MULTILAYER NEURAL NETWORK BASED ON MULTI-VALUED NEURONS (MLMVN) APPLIED TO CLASSIFICATION OF MICRORARRAY GENE EXPRESSION DATA

Igor Aizenberg¹, Pekka Ruusuvuori², Olli Yli-Harja² and Jaakko T. Astola²

¹ Texas A&M University-Texarkana
Department of Computer and Information Sciences
P.O. Box 5518, 2600 N. Robison Rd. Texarkana, Texas 75505 USA,
²Institute of Signal Processing, Tampere University of Technology,
P.O. Box 553, FI-33101 Tampere, Finland,
igor.aizenberg@gmail.com, pekka.ruusuvuori@tut.fi, olli.yli-harja@tut.fi, jaakko.astola@tut.fi

ABSTRACT
Classification of microarray gene expression data is a common problem in bioinformatics. Classification problems with more than two output classes require more attention than the normal binary classification. Here we apply a multilayer neural network based on multi-valued neurons (MLMVN) to the multiclass classification of microarray gene expression data. Two four-class test cases are considered. The results show that MLMVN can be used for classifying microarray data accurately.

1. INTRODUCTION
A multilayer neural network based on multi-valued neurons (MLMVN) has been introduced in [1] and then it has been developed in [2]. This network and its backpropagation learning is comprehensively observed and developed further in [3]. The MLMVN consists of multi-valued neurons (MVN). That is a neuron with complex-valued weights and an activation function, defined as a function of the argument of a weighted sum. MVN is based on the principles of multiple-valued threshold logic over the field of complex numbers. A comprehensive observation of the discrete-valued MVN, its properties and learning is presented in [4]. A continuous-valued MVN and its learning are considered in [1]-[3]. The most important properties of MVN are: the complex-valued weights, inputs and output coded by the \( k \)-th roots of unity (a discrete-valued MVN) or lying on the unit circle (a continuous-valued MVN), and an activation function, which maps the complex plane into the unit circle. Both MVN and MLMVN learning are reduced to the movement along the unit circle. The most important property and advantage of their learning is that it does not require differentiability of the activation function. The MVN learning algorithm [3], [4] is based on a simple linear error correction rule. This learning rule is generalized for the MLMVN as a backpropagation learning algorithm [3], which is simpler and more efficient than traditional backpropagation learning. MLMVN outperforms a classical multilayer feedforward network (usually referred to as a multilayer perceptron - MLP) and different kernel-based networks in the terms of learning speed, network complexity, and classification/prediction rate tested for such popular benchmark problems as the parity \( n \), the two spirals, the sonar, and the Mackey-Glass time series prediction [1]-[3]. These properties of MLMVN show that it is more flexible and adapts faster in comparison with other solutions based on neural networks. It is important to note that since MLMVN (as well as a single MVN) implements such mappings that are described by multiple-valued (up to infinite-valued) functions, it can be an efficient mean for solving the multiclass classification problems.

In this paper we apply MLMVN to the multiclass classification of microarray gene expression data. After presenting the basic properties of MLMVN and its backpropagation learning algorithm we will consider two four-class test cases of microarray gene expression data classification. The classification results of MLMVN classifier are compared to those given by nearest neighbor classifiers with different number of neighbors.

2. MULTILAYER NEURAL NETWORK BASED ON MULTI-VALUED NEURONS

2.1. Multi-valued neuron (MVN)
MVN [4] is a neural element based on the principles of multiple-valued threshold logic over the field of complex numbers. A single MVN performs a mapping between \( n \) inputs and a single output. For the discrete-valued MVN this mapping is described by a multiple-valued \( (k\)-valued) function of \( n \) variables \( f(x_1, ..., x_n) \) with \( n+1 \) complex-valued weights as parameters:

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

where \( X = (x_1, ..., x_n) \) is a vector of inputs (a pattern vector) and \( W = (w_0, w_1, ..., w_n) \) is a weighting vector. The inputs and output of the discrete-valued MVN are the \( k \)-th roots of unity: \( e^{j\pi / K} \), \( j = 0, ..., k-1 \),
where $i$ is an imaginary unity. $P$ is the activation function of the neuron:

$$P(z) = \exp(2\pi j/k), \text{ if } 2\pi j/k \leq \arg z < 2\pi(j+1)/k, \quad (2)$$

where $j=0,\ldots,k-1$ are the values of $k$-valued logic, $z = w_0 + w_1 x_1 + \ldots + w_n x_n$ is a weighted sum, $\arg z$ is the argument of the complex number $z$. Function (2) divides a complex plane onto $k$ equal sectors and maps the whole complex plane into a set of $k$\textsuperscript{th} roots of unity (see Figure 1).

The activation function (2) is discrete. It has been recently proposed in [1]-[3], to modify the function (2) in order to generalize it for the continuous case in the following way. If $k \rightarrow \infty$ in (2) then the angle value of the sector (see Figure 1) tends to zero. Hence, the function (2) is transformed in this case as follows:

$$P(z) = \exp(i \arg z) = e^{i\arg z} = \frac{z}{|z|}, \quad (3)$$

where $\arg z$ is a main value of the argument of the complex number $z$ and $|z|$ is its modulo. Thus the activation function (3) determines a continuous-valued MVN. