UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

Multi-Layered Neural Network Based on Multi-Valued Neurons (MLMVN) and a Backpropagation Learning Algorithm

Igor Aizenberg and Claudio Moraga

No. CI-171/04

Technical Report ISSN 1433-3325 April 2004

Secretary of the SFB 531 · University of Dortmund · Dept. of Computer Science/XI
44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, "Computational Intelligence", at the University of Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

Multi-Layered Neural Network based on Multi-Valued Neurons (MLMVN) and a Backpropagation Learning Algorithm

Igor Aizenberg Claudio Moraga

Abstract

A multi-layered neural network based on multi-valued neurons is considered in the paper. It is shown that using a traditional architecture of multi-layered feedforward neural network (MLF) and the high functionality of the multi-valued neuron, it is possible to obtain a new powerful neural network. Its training does not require a derivative of the activation function and its functionality is higher than the functionality of MLF containing the same number of layers and neurons. These advantages of MLMVN are confirmed by testing using Parity *n*, two spirals and "sonar" benchmarks.

I. INTRODUCTION

Neural networks with a backpropagation learning algorithm have started their history from the ideas presented by D.E. Rumelhart and J.L McClelland in [1]. These neural networks are characterized by a multi-layer architecture with a feedforward dataflow through nodes requiring full connection between consecutive layers. This architecture is the result of a "universal approximator" computing model based on Kolmogorov's Theorem [2] (see e.g. [3], [4] and the more comprehensive observation done by R. Hecht-Nielsen in [5]). It has been shown in [6]-[7] that these neural networks are universal approximators. So there are two main ideas behind a feedforward neural network. The first idea is a full connection architecture: the outputs of neurons from the previous layer are connected with the corresponding inputs of all neurons of the following layer. The second idea is a backpropagation learning algorithm, when the errors of the neurons from the output layer are being sequentially backpropagated through all the layers from the "right hand" side to the "left hand" side, in order to calculate the errors of all other neurons. One more common property of a major part of the feedforward networks is the use of sigmoidal activation functions for its neurons.

It is possible to find hundreds of papers and many books published during last 10-15 years, where the ideas of multi-layered neural networks and backpropagation learning were developed. One of the most comprehensive observations is presented e.g. in [8].

At the same time there is at least one important problem, which is still open. Although it is proven that a feedforward network is a universal approximator, a practical implementation of learning often is a very complicated task. It depends on several factors: the complexity of the mapping to be implemented, the chosen structure of the network (the number of hidden layers, the number of neurons on each layer), and the control over the learning process (usually this control is being implemented using the learning rate). Increasing both the number of hidden layers and neurons on them, we can make the network more flexible to the mapping to be implemented. This corresponds to the well-known Cover's theorem [9] on the separability of patterns, which states that a pattern classification problem is more likely to be linearly separable in a high dimensional feature space than in a low dimensional one, while projected into a high dimensional space nonlinarly. However, the computations in a higher dimensional space require more resources and much more time. The case of MLF and similar networks is not an exclusion. Moreover, increasing the number of hidden neurons increases the risk of overfitting.

In order to minimize the implementation of complex mappings, several supporting algorithms were proposed. A very popular family of kernel-based learning algorithms that use nonlinear mappings from input spaces to high dimensional feature spaces should be mentioned. The best known of them is the support vector machine (SVM) introduced in [10]. Different kernel-based techniques are considered e.g. in [11]. A fuzzy kernel perceptron [12] should be distinguished among the most recent publications, where the kernel-based approach is developed.

Another direction is related to improvement of the MLF learning, search for more sophisticated learning techniques, as well as different modifications of the MLF structure and architecture. From the very recent publications we can mention [13], where the modular feedforward networks, with not fully connected neighboring layers are studied and [14], where the original modification of the MLF learning algorithm is considered.

At the same time it is very attractive to consider a different solution, which will preserve a traditional MLF architecture, however, it will be based on the use of the different basic neurons. We will consider in this paper a *multi-layered neural network based on multi-valued neurons* (MLMVN). A multi-valued neuron (MVN) was introduced in [15]. It is based on the principles of multiple-valued threshold logic over the field of the complex numbers formulated in [16] and then developed in [17]. A comprehensive observation of MVN, its properties and learning is presented in [18]. The most important properties of MVN are: the complex-valued weights, inputs and output coded by the k^{th} roots of unity and the activation function, which maps the complex plane into the unit circle. It is very important for us that MVN learning is reduced to the movement along the unit circle. The MVN learning algorithm is based on the simple linear error correction rule and does not require a derivative. Moreover, it converges very quickly. Different applications of MVN have been considered during the last years. We will mention some of them: MVN has been used as a basic neuron in cellular neural networks [18], as a basic neuron of neural-based associative memories [18], [19]-[22] and as the basic neuron of different pattern recognition systems [22]-[24].

It is very important that the functionality of a single MVN is higher than the functionality of a single neuron with a sigmoid activation function [18]. So it is very attractive to consider a multi-layered neural network with the same architecture as MLF, but with MVN as a basic neuron. It should be mentioned that the feedforward networks based on the neurons with the complex-valued weights have been already considered, for example in [25]-[27]. But in these papers a classical sigmoid function and a classical backpropagation algorithm were generalized for the complex-valued valued case. We will consider here a different solution.

It is also important to mention that the development of the complex-valued neural networks is becoming more and more popular. It is possible to refer to the recently published book [28], where, for example, the MVN-based associative memory presented in [20] and the complex domain backpropagation presented in [27] are observed in more details.

In the Section II we will remind some basic ideas related to MVN and its learning. A modified continuous-valued activation function also will be introduced. In the Section III MLMVN will be introduced. We will consider a backpropagation learning algorithm for it, which does not require a derivative of the activation function. Two possible variants of the backpropagation will be considered. Simulation results will be presented in the Section IV. Using some standard benchmarks, we will show the efficiency of MLMVN. It will be shown that such popular problems as parity *N*, two spirals and "sonar" can be solved using a simpler and smaller network than the known ones.

II. Discrete and Continuous MVN

Let us remind some basic ideas related to MVN and its training. A single MVN performs a mapping between *n* inputs and a single output ([15], [18]). This mapping is described by a multiple-valued (*k*-valued) function of *n* variables $f(x_1,\dots,x_n)$ with *n*+1 complex-valued weights as parameters:

$$f(x_1,\dots,x_n) = P(w_0 + w_1x_1 + \dots + w_nx_n) \tag{1}$$

where $x_1,\dots,x_n$ are the variables, on which the performed function depends and $w_0, w_1, \dots, w_n$ are the weights. The values of the function and of the variables are complex. They are the k^{th} roots of unity: $\varepsilon^j = \exp(i2\pi j/k)$, $j \in \{0, k\text{-}1\}$, *i* is an imaginary unity. *P* is the activation function of the neuron:

$$P(z) = \exp(i2\pi j/k),\ if\ \ 2\pi j/k \le \arg z < 2\pi\ (j+1)/k \tag{2}$$

where *j*=0, 1, ..., *k*-1 are values of the *k*-valued logic, $z = w_0 + w_1x_1 + \dots + w_nx_n$ is the weighted sum, arg *z* is the argument of the complex number *z*. Equation (2) is illustrated in Fig. 1. Function (2) divides a complex plane onto *k* equal sectors and maps the whole complex plane into a subset of points belonging to the unit circle. This is exactly a set of k^{th} roots of unity. Function (2) was initially proposed in [16] as a *k*-valued predicate *csign*, which is a key element of multiple-valued threshold logic over the field of the complex numbers [17]-[18].

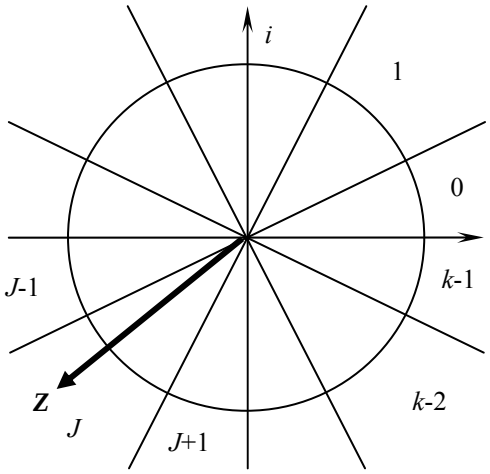


Fig. 1 Geometrical interpretation of the MVN activation function

MVN learning is reduced to the movement along the unit circle. This movement does not require a derivative of the activation function, because it is impossible to move in the incorrect direction. Any direction of movement along the circle will lead to the target. The shortest way of this movement is completely determined by the error that is a difference between the "target" and the "current point", i.e. between the desired and actual output, respectively. This MVN property is very important for the further development of the learning algorithm for a multi-layered network. Let us consider how it works.

