10 independent runs to check the classification
ability of a single MVN-P with the activation function (11)
\((l = 2, k = 2, m = 4)\) using the cross-validation. The dataset
was divided into the learning set (98 samples) and the testing
set (96 samples). We achieved absolute success in this testing:
100% classification rate is achieved in all experiments. Just
2–5 iterations were needed to learn the learning set using the
Learning Strategy 1 and either of the learning rules (4) or
(5).

C. Breast Cancer Wisconsin (Diagnostic)

This famous benchmark dataset was downloaded from the
UC Irvine Machine Learning Repository [25]. The dataset
contains two classes which are represented by 569 instances
(357 benign and 212 malignant), which are described by 30
real-valued features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
features. Thus, this is 30-dimensional, two-class

The two-spirals problem completely with no errors using
the Learning Strategy 1 and either of the learning rules (4) or
(5). These results are based on the 10 independent runs of
the learning algorithm for each of the learning rules. We
also used 10 independent runs to check the classification
ability of a single MVN-P with the activation function (11)
\((l = 2, k = 2, m = 4)\) using the cross-validation. The dataset
was divided into the learning set (98 samples) and the testing
set (96 samples). We achieved absolute success in this testing:
100% classification rate is achieved in all experiments. Just
2–5 iterations were needed to learn the learning set using the
Learning Strategy 1 and either of the learning rules (4) or
(5).

C. Breast Cancer Wisconsin (Diagnostic)

This famous benchmark dataset was downloaded from the
UC Irvine Machine Learning Repository [25]. The dataset
contains two classes which are represented by 569 instances
(357 benign and 212 malignant), which are described by 30
real-valued features. Thus, this is 30-dimensional, two-class
classification problem. To transform the input features into the
numbers located on the unit circle, we used (1) with \(a = 6.0\).
The whole dataset may be easily learned by a single MVN-
P with the activation function (11) \((l = 2, k = 2, m = 4)\).
Ten independent runs give 280–370 iterations for the Learning Strategy 1 and learning rule (4) and 380–423 iterations for the Learning Strategy 2 and learning rule (4) (these are the best results among the different combinations of learning strategies and rules).

To check the classification ability of a single MVN-P, we used 10-fold cross-validation as recommended for this dataset, for example, in [25] and [30]. The entire dataset was randomly divided into the 10 subsets: 9 of them contained 57 samples and the last one contained 56 samples. The learning set every time was formed from 9 of 10 subsets and the remaining subset was used as the testing set. We used the same parameters in the activation function (11) \((l = 2, k = 2, m = 4)\). The best average classification accuracy (97.68\%) was achieved after learning using the Learning Strategy 2 and learning rule (4). The learning process required 74–258 iterations. The classification accuracy is comparable with the results reported for SVM in [30] (98.56\% for the regular SVM and 99.29\% for the SVM with an additional “majority decision” tool) and slightly better than 97.5\% reported as the estimated accuracy for the different classification methods in [25]. However, a single MVN-P uses fewer parameters than, for example, SVM (the average amount of support vectors for different kernels used in [30] is 54.6, whereas the MVN-P uses 31 weights).

D. Mod \(k\) Addition of \(n\) \(k\)-Valued Variables

This problem is a generalization of the famous Parity \(n\) problem for the \(k\)-valued case. In fact, Parity \(n\) problem is a mod 2 addition of \(n\) variables. It is well known that this is a classical nonlinearly separable problem. However, it may be learned by a single UBN [24]. As mentioned in [3], mod \(k\) addition of \(n\) variables is a non-threshold multiple-valued function for any \(k\) and any \(n\) and therefore it cannot be learned by a single MVN. To our best knowledge, there is no evidence that this function can be learned by any other single neuron. However, as we will see now, it is not a problem to learn this function using a single MVN-P with the activation function (11). The problem of mod \(k\) addition of \(n\) variables is very interesting, but its deeper study and possible universal solution in terms of the relationship between \(k\), \(n\) on the one side and \(l\) in (11) on the other side is beyond the scope of this paper. We just want to show here that this multiple-valued problem is really solvable at least for those \(k\) and \(n\), for which we have performed experimental testing. The experimental results are summarized in Table I. Since the Learning Strategy 1 showed better performance for this problem (fewer learning iterations and time), all results are given for this strategy only.

It should be mentioned that none of learning rules (4)–(7) can be distinguished as the “best.” Each of them can be good for solving a different problem. It is also not possible to distinguish the better one between Strategy 1 and Strategy 2. For example, the “Iris” problem can be learned using Strategy 2, while for some other problems Strategy 1 gives better results.