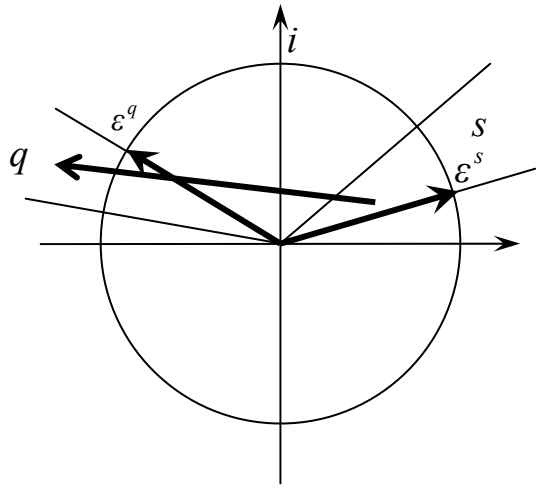


Fig. 2 Geometrical interpretation of the MVN learning rule

Let $\varepsilon^q = T$ be a desired output of the neuron (see Fig. 2). Let $\varepsilon^s = Y = P(z)$ be an actual output of the neuron. The MVN learning algorithm based on the error correction learning rule is defined as follows [18]:

$$W_{m+1} = W_m + \frac{C_m}{(n+1)}(\varepsilon^q - \varepsilon^s)\,\overline{X}, \tag{3}$$

where X is an input vector[1], n is the number of neuron inputs, $\overline{X}$ is a vector with the components complex conjugated[2] to the components of vector X, m is the number of the learning iteration, W_m is a current weighting vector (to be corrected), W_{m+1} is the following weighting vector (after correction), C_m is a

[1] We will add to the n-dimensional vector X an $n+1^{th}$ component $x_0 \equiv 1$ realizing a bias, in order to simplify mathematical expressions for the correction of the weights

[2] Here and further $\overline{x}$ is a number complex conjugated to x and $\overline{X}$ is a vector with the components complex conjugated to the components of X.

learning rate. The convergence of the learning process based on the rule (3) is proven in [18]. What is a sense of the rule (3)? It ensures such a correction of the weights that a weighted sum is moving from the sector s to the sector q (see Fig. 2). The direction of this movement is completely defined by the difference $\delta = \varepsilon^q - \varepsilon^s$. Thus $\delta = \varepsilon^q - \varepsilon^s$ determines the MVN error. According to (3) a correcting item $\Delta w_i = \frac{C_m}{(n+1)}(\varepsilon^q - \varepsilon^s)\,\bar{x}_i, i = 0,1,...,n$, which is added to the corresponding weight in order to correct it, is proportional to δ.

The correction of the weights according to (3) changes the value of the weighted sum exactly on δ. Indeed, let $z = w_0 + w_1 x_1 + ... + w_n x_n$ be a current weighted sum. Let us correct the weights according to the rule (3) (we take C=1):

$$\tilde{w}_0 = w_0 + \frac{\delta}{(n+1)};\quad \tilde{w}_1 = w_1 + \frac{\delta}{(n+1)}\bar{x}_1;\quad ...;\quad \tilde{w}_n = w_n + \frac{\delta}{(n+1)}\bar{x}_n.$$

The weighted sum after the correction is obtained as follows:

$$\begin{aligned}
&\tilde{z} = \tilde{w}_0 + \tilde{w}_1 x_1 + ... + \tilde{w}_n x_n = \\
&= (w_0 + \frac{\delta}{(n+1)}) + (w_1 + \frac{\delta}{(n+1)}\bar{x}_1)x_1 + ... + (w_1 + \frac{\delta}{(n+1)}\bar{x}_n)x_n = \\
&= w_0 + \frac{\delta}{(n+1)} + w_1 x_1 + \frac{\delta}{(n+1)} + ... + w_n x_n + \frac{\delta}{(n+1)} = \\
&= w_0 + w_1 x_1 + ... + w_n x_n + \delta = z + \delta.
\end{aligned} \tag{4}$$

Equation (4) shows the importance of factor $\frac{1}{n+1}$ in the learning rule (3). This factor shares the error δ uniformly among the neuron's inputs.

Evidently, the activation function (2) is discrete. More exactly, it is piece-wise discontinuous, it has discontinuities on the borders of the sectors. Let us modify the function (2) in order to generalize it for the continuous case in the following way. Let us consider, what will happen, when $k \to \infty$ in (2). It means that the angle value of the sector (see Fig. 1) will go to zero. It is easy to see that the function (2) is transformed in this case as follows:

$$P(z) = \exp(i\,(\arg z)) = e^{iArg\,z} = \frac{z}{|z|}, \tag{5}$$

where z is the weighted sum, Arg z is a main value of its argument and $|z|$ is a modulo of the complex number z.

The function (5) maps the complex plane into a whole unit circle, while the function (2) maps a complex plane just on a discrete subset of the points belonging to the unit circle. The function (2) is discrete, while the function (5) is continuous. We will use here exactly the function (5) as the activation function for the MVN. Both functions (2) and (5) are not differentiable as functions of a complex variable, but this is not important, because their differentiability is not required for MVN learning. The learning rule (3) will be modified for the continuous-valued case in the following way:

$$W_{m+1} = W_m + \frac{C_m}{(n+1)}(\varepsilon^q - e^{iArg\,z})\,\overline{X} = W_m + \frac{C_m}{(n+1)}\left(\varepsilon^q - \frac{z}{|z|}\right)\overline{X}. \tag{6}$$

It is absolutely clear that convergence of the learning algorithm based on the learning rule (6) may be proven in the same way as it was done in [18] for the rule (3).

Let us consider how the weighted sum is being moved to the correct direction according to the rule (6). Let T be a desired and $Y = \frac{z}{|z|}$ be an actual output. Thus, the error and simultaneously the direction of the correction is determined by the difference

$$\delta = T - Y = T - \frac{z}{|z|} = \frac{T|z| - z}{|z|}. \tag{7}$$

Taking into account (7) and we can now transform (4) as follows:

$$\tilde{z} = z + \delta = z + \left(T - \frac{z}{|z|}\right) = z + T - \frac{z}{|z|} = T + z - \frac{z}{|z|}, . \tag{8}$$

and it is clear that a step of learning determined by (6) is as successful, as smaller is the absolute value of $z - \frac{z}{|z|}$, because it ensures a smaller deviation of $\arg\left(T + z - \frac{z}{|z|}\right)$ from $\arg T$.

It is also interesting to consider the following modification of (6):

$$W_{m+1} = W_m + \frac{C_m}{(n+1)} \tilde{\delta}\, \overline{X}, \tag{9}$$

where $\tilde{\delta}$ is obtained from $\delta = \varepsilon^q - \frac{z}{|z|} = T - \frac{z}{|z|}$ using a normalization by the factor $\frac{1}{|z|}$:

$$\tilde{\delta} = \frac{1}{|z|}\delta = \frac{1}{|z|}\left(T - \frac{z}{|z|}\right), \tag{10}$$

and instead of (8) we obtain the following:

$$\tilde{z} = z + \tilde{\delta} = z + \frac{1}{|z|}\left(T - \frac{z}{|z|}\right) = z + \frac{T}{|z|} - \frac{z}{|z|^2} = \frac{T}{|z|} + z - \frac{z}{|z|^2}. \tag{11}$$

Learning according to the rule (9)-(10) makes it possible to squeeze a space for the possible values of the weighted sum.

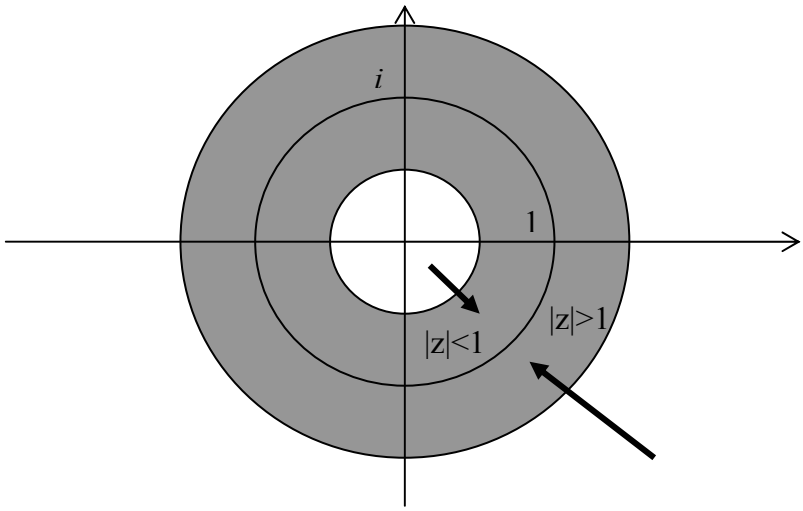


Fig. 3. Normalization of the weighted sum z by the factor 1/|z| (see (9)-(11))

Using (9)-(10) instead of (6), we can reduce this space to the respectively narrow ring, which will include the unit circle inside (see Fig. 3). Indeed, if $|z|<1$ and we correct the weights according to (9)-(10), then according to (11) $|\tilde{z}|>|z|$ and $\tilde{z}$ will be closer to the unit circle than *z*, approaching to the unit circle form "inside". If $|z|>1$ and we correct the weights according to (9)-(10), then according to (11) $|\tilde{z}|>|z|$ and $\tilde{z}$ will be closer to the unit circle than *z*, approaching to the unit circle form "outside". This approach can be useful in order to make *z* more smooth as a function of the weights and to exclude a situation, when a small change either of the weights or the inputs will lead to the significant change of *z*. At the same time the choice of the learning rule depends on a particular mapping that we want to implement. For example, if it is described by the smooth function, there is no reason to use (9)-(10) adding more calculations. We will return below to the choice of the learning rule, when we will discuss a learning algorithm for the MVN-based neural network.

III. Multi-Layered MVN-based Neural Network and a Backpropagation Learning Algorithm

Let us consider a multi-layered neural network with traditional feedforward architecture, when the outputs of neurons of the input and hidden layers are connected with the corresponding inputs of the neurons from the following layer. Let us suppose that the network contains one input layer, *m*-1 hidden layers and one output layer. We will use here the following notations.

Let

T_{km} - be a desired output of the k^{th} neuron from the m^{th} (output) layer

Y_{km} - be an actual output of the k^{th} neuron from the m^{th} (output) layer.

Then a global error of the network for the k^{th} neuron of the m^{th} (output) layer can be calculated as follows:

$$\delta^*_{km} = T_{km} - Y_{km} \text{ - error for the } k^{th} \text{ neuron from output layer} \tag{12}$$

δ^*_{km} will denote here and further a global error of the network. We have to distinguish it from the local errors δ_{km} of the particular neurons.

The learning algorithm for the classical feedforward network is derived from the consideration that a global error of the network expressed in the terms of squared error (SE) must be minimized. The functional of error may be defined as follows:

$$\mathrm{E} = \frac{1}{2}\sum_k (\delta^*_{km})^2 (W) \tag{13}$$

where δ^*_{km} is a global error of the k^{th} neuron of the m^{th} (output) layer and W are the weighting vectors of all the neurons of the network (it is principal that the error depends not only on the weights of the neurons from the output layer, but on all neurons of the network). The minimization of the functional (13) is reduced to the search for those weights for all the neurons that ensure a minimal MSE.

Let us remind briefly how it works for MLF. The most important problem for network learning is to express the error of the each neuron through the global errors of the network. This problem is solved using the backpropagation of the global errors through the network: from the output layer through all the hidden layers. It is also well known that the correction of the weights for all neurons is organized in such a way that each weight w_i has to be corrected by a correcting item Δw_i, which must be proportional to the gradient $\frac{\partial E}{\partial w_i}$ of the error function $E(w)$ with respect to the weights [8]. This lead to the following:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \delta} \cdot \frac{\partial \delta}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_i}, \; i = 0, 1, ..., n .$$

For each neuron of the network this gives the following ($y(z)$ is a neuron's activation function, z is the weighted sum, δ is the error for the considered neuron). Since

$$\frac{\partial z}{\partial w_i} = (w_0 + w_1 x_1 + ... + w_n x)' = x_i\ ,\ i = 0, 1, ..., n;\ x_0 \equiv 1\ ;$$

$$\frac{\partial y}{\partial z} = y'(z)\ ;\ \frac{\partial \delta}{\partial y} = (T - Y)' = -1$$

and according to (13) $\frac{\partial E}{\partial \delta} = \delta$, then we obtain

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \delta} \cdot \frac{\partial \delta}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_i} = \delta x_i y'(z)\ ,\ i = 0, 1, ..., n;\ x_0 \equiv 1 \text{ and}$$

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = \begin{cases} \alpha \delta x_i y'(z) & \text{for } w_1, ..., w_n \\ \alpha \delta y'(z) & \text{for } w_0, \end{cases} \tag{14}$$

where $\alpha > 0$ is a coefficient representing a learning rate.

The errors of the neurons are obtained as follows. For the neurons of the m^{th} (output) layer we obtain the following expression of their errors:

$$\delta_{km} = y'(z_{km}) \delta^*_{km}\ , \tag{15}$$

where $y'(z_{km})$ is the derivative of the activation function, calculated on the value of the weighted sum z_{km} for the k^{th} neuron of the m^{th} (output) layer, δ^*_{km} - is a global error of the network calculated according to (12) for the same k^{th} neuron of the m^{th} (output) layer.

To obtain the errors for the neurons of all other layers, a sequential error backpropagation through the network from right to left (from the m^{th} layer to the m-1^{st} one, from the m-1^{st} one to the m-2^{nd} one, …, from the 2^{nd} one to the 1^{st} one) is organized. When the error is propagated from the layer j+1 to the layer j, the error of each neuron belonging to the j+1^{st} layer is being multiplied by the weight connecting the corresponding input of this neuron from the j+1^{st} layer with the corresponding output of the neuron from the j^{th} layer. For example, the error δ_{kj+1} is propagated from the neuron kj+1 (the k^{th} neuron from the j+1^{th} layer) to the lj^{th} neuron (the l^{th} neuron from the j^{th} layer) as follows: δ_{kj+1} is multiplied by the weight w_l^{kj+1}, namely by the weight corresponding to the l^{th} input of the neuron kj+1.

Let us use the following notation.

Let w_i^{kj} be the weight corresponding to the i^{th} input of the neuron kj (k^{th} neuron of the j^{th} level). Let Y_{ij} be the actual output of the i^{th} neuron from j^{th} layer (j=1,…,m). Let N_j be the number of the neurons in the j^{th} layer. By the way, it means that the neurons from the j+1^{st} layer have exactly N_j inputs. Let $x_1, ..., x_n$ be the network inputs (they are also the inputs of the neurons from the 1^{st} layer, respectively).

Then the error for any neuron excepting the neurons from the output layer is expressed as follows:

$$\delta_{kj} = y'(z_{kj}) \sum_{i=1}^{N_{j+1}} \delta_{ij+1} w_k^{ij+1} \text{ - error for the } k^{\text{th}} \text{ neuron from } j^{\text{th}} \text{ layer } (j=1,\ldots,m-1), \tag{16}$$

where N_{j+1} is the number of the neurons in the j+1st layer, $y'(z_{kj})$ is a derivative of the activation function, calculated on the value of the weighted sum z_{kj} of the k^{th} neuron from the j^{th} layer, δ_{ij+1} is the error of the i^{th} neuron from the j+1st layer (j=1,…,m-1).

Let us now move back to the MLMVN. As it was mentioned from the beginning, the MVN activation function (5) is not differentiable. It means that the formulas (15)-(16) that determine the error calculation for MLF and (14) that determine the weights correction for MLF cannot be applied for the case of MLMVN because all of them contain the derivative of the activation function. However, for the MVN-based network this is not a problem!

As it was shown above for the single neuron, the differentiability of the MVN activation function is not required for learning. Since MVN learning is reduced to the movement along the unit circle, the correction of the weights is completely determined by the neuron's error. The same property is true not only for the single MVN, but for the network (MLMVN). The errors of all the neurons from MLMVN are completely determined by the global errors of the network (12). As well as MLP learning, MLMVN learning is based on the minimization of the error functional (13). Let us generalize all considerations that we made above for the single neuron for the case of MLMVN.

To make things simpler, let us start from the simplest case, which is a network with two neurons (one hidden neuron and one output neuron) with a single input (see Fig. 4).