VI. Conclusion and Future Work

In this paper, a new periodic activation function and a modified learning algorithm were proposed for the MVN. These innovations brought us to the concept of the MVN-P. It was shown that this concept generalizes the previously developed concepts of MVN and UBN. Both become particular cases of the MVN-P. It was shown that a single MVN-P has much higher functionality than a single MVN. A single MVN-P can learn popular nonlinearly separable benchmark problems. This is possible because a newly introduced periodic activation function projects an initial problem from \(k\)-valued logic, where it is described by a non-threshold function, to \(kl = m\)-valued logic, where this function becomes a partially defined threshold function.

A learning algorithm, which was developed for the MVN-P, integrates four modifications of the error-correction learning rule and two learning strategies.

It is important to point out some directions for further development of the presented results. In our view, it will be very interesting to use the MVN-P as a basic neuron in neural networks, for example, as an output neuron in a feedforward neural network. It should be expected that in this way the functionality of a network might be significantly increased. This will also be very important for solving those highly nonlinear multiple-class classification problems where classes are multi-clustered and therefore nonlinearly separable.

VII. Appendix

We want to prove here Theorem 1. Let us first consider the learning rule (4). Thus, we have to prove that, if a non-threshold \(k\)-valued function \(f(x_1,\ldots,x_n)\) is a partially defined \(m\)-valued threshold function \([f(x_1,\ldots,x_n)\) is an \(m\)-valued threshold function on its whole domain], then it can be learned by a single MVN-P, and a weighting vector for this function may be obtained by the learning algorithm based on the learning rule (4). This means that the learning algorithm converges to the weighting vector after a finite number of learning iterations.

Since we are given a condition that \(f(x_1,\ldots,x_n)\) is an \(m\)-valued threshold function, this means that there exists a weighting vector \(W = (w_0, w_1,\ldots, w_m)\) such that (2) holds for any \(X = (x_1,\ldots,x_n)\) from the domain of \(f\). For simplification, we may rewrite (2) as \(P(X, W) = f(X)\), where \(W\) is a vector whose components are complex-conjugated to the ones of \(W\), and \((X, W)\) is a dot product of vectors \(X\) and \(W\) in the complex (unitary) space.

Let us now look for a weighting vector applying the learning rule (4) according either to the Learning Strategy 1 or Learning Strategy 2. As we mentioned, we may set \(C_r = 1\) in (4) for any \(r\). For simplicity and without loss of generality, let us start the learning process from the zero vector \(W_1 = ((0,0),(0,0),\ldots,(0,0))\), where \((a,b)\) is a complex number \(a + bi\), where \(i\) is an imaginary unity. Let \(S_X = (X_1, X_2,\ldots, X_N)\) be a learning sequence of input vectors \(X_j = (x_{1j},\ldots,x_{nj})\), \(j = 1,\ldots,N\) and \(S_W = (W_1, W_2,\ldots, W_r,\ldots)\) be a sequence of weighting vectors, which appear during the learning process. We have to prove that this sequence cannot be infinite. Let us remove from the learning sequence those vectors for which \(W_{r+1} = W_r\), in other words, those input vectors for which (2) holds without
learning. Let \( S^{W}_{r} \) be the reduced sequence of the weighting vectors. The theorem will be proven if we show that the sequence \( S^{W}_{r} \) is finite. Let us suppose that the opposite is true: i.e., the sequence \( S^{W}_{r} \) is infinite. Let \( e^{q_{2}} \) be the actual output for the input vector \( X_{1} \), and let the weighting vector \( W_{1} \) and \( e^{q_{1}} \) be the desired output defined according to the Step 6 of the Learning Strategy 1 or Step 5 of the Learning Strategy 2. Since the desired and actual outputs do not coincide with each other, we have to apply the learning rule (4) to adjust the weights. According to (4), we obtain

\[
\begin{align*}
W_{2} &= \frac{1}{n+1} (e^{q_{1}} - e^{q_{2}}) \tilde{X}_{1} \\
W_{3} &= W_{2} + \frac{1}{n+1} (e^{q_{2}} - e^{S_{2}}) \tilde{X}_{2} \\
&= \frac{1}{n+1} \left[ (e^{q_{1}} - e^{S_{1}}) \tilde{X}_{1} + (e^{q_{2}} - e^{S_{2}}) \tilde{X}_{2} \right] \\
\bar{W}_{r+1} &= \frac{1}{n+1} \left[ (e^{q_{1}} - e^{S_{1}}) \tilde{X}_{1} + \ldots + (e^{q_{r}} - e^{S_{r}}) \tilde{X}_{r} \right].
\end{align*}
\]

Let us find a dot product of both parts of (13) with \( W \)
\[
\left( \bar{W}_{r+1}, \bar{W} \right) = \frac{1}{n+1} \left[ \left( \bar{W}_{r+1}, \bar{W} \right) + \left( \bar{W}_{r}, \bar{W} \right) \right] + \left( e^{q_{r}} - e^{S_{r}} \right) \tilde{X}_{r}.
\]

Let \( e^{q_{j}} - e^{q_{j}} = \lambda_{j}, j = 1, \ldots, r \). Then the last equation may be written as follows:
\[
\left( \bar{W}_{r+1}, \bar{W} \right) = \frac{1}{n+1} \left[ (\lambda_{1} \tilde{X}_{1}, \bar{W}) + \ldots + (\lambda_{r} \tilde{X}_{r}, \bar{W}) \right].
\]

Let us estimate the absolute value of the sum in the right-hand side of (15) is always greater than or equal to the absolute values of the real and imaginary parts of this sum.