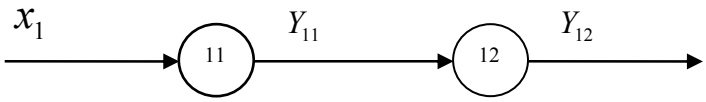


Fig. 4 The simplest MLMVN

The error on the network output is equal to $\delta = T - Y_{12}$, where T is a desired output. We need to understand how we can obtain the errors for each particular neuron, backpropagating the error δ form the right-hand side to the left-hand side.

Let us suppose that the errors for the neurons 11 and 12 are already known. We will use the learning rule (6) for the correction of the weights. We will also take into account that correction of the weights according to (6) leads to the correction of the weighted sum according to (8). Let us suppose that the neuron 11 from the 1st layer is already trained. Let us now correct the weights for the neuron 12 (the output neuron) and estimate its weighted sum:

$$\begin{aligned}
\tilde{z}_{12} &= (w_0^{12} + \frac{1}{2}\delta_{12}) + \left(w_1^{12} + \frac{1}{2}\delta_{12}\overline{(Y_{11} + \delta_{11})} \right)(Y_{11} + \delta_{11}) = \\
&= w_0^{12} + \frac{1}{2}\delta_{12} + w_1^{12}Y_{11} + w_1^{12}\delta_{11} + \frac{1}{2}\delta_{12} = \\
&= \underbrace{w_0^{12} + w_1^{12}Y_{11}}_{z_{12}} + \frac{1}{2}\delta_{12} + \frac{1}{2}\delta_{12} + w_1^{12}\delta_{11} = \\
&= z_{12} + \frac{1}{2}\delta_{12} + \frac{1}{2}\delta_{12} + w_1^{12}\delta_{11} = z_{12} + \delta_{12} + w_1^{12}\delta_{11},
\end{aligned} \tag{17}$$

where (w_0^{12}, w_1^{12}) is an initial weighting vector of the neuron 12, Y_{11} - an initial output of the neuron 11, Y_{12} - an initial output of the neuron 12, z_{12} - the weighted sum on the neuron 12 before the correction, δ_{11} - unknown error of the neuron 11 and δ_{12} - unknown error of the neuron 12.

To ensure that after the correction procedure the output of the network will be equal exactly to $Y_{12} + \delta$, it is clear from (17) that we need to satisfy the following:

$$\delta_{12} + w_1^{12}\delta_{11} = \delta. \tag{18}$$

Of course, if we will consider (18) as a formal equation, we will not get something useful. This equation has infinite number of solutions, while we have to find among them a single solution, which will correctly represent the local errors of each neuron through the global error of the network and through the errors of the neurons of the "oldest" layers.

Let us come back to the learning rule (6) for the single MVN. In this rule $\Delta W = \frac{C_m}{(n+1)}\left(\varepsilon^q - \frac{z}{|z|}\right)\overline{X}$. It contains a factor $\frac{1}{(n+1)}$ in order to distribute a contribution of the correction uniformly among all n+1 weights $w_0, w_1, \ldots, w_n$. It is easy to see that if we will omit this factor then the corrected weighted sum will not be equal to $z + \delta$ (see (4)), but to $z + (n+1)\delta$. On the other hand, since all the inputs are equitable, it will be correct and natural that during the correction procedure ΔW will be distributed among the weights uniformly, so it must be shared among the weights. This makes clear the important role of the factor $\frac{1}{(n+1)}$ in (6).

If we have not a single neuron, but a feedforward network, we have to take into account the same property. It has to be used, to implement properly a backpropagation of the error through the network. It means that if the error of a neuron on the layer j is equal to $\tilde{\delta}$, this $\tilde{\delta}$ must contain a factor equal to $\frac{1}{s_j}$, where $s_j = N_{j-1} + 1$ is the number of neurons whose outputs are connected to the inputs of the considered neuron (let us remind that N_i is the number of neurons on the layer i, and all of this neurons are connected to the considered neuron) incremented by 1 (the considered neuron itself). This ensures sharing of the error among all the neurons on which the error of the considered neuron depends. In other words, the error of each neuron is uniformly distributed among the neurons connected to it and itself. It should be mentioned that for the 1st hidden layer $s_1 = 1$ because there is no previous hidden layer, and there are no neurons, with which the error may be shared, respectively.

If we will apply this rule to our network (see Fig. 4), then we will conclude that δ_{12} must contain in its expression through a global error δ a factor $\frac{1}{2}$ (the input of the neuron 12 is connected with 1 neuron and we have to share the error with the neuron 12 itself), while δ_{11} will not contain a mentioned factor "de-facto", because $\frac{1}{1} = 1$.

It is clear now that for the neuron 12 we have

$$\delta_{12} = \frac{1}{2}\delta. \tag{19}$$

We have to take also into account that the error δ_{11} is a result of backpropagation of the error δ_{12} to the first layer (to the neuron 11). On the other hand, from (18) we obtain:

$$\delta_{11} = \frac{\delta - \delta_{12}}{w_1^{12}} = (\delta - \delta_{12})\left(w_1^{12}\right)^{-1} = \frac{(\delta - \delta_{12})\overline{w}_1^{12}}{\left|\overline{w}_1^{12}\right|^2}, \tag{20}$$

But from (19) $\delta = 2\delta_{12}$ and therefore from (20) we obtain the following expression for δ_{11}:

$$\delta_{11} = \frac{(\delta - \delta_{12})\overline{w}_1^{12}}{\left|\overline{w}_1^{12}\right|^2} = \frac{(2\delta_{12} - \delta_{12})\overline{w}_1^{12}}{\left|\overline{w}_1^{12}\right|^2} = \frac{\delta_{12}\overline{w}_1^{12}}{\left|\overline{w}_1^{12}\right|^2} = \delta_{12}\left(w_1^{12}\right)^{-1}. \qquad (21)$$

(21) is not only a formal expression for δ_{11}, but it leads us to the following important conclusion: *during a backpropagation procedure the backpropagated error must be multiplied by the inverse values of the corresponding weights.*

Let us now substitute δ_{11} and δ_{12} from (19) and (21) to (17):

$$\widetilde{z}_{12} = z_{12} + \delta_{12} + w_1^{12}\delta_{11} = z_{12} + \frac{1}{2}\delta + w_1^{12}\left(\frac{\left(w_1^{12}\right)^{-1}\delta}{2}\right) = z_{12} + \frac{\delta}{2} + \frac{\delta}{2} = z_{12} + \delta. \qquad (22)$$

So, we obtained exactly the result that is our target: $\widetilde{z}_{12} = z_{12} + \delta$.

Now it will not be difficult to generalize everything that we considered for the simplest network (Fig. 4) to the network with arbitrary number of layers and arbitrary number of neurons in each layer. We will obtain the following. The global errors of the whole network are determined by (12).
For the errors of the m^{th} (output) layer neurons:

$$\delta_{km} = \frac{1}{s_m}\delta_{km}^*, \qquad (23)$$

where km is a k^{th} neuron of the m^{th} layer; $s_m = N_{m-1} + 1$ (the number of all neurons on the previous layer (m-1, to which the error is backpropagated) incremented by 1).

For the errors of the hidden layers neurons:

$$\delta_{kj} = \frac{1}{s_j}\sum_{i=1}^{N_{j+1}} \frac{1}{\left|w_k^{ij+1}\right|^2}\delta_{ij+1}(\overline{w}_k^{ij+1}) = \frac{1}{s_j}\sum_{i=1}^{N_{j+1}}\delta_{ij+1}(w_k^{ij+1})^{-1}, \qquad (24)$$

where kj is a k^{th} neuron of the j^{th} layer (j=1,…,m-1); $s_j = N_{j-1} + 1, \quad j = 2,...,m; \; s_1 = 1$ (the number of all neurons on the previous layer (previous to j, to which the error is backpropagated) incremented by 1).

It should be mentioned that the backpropagation rule (24) is based on the same heuristic assumption as for the classical backpropagation. According to this assumption we suppose that the error of each neuron from the previous (j^{th}) layer depends on the errors of all neurons from the following (j+1$^{\text{st}}$) layer.

After calculation of the errors, the weights for all neurons of the network must be corrected. To do it, we can use the learning rule (6) applying it sequentially to all layers of the network from the first hidden layer to the output one. The rule (6) can be rewritten for this case in the following form:

Correction rule for the neurons from the 2$^{\text{nd}}$ hidden layer till the m^{th} (output) layer (k^{th} neuron of the j^{th} layer (j=2, …, m):

$$\begin{aligned} \widetilde{w}_i^{kj} &= w_i^{kj} + \frac{C_{kj}}{(n+1)}\delta_{kj}\overline{\overline{Y}}_{ij-1}, \quad i = 1,...,n \\ \widetilde{w}_0^{kj} &= w_0^{kj} + \frac{C_{kj}}{(n+1)}\delta_{kj} \end{aligned} \qquad (25)$$

Correction rule for the neurons from the 1st hidden layer:

$$\begin{aligned} \widetilde{w}_i^{k1} &= w_i^{k1} + \frac{C_{k1}}{(n+1)} \delta_{k1} \bar{x}_i, \quad i = 1,\ldots,n \\ \widetilde{w}_0^{k1} &= w_0^{k1} + \frac{C_{k1}}{(n+1)} \delta_{k1} \end{aligned} \tag{26}$$

where C_{kj} is a learning rate for the k^{th} neuron of j^{th} layer and n is the number of the corresponding neuron inputs.

On the other hand we have to take into account the following consideration. For the output layer neurons we have the exact errors calculated according to (12), while for all the hidden neurons the errors are obtained according to the heuristic rule. This may cause a situation, where either the weighted sum for the hidden neurons (more exactly, the absolute value of the weighted sum) may become a not-smooth function with dramatically high jumps or the hidden neuron output will be close to some constant with very small variations around it. In both cases thousands and even the hundreds of thousands of additional steps for the weights adjustment will be required. To avoid this situation we can use for the hidden neurons the learning rule (8)-(9) instead of the rule (6) and therefore to normalize ΔW for the hidden neurons by $|z|$ every time, when the weights are being corrected. This will make the absolute value of the weighted sum for the hidden neurons (considered as a function of the weights) more smooth. This also can avoid the concentration of the hidden neurons output in some very narrow interval. On the other hand, the factor $1/|z|$ in (9) can be considered as a variable part of the learning rate. While used, it provides the adaptation of ΔW on each step of learning. At the same time it is not reasonable to use the rule (8)-(9) for the output layer. The exact errors and the exact desired outputs for the output neurons are known. On the other hand, since these errors are shared among all neurons of the network according to (23)-(24), there is no reason to normalize by $1/|z|$ the errors of the output neurons. The absolute value of the output neurons weighted sums belongs to the narrow ring, which includes the unit circle (see Fig. 3), without this normalization. We will return to these features of the learning algorithm in the Section IV, where several examples will be considered. So the weights of the output neurons may be corrected by the rule (6). In this way we are coming to the following modification of the correction rule (25)-(26).

Correction rule for the neurons from the m^{th} (output) layer (k^{th} neuron of m^{th} layer):

$$\begin{aligned} \widetilde{w}_i^{kj} &= w_i^{km} + \frac{C_{km}}{(n+1)} \delta_{km} \overline{\overline{Y}}_{im-1}, \quad i = 1,\ldots,n \\ \widetilde{w}_0^{km} &= w_0^{km} + \frac{C_{km}}{(n+1)} \delta_{km} \end{aligned} \tag{27}$$

Correction rule for the neurons from the 2nd till m-1st layer (k^{th} neuron of the j^{th} layer (j=2, …, m-1):

$$\begin{aligned} \widetilde{w}_i^{kj} &= w_i^{kj} + \frac{C_{kj}}{(n+1)\,|\,z_{kj}\,|} \delta_{kj} \bar{x}_i, \quad i = 1,\ldots,n \\ \widetilde{w}_0^{kj} &= w_0^{kj} + \frac{C_{kj}}{(n+1)\,|\,z_{kj}\,|} \delta_{kj} \end{aligned} \tag{28}$$

Correction rule for the neurons from the 1st hidden layer:

$$\begin{aligned} \tilde{w}_i^{k1} &= w_i^{k1} + \frac{C_{k1}}{(n+1)\,|\,z_{kj}\,|}\delta_{k1}\bar{x}_i, \quad i = 1,\ldots,n \\ \tilde{w}_0^{k1} &= w_0^{k1} + \frac{C_{k1}}{(n+1)\,|\,z_{kj}\,|}\delta_{k1} \end{aligned} \tag{29}$$

We cannot exclude a priori the case, when a mapping to be implemented by the network will be presented by a so complicated nonlinear and non-smooth function, when the adaptation of the output neurons weights will require their normalization by $1/|z|$. However, we did not find such a function testing the network using different benchmarks. To use this normalization, it is only necessary to use (28) instead of (27) for the neurons from the output layer.

All the considerations above lead us to the conclusion that the errors of the output layer neurons and therefore the global errors of the network will descent after correction of the weights according to the rules (27)-(29). It means that the error functional (13) will also descent step by step. In general the learning process should continue until the following condition is satisfied

$$\sum_k (\delta_{km}^*)^2(W) \le \varepsilon\,, \tag{30}$$

where ε determines the precision of learning. In particular, it should be the case $\varepsilon = 0$, and (30) will be transformed to $\sum_k (\delta_{km}^*)^2(W) = 0$.

It is necessary to make one remark regarding the error backpropagation. The equation (24) for the error of the neuron belonging to any hidden layer was obtained from equation (21) representing the case of the simplest network by its direct generalization and taking into account a general heuristic view on the error backpropagation. This view says that the error of each neuron from the layer j+1 must be weighted by the k^{th} weight of this neuron, while backpropagated to the k^{th} neuron from the layer j.

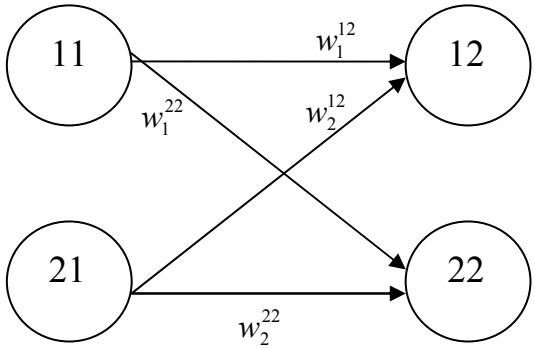


Fig. 5 A fragment of MLVN

On the other hand this view does not take into account that the error of each neuron form the j+1st layer contains the errors of all neurons from the layer j. To clarify this, let us consider the following simple network containing two layers and two neurons on each of the layers (see Fig. 5).

Let us suppose without loss of generality that the weights are being corrected according to (25)-(26). Let us represent the weighted sums of the neurons 12 and 22 after the correction of the weights.

$$z_{12} = w_0^{12} + w_1^{12}(Y_{11} + \delta_{11}) + w_2^{12}(Y_{21} + \delta_{21}) + \frac{\delta_{12}}{3} =$$

$$= w_0^{12} + w_1^{12}Y_{11} + w_1^{12}\delta_{11} + w_2^{12}Y_{21} + w_2^{12}\delta_{21} + \frac{\delta_{12}}{3} =$$

$$= \underbrace{w_0^{12} + w_1^{12}Y_{11} + w_2^{12}Y_{21}}_{z_{12}} + w_1^{12}\delta_{11} + w_2^{12}\delta_{21} + \frac{\delta_{12}}{3} =$$

$$= z_{12} + w_1^{12}\delta_{11} + w_2^{12}\delta_{21} + \frac{\delta_{12}}{3}$$

and

$$z_{22} = w_0^{22} + w_1^{22}(Y_{11} + \delta_{11}) + w_2^{22}(Y_{21} + \delta_{21}) + \frac{\delta_{22}}{3} =$$

$$= w_0^{22} + w_1^{22}Y_{11} + w_1^{22}\delta_{11} + w_2^{22}Y_{21} + w_2^{22}\delta_{21} + \frac{\delta_{22}}{3} =$$

$$= \underbrace{w_0^{22} + w_1^{22}Y_{11} + w_2^{22}Y_{21}}_{z_{22}} + w_1^{22}\delta_{11} + w_2^{22}\delta_{21} + \frac{\delta_{22}}{3} =$$

$$= z_{22} + w_1^{22}\delta_{11} + w_2^{22}\delta_{21} + \frac{\delta_{22}}{3}$$

It follows directly from the last two equations that

$$w_1^{12}\delta_{11} + w_2^{12}\delta_{21} + \frac{\delta_{12}}{3} = \delta_{12}$$

$$w_1^{22}\delta_{11} + w_2^{22}\delta_{21} + \frac{\delta_{22}}{3} = \delta_{22}$$

and, respectively,

$$w_1^{12}\delta_{11} + w_2^{12}\delta_{21} = \frac{2}{3}\delta_{12}$$

$$w_1^{22}\delta_{11} + w_2^{22}\delta_{21} = \frac{2}{3}\delta_{22}.$$

Formally the last system of equations can be considered as a system of linear algebraic equations with respect to δ_{11} and δ_{21}. But we cannot always consider it in this way. It is good, if such a system will contain the same number of equations as the number of unknowns. It will mean that the system has a unique solution, while its determinant is not equal to zero and rank of its matrix its equal to the one of the extended matrix. In this case it is very easy to solve the system using, for example, a classical Kramer's rule. But the first condition will hold only for the networks that contain the same number of neurons on the neighboring layers. So an exact solution of the system may be obtained for the case, when the neighboring layers contain the same number of neurons.