Let \( a = \min_{j=1, \ldots, r} |Re(\lambda_{j} \tilde{X}_{j}, \bar{W})| \). Then it follows from (15) that
\[
\left( \bar{W}_{r+1}, \bar{W} \right) \geq \frac{ra}{n+1}.
\]

On the other hand, according to the fundamental Schwarz inequality, the squared dot product of the two vectors does not exceed the product of the squared norms of these vectors or, in other words, the norm of the dot product of the two vectors does not exceed the product of the norms of these vectors. Thus, according to the Schwartz inequality
\[
\left| \left( \bar{W}_{r+1}, \bar{W} \right) \right| \leq \left\| \bar{W}_{r+1} \right\| \cdot \left\| \bar{W} \right\|.
\]

Taking into account (16), we obtain from (17) \( ra/(n+1) \leq \left| \left( \bar{W}_{r+1}, \bar{W} \right) \right| \leq \left\| \bar{W}_{r+1} \right\| \cdot \left\| \bar{W} \right\| \). Then it follows from the last inequality that
\[
\left\| \bar{W}_{r+1} \right\| \geq \frac{ra}{\left\| \bar{W} \right\| (n+1)}.
\]

Let for simplicity \( a/(n+1) = \bar{a} \). Then (18) is transformed as follows:
\[
\left\| \bar{W}_{r+1} \right\| \geq \frac{r\bar{a}}{\left\| \bar{W} \right\|}.
\]

As mentioned, \( W \) is some weight vector for our function \( f \). This vector exists according to Definition 2 because we are given a condition that \( f \) is a partially defined \( m \)-valued threshold function. According to our assumption, the sequence \( S^{W}_{r} \) of the weighting vectors is infinite. Since \( r \) is the number of learning iteration, let us consider (19) when \( r \rightarrow \infty \).

\[ \left\| \bar{W}_{r+1} \right\| \geq \frac{r\bar{a}}{\left\| \bar{W} \right\|} \rightarrow \infty. \]
changes. If we apply the learning rule (7), then we again have to substitute (13), this time as follows:

\[
\tilde{w}_{r+1} = \frac{1}{n+1} \left[ \frac{1}{|z_1|} \left( e^{q_1} - \frac{z_j}{|z_1|} \right) \tilde{x}_1 + \ldots \right. \\
+ \left. \frac{1}{|z_r|} \left( e^{q_r} - \frac{z_j}{|z_r|} \right) \tilde{x}_r \right]
\]

Then putting \((1/|z_j|)(e^{q_j} - (z_j/|z_j|)) = k_j, j = 1, \ldots, r\), we obtain (14) and from that moment the proof continues again with no changes.

It should be mentioned that the convergence of the learning algorithms does not depend on the choice of the learning strategy (1 or 2). The choice of the learning strategy may only increase or decrease the number of learning iterations. If a function to be learned is a partially defined \(m\)-valued threshold function, the learning process converges anyway.

REFERENCES


Igor Aizenberg (M’91–SM’06) received the M.Sc. degree in mathematics from Uzhgorod National University, Uzhgorod, Ukraine, in 1982, and the Ph.D. degree in computer science from the Dorodnicyn Computing Center of the Russian Academy of Sciences, Moscow, Russia, in 1986.

He was a Research Scientist with the Institute for Information Transmission Problems of the Russian Academy of Sciences, Moscow, Russia, from 1982 to 1990. From 1990 to 1993, he was an Assistant Professor, and then from 1993 to 1996 and 1998 to 1999, he was an Associate Professor with the Department of Cybernetics at Uzhgorod National University. From 1996 to 1998, he was a Research Scientist with the Department of Electrical Engineering at the Catholic University of Leuven, Leuven, Belgium. From 1999 to 2002, he was a Vice President of Research at Neural Networks Technologies, Ramat-Gan, Israel. From 2002 to 2006, he was a Visiting Research Professor at the Dortmund University of Technology, Dortmund, Germany, and Tampere University of Technology, Tampere, Finland. From 2003 to 2010, he was also serving a number of times as a Visiting Professor in schools organized at the University of Zaragoza, Zaragoza, Spain, University of Nis, Nis, Serbia, and Technical University of Porto, Porto, Portugal. From March 2006, he is with Texas A&M University-Texarkana, Texarkana, where he currently is the Arnold Associate Professor of Computer Science and Director of the Computational Intelligence Laboratory. His current research interests include complex-valued neural networks, pattern recognition, image processing, and spectral techniques.