If not, we can assume that the neurons from the layer *j* equitably and independently contribute to the errors of the neurons from the layer *j*+1. This heuristic assumption leads us to the conclusion that the errors of the neurons from the layer *j*+1 do not depend on each other. This means that in our example errors δ_{11} and δ_{21} do not depend on each other and may be calculated separately from the assumption that their

contribution to the errors δ_{12} and δ_{22} of the neurons 12 and 22 is equal that means $w_1^{12}\delta_{11} = w_2^{12}\delta_{21} = \frac{1}{3}\delta_{12}$ and $w_1^{22}\delta_{11} = w_2^{22}\delta_{21} = \frac{1}{3}\delta_{22}$. In this way we obtain the following systems of equations for δ_{11} and δ_{21}, respectively:

$$w_1^{12}\delta_{11} = \frac{1}{3}\delta_{12}$$

$$w_1^{22}\delta_{11} = \frac{1}{3}\delta_{22}$$

and

$$w_2^{12}\delta_{21} = \frac{1}{3}\delta_{12}$$

$$w_2^{22}\delta_{21} = \frac{1}{3}\delta_{22}.$$

It is easy to obtain from the last two systems that

$$\delta_{11} = \frac{1}{3}\left(\delta_{12} + \delta_{22}\right)\left(w_1^{12} + w_1^{22}\right)^{-1} = \frac{1}{3}\left(\delta_{12} + \delta_{22}\right)\frac{\overline{\left(w_1^{12} + w_1^{22}\right)}}{\left|\left(w_1^{12} + w_1^{22}\right)\right|^2}.$$

and

$$\delta_{21} = \frac{1}{3}\left(\delta_{12} + \delta_{22}\right)\left(w_2^{12} + w_2^{22}\right)^{-1} = \frac{1}{3}\left(\delta_{12} + \delta_{22}\right)\frac{\overline{\left(w_2^{12} + w_2^{22}\right)}}{\left|\left(w_2^{12} + w_2^{22}\right)\right|^2}$$

Let us generalize the considered case of the network containing two layers and two neurons on each of them for the case of the *m*-layered network. The errors for the neurons of output layer (m^{th} layer) are still defined by (23). But for the neurons from the hidden layers 1, …, *m*-1 we obtain the following.

$$\sum_{k=1}^{N_j} w_k^{ij+1}\delta_{kj} = \delta_{ij+1}\ ,\ i = 1,...,N_{j+1}\,. \tag{31}$$

(31) is the system of *k* equations with respect to δ_{kj}. We omit a factor $\frac{1}{s_{j+1}}$ in the right hand part of (31), because when *j*+1=*m* this factor is already included to the corresponding δ (see (23)), while for all other layers we will include the corresponding factor to the equation that will determine the backpropagation, like it was done in (24). Thus we obtain the following formula for the error δ_{kj} of any neuron belonging to any hidden layer from 1 to *m*-1:

$$\delta_{kj} = \frac{1}{s_j}\left(\sum_{t=1}^{N_{j+1}} w_k^{tj+1}\right)^{-1}\sum_{i=1}^{N_{j+1}}\delta_{ij+1}, \tag{32}$$

where N_j is the number of neurons on the layer *j*, *kj* is a k^{th} neuron of the j^{th} layer (*j*=1,…,*m*-1); $s_j = N_{j-1} + 1,\quad j = 2,...,m-1;\ s_1 = 1$ (the number of all neurons on the previous layer (previous to *j*, to which the error is backpropagated) incremented by 1).

The backpropagation rule (32) is an alternative to the rule (24). Both of them are based on different heuristic assumptions about mutual dependence and influence of the errors corresponding to the neurons

from the neighboring layers. In particular, for the network consisting of a single hidden layer and the output layer (32) coincides with (24). The experimental testing does not show that one of this backpropagation rules is better than the another one. Both of them show in average the same efficiency. It means only that both heuristic assumptions about the mutual dependence and mutual influence of the errors for the neurons from the neighboring layers are not contradictory.

IV. Simulation Results

Simulating MLMVN we were trying to operate with a network with a minimal possible number of the hidden layers and with a minimal possible number of the neurons. So the simulation results that we will consider now, were obtained using the simplest network structure $n \rightarrow S \rightarrow 1$ (n inputs, S neurons on the 1st hidden layer and 1 neuron on the output layer). It should be mentioned that adding additional hidden layers we did not get any significant improvement. So the efficiency of the backpropagation algorithm (23)-(24) (or (23)-(32), which is the same for the network with a single hidden layer) and of the learning algorithm based on the rules (27)-(29) has been tested by the experiments with the three standard and popular benchmarks: parity n, two spirals and "sonar". For two spirals and "sonar" benchmarks we downloaded data from the CMU benchmark collection[3]

The neural network was implemented using a software simulator that was run on the PC with a Pentium III 600 MHz processor. The real and imaginary parts of the starting weights were taken as random numbers from the interval [0, 1] for all experiments.

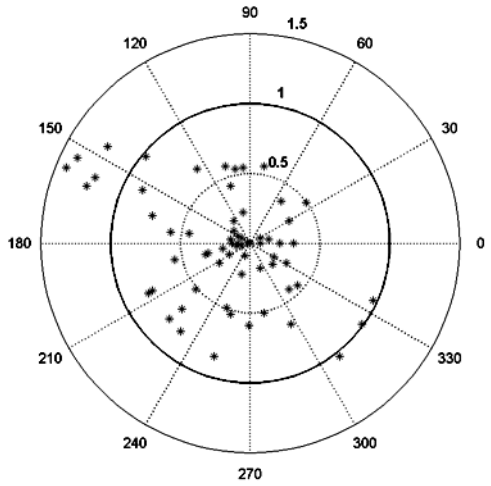
Table 1 Implementation of the Parity n function ($n = 2,3,...,9$) on the MLMVN $S \rightarrow 1$ ($S \le n$)

Function	Configuration of the network	Number of epochs (the median value of 5 independent experiments is taken)	Processing time on P-III 600 MHz
Parity 3	3→2→1	57	2 seconds
Parity 4	4→2→1	109	3 seconds
Parity 5	5→3→1	2536	15 seconds
Parity 6	6→4→1	7235	30 seconds
Parity 7	7→4→1	26243	2 min. 50 sec.
Parity 8	8→7→1	312541	122 min.
Parity 8	8→6→1	1085677	215 min.
Parity 9	9→7→1	24234	20 min.

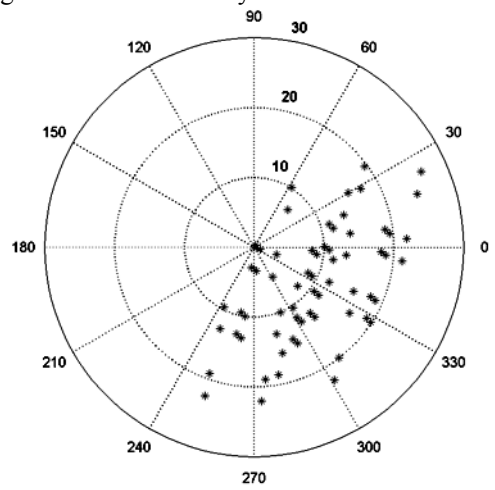
The parity n functions $(3 \le n \le 9)$ were trained completely (see Table 1) using the network $n \rightarrow S \rightarrow 1$, where $S < n$. It should be mentioned that parity 9 and parity 8 functions were implemented using only 7 and 6 hidden neurons, respectively. These results show advantages of MLMVN in comparison with the traditional solutions. Indeed, it was reported in [29] that the most optimistic estimation for the number of the hidden neurons for the implementation of the n bit parity function using one hidden layer is $\sqrt{n}$, while the realistic estimation is Ό(n). In [30] it was shown up to $n = 4$, that the minimum size of the hidden layer required to solve the N-bit parity is n. Using a special learning algorithm the parity 7 function was implemented using a 7-4-1 MLF in [14]. Using a modular network architecture the parity 8 function was implemented in [13]. We did use neither some special architecture nor some specific learning strategy. Moreover, no adaptation of the constant part of the learning rate was used in all our experiments not only with the parity functions, but with all the benchmarks, i.e. all $C_{kj} = 1$ in (27)-(29).

Let us return to the role of the factor $\frac{1}{z_{kj}}$ in the correction rule (28)-(29) for the hidden neurons and to absence of this factor in the correction rule (27) for the output neurons. Without loss of generality we can consider the example of parity 6 function. This function is implemented using the network 6→4→1 (four hidden neurons and one output neuron), as it is shown in Table 1. Training this network, we applied the rule (25)-(26) and (27)-(29) starting from the same randomly chosen weighting vector. With the rule (27)-(29) we get the convergence after 5874 epochs. The distribution of the weighted sums after the convergence is shown for the output neuron and one of the hidden neurons on the Fig. 6a and Fig. 6b, respectively.

[3] http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu (a number of references to the different experimental results and the summary of these results can be also found at the same directory)



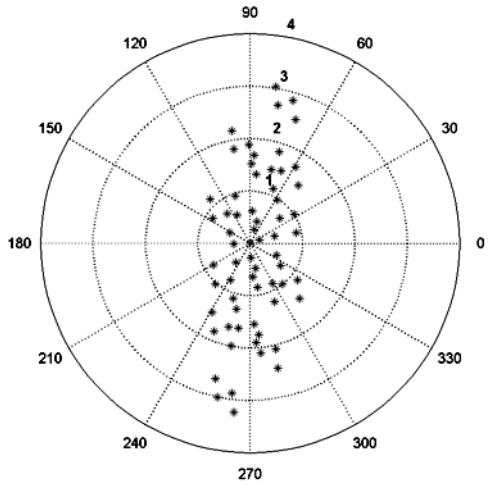
(a) output neuron after convergence of the learning algorithm. The weighted sums belong to the ring, whose longer border is limited by the circle of a radius 1.5



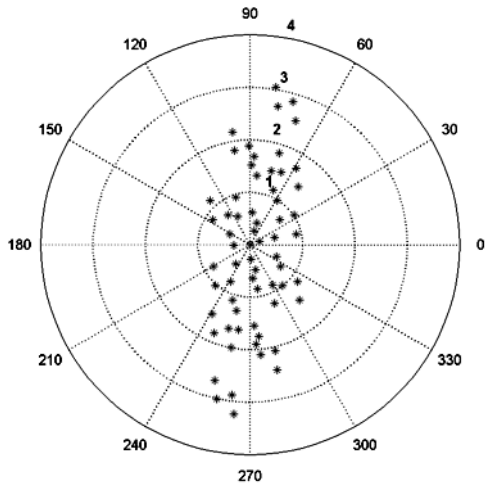
(b) the first neuron from the hidden layer after convergence of the learning algorithm (the rule (29) was used, 5874 epochs). The weighted sums are distributed in such a way that the neuron's output belongs to the wide interval approximately equal to the half of the unit circle

Fig. 6 Distribution of the weighted sums for the output neuron and one of the hidden neurons for the network 6→4→1 implementing the parity 6 function after the convergence. 64 weighted sums are shown as points on the complex plane

It is clearly visible from the Fig. 6a that the weighted sums for the output neuron belong to the narrow ring, whose longer border is limited by the circle of a radius 1.5. For the first hidden neuron (Fig. 6b) the weighted sums are distributed in such a way that the neuron's output belongs to the wide interval approximately equal to the half of the unit circle. Actually, exactly this is our target: we introduce the hidden layer in order to replace a Boolean non linearly separable function (parity N) by the multiple-valued function with a binary output, however with informally multiple-valued inputs. With the rule (25)-(26), which does not contain factor $\frac{1}{z_{kj}}$ as a variable part of the learning rate for the hidden neurons, we can't get the convergence even after 20000 epochs with the same starting weighting vector.



(a) the first neuron from the hidden layer after 10000 epochs without convergence using the rule (26). The weighted sums are distributed within a narrow domain and the neuron's output also belongs to a quite limited domain



(b) the first neuron from the hidden layer after 20000 epochs without convergence using the rule (26). Almost no changes in comparison with the picture (b) after additional 10000 epochs.

Fig. 7 Distribution of the weighted sums for the output neuron and one of the hidden neurons for the network 6→4→1 implementing the parity 6 function without a self-adaptation of the learning rate and without the convergence.
64 weighted sums are shown as points on the complex plane

Fig. 7a shows that after 10000 epochs the weighted sums for the first hidden neuron are concentrated within a narrow domain, which looks as a stripe. As a result, the neuron's output belongs to a very limited domain. It is not binary, but it is not enough multiple-valued. Projections of too many weighted sums on the unit circle coincide with each other. There are practically no changes after additional 10000 epochs (Fig. 7b). This example is typical. It shows that the use of the factor $\frac{1}{z_{kj}}$ as a variable part of the learning rate is very important for the hidden neurons. It makes the learning rate self-adaptive to the particular case.

The two spirals problem is a well known classification problem, where the two spirals points to (see Fig. 8) must be classified as belonging to the 1st or to the 2nd spiral. The two spirals data also was trained completely without the errors using the networks n→40→1 (800123 epochs is a median of 11 experiments) and n→30→1 (1590005 epochs is a median of 11 experiments). For example, for MLF with an adapted learning algorithm there is the result with about 4% errors for the network n→40→1 and with about 14% errors for the network n→30→1 after about 150000 learning epochs [14]. This result is reported as one of the best ones. It should be mentioned that after the same number of learning epochs the MLMVN shows not more than 2% of errors for the network n→40→1 and not more than 6% errors for the network n→30→1.

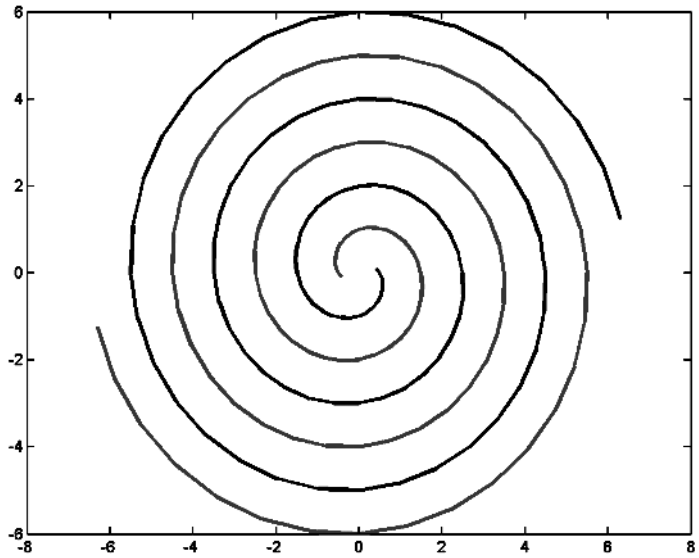


Fig. 8 Two spirals

On the other hand the two spirals data set containing 194 points was separated into training (98 points) and testing (96 points) subsets (the first two points were assigned to the training subset, while the next two points were assigned to the testing subset, etc). After training using the first subset, the prediction capability was tested using the second one. For this experiment we used the networks containing 26, 28, …, 40 hidden neurons on the single hidden layer. The networks with the larger number of the hidden neurons are trained much faster, but the prediction results are approximately the same for all the networks. Table II shows, how the number of training epochs decreases with increasing of the number of the hidden neurons. The median, minimal and maximal numbers of the epochs for nine experiments with each network are shown.

Table II Correspondence between the number of hidden neurons and the number of training epochs

# of epochs (median for 9 independent experiments)	1298406	1015315	516807	218004	98732	84338	31145	19781
# of epochs (min and max for 9 independent experiments)	1010877-1412355	881017-1414385	375289-1467516	157278-587619	60115-159784	51914-138137	18385-130853	3866-129291
# of neurons on the hidden layer	26	28	30	32	34	36	38	40

The prediction results for the two spiral benchmark are stable for all the considered networks from 2→26→1 till 2→40→1. The prediction rate for the two spirals testing set obtained as a result of nine independent experiments with each network is 68-72%. These results that obtained using a smaller and simpler network are comparative with the best known results (70-74.5%) [12]. It should be also mentioned that some additional series of the experiments show that it is possible to obtain even a better prediction rate (74-75%). This result appeared very randomly (2 times per 100 independent experiments with the 2→40→1 network), so it can not be recognized as a stable result, but it is an additional argument for the high potential capabilities of MLMVN.

The same experiment was performed for the "sonar" data set, but using the simplest possible network 60→2→1 (the "sonar" problem initially depends on 60 input parameters) with only two neurons in the hidden layer. The "sonar" data set contains 208 samples. 104 of them are recommended to be used for training and another 104 for testing, respectively. The training process requires 400-2400 epochs and a few seconds, respectively. This statistics is based on 50 independent experiments. The prediction results are also very stable. The predictions rate from the same experiments is 88-93%. This result is comparative with the best known result for the fuzzy kernel perceptron (94%) [12] and SVM (89.5%), however our result is obtained using the smallest possible network. On the other hand the whole "sonar" data set was trained completely without the errors using the same simplest network 60→2→1. This training process requires from 817 till 3700 epochs according to the results of 50 experiments.

V. CONCLUSIONS

In this paper a new feedforward neural network was proposed. A multi-layered neural network based on multi-valued neurons (MLMVN) is a network with a traditional feedforward architecture and a multi-valued neuron (MVN) as a basic one. A single MVN has the higher functionality than the traditional neurons. Its learning, which is reduced to the movement along the unit circle, does not require the differentiability of the activation function. The learning process is completely determined by the simple linear rule. These properties make MLMVN more powerful than traditional feedforward networks. The backpropagation learning algorithm for MLMVN that was developed in the paper also does not require the differentiability of the activation function. Being similar to the classical backpropagation algorithm, it has some important differences that make it more stable and less heuristic. The main differences are: sharing of the network error among all the neurons of the network and a self adaptation of the learning rate for the hidden neurons.

These advantages of the MLMVN make it possible to solve different prediction, recognition and classification problems using a smaller network and a simpler learning algorithm than ones traditionally used. MLMVN shows a good performance in convergence speed. Its testing using several traditional benchmarks shows better or comparative to the best known results.

REFERENCES

[1] D.E. Rumelhart and J.L. McClelland *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, 1986.

[2] A.N. Kolmogorov "On the Representation of Continuous Functions of many Variables by Superposition of Continuous Functions and Addition", Doklady Akademii Nauk SSSR, vol. 114, pp. 953-956, 1957 (in Russian).

[3] R. Hecht-Nielsen "Kolmogorov Mapping Neural Network Existence Theorem". *Proc. Of the 1st IEEE International Conference on Neural Networks*, vol. 3, pp. 11-13, IEEE Computer Society Press, 1988.

[4] K.I. Funahashi "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, vol. 2, pp. 183-192, 1989,

[5] R. Hecht-Nielsen *Neurocomputing*. Adison Wesley, New York, 1990.

[6] K. Hornik, M. Stinchcombe, H. White. "Multilayer Feedforward Neural Networks are Universal Approximators", *Neural Networks*, vol. 2, pp. 259-366, 1989.

[7] H. Siegelman, E. Sontag. "Neural Nets are Universal Computing Devices". *Research Report SYCON-91-08. Rutgers Center for Systems and Control. Rutgers University*, 1991.

[8] S. Haykin "*Neural Networks: A Comprehensive Foundation (2nd Edition)*", Prentice Hall, 1998.

[9] T.M. Cover "Geometrical and Statistical Properties of systems of Linear Inequalities with application in pattern recognition" IEEE Trans. Electron. Comput., vol. EC-14, pp. 326-334, Mar. 1965.

[10] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[11] K.-R.Müller, S. Mika, G. Rätsch, K. Tsuda and B. Shölkopf "An Introduction to Kernel-based Learning Algorithms" *IEEE Trans. Neural Networks*, vol. 12, pp. 181-201, Jan. 2001.

[12] J.-H.Chen and C.-S. Chen "Fuzzy Kernel Perceptron" *IEEE Trans. Neural Networks*, vol. 13, pp. 1364-1373, Nov. 2002.

[13] L. Franco and S.A. Cannas "Generalization Properties of Modular Networks: Implementing the Parity Function", *IEEE Trans. Neural Networks*, vol. 12, pp. 1306-1313, Nov. 2001.

[14] E. Mizutani and S.E. Dreyfus "MLP's hidden-node saturations and insensitivity to initial weights in two classification benchmark problems: parity and two-spirals", Proc. of the 2002 International Joint Conference on Neural Networks (IJCNN'02), May 12-17, 2002, pp. 2831-2836.

[15] N.N. Aizenberg and I.N. Aizenberg "CNN Based on Multi-Valued Neuron as a Model of Associative Memory for Gray-Scale Images", *Proceedings of the Second IEEE Int. Workshop on Cellular Neural Networks and their Applications,* Technical University Munich, Germany October 14-16, 1992, pp.36-41.

[16] N.N. Aizenberg, Yu. L. Ivaskiv and D.A. Pospelov "About one Generalization of the Threshold Function" Doklady Akademii Nauk SSSR ("The Reports of the Academy of Sciences of the USSR"), vol. 196, No 6, pp. 1287-1290, 1971 (in Russian).

[17] N.N. Aizenberg and Yu.L. Ivaskiv *Multiple-Valued Threshold Logic*, Naukova Dumka Publisher House, Kiev, 1977 (in Russian).

[18] I. Aizenberg, N. Aizenberg and J.Vandewalle "*Multi-valued and universal binary neurons: theory, learning, applications*", Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[19] S. Jankowski, A. Lozowski and J.M. Zurada "Complex-Valued Multistate Neural Associative Memory", *IEEE Trans. Neural Networks*, vol. 7, 1996, pp. 1491-1496.
[20] H. Aoki and Y. Kosugi "An Image Storage System Using Complex-Valued Associative Memory", *Proc. of the 15th International Conference on Pattern Recognition*, Barcelona, 2000, IEEE Computer Society Press, vol. 2, pp. 626-629.
[21] M. K. Muezzinoglu, C. Guzelis and J. M. Zurada, "A New Design Method for the Complex-Valued Multistate Hopfield Associative Memory", *IEEE Trans. Neural Networks*, vol. 14, No 4, 2003, pp. 891-899.
[22] H. Aoki, E. Watanabe, A. Nagata and Y. Kosugi "Rotation-Invariant Image Association for Endoscopic Positional Identification Using Complex-Valued Associative Memories", *Bio-inspired Applications of Connectionism, Lecture Notes in Computer Science*, (J. Mira, A. Prieto - Eds.), vol. 2085 Springer 2001, pp. 369-374.
[23] I. Aizenberg, E. Myasnikova, M. Samsonova and J. Reinitz "Temporal Classification of Drosophila Segmentation Gene Expression Patterns by the Multi-Valued Neural Recognition Method", *Journal of Mathematical Biosciences* (Elsevier), Vol.176 (1), pp. 145-159, 2002.
[24] I. Aizenberg, T. Bregin, C. Butakoff, V.Karnaukhov, N. Merzlyakov and O. Milukova "Type of Blur and Blur Parameters Identification Using Neural Network and Its Application to Image Restoration". *Lecture Notes in Computer Science*, (J.R. Dorronsoro – Ed.) vol. 2415, Springer, pp. 1231-1236, 2002.
[25] H. Leung and S. Haykin "The Complex Backpropagation Algorithm", IEEE Trans. Signal Processing, vol.39, pp. 2101-2104, Sept., 1991.
[26] G.M. Georgiou and C. Koutsougeras "Complex Domain Backpropagation" *IEEE Trans. Circuits and Systems CAS-II*, vol. 39, pp. 330-334, May, 1992.
[27] T. Nitta "An extension of the Backpropagation Algorithm to Complex Numbers", *Neural Networks*, vol. 10, no. 8, pp. 1391-1415, August, 1997.
[28] A. Hirose (Ed.) "Complex Valued Neural Networks. Theories and Applications", World Scientific, Singapore, 2003.
[29] R. Impagliazzo, R. Paturi, and M. E. Saks, "Size-depth tradeoffs for threshold circuits," *SIAM J. Comput.*, vol. 26, no. 3, pp. 693–707, 1997.
[30] H. Fung and L. K. Li, "Minimal feedforward parity networks using threshold gates," *Neural Comput.*, vol. 13, pp. 319–326, 2001.

Acknowledgement

The first author gladly acknowledges the support of the Collaborative Research Center 531, where he was an Invited Researcher, February through April 2004.

E-mail address of the authors:
Igor Aizenberg igora@hotbox.ru
Claudio Moraga claudio.moraga@udo.edu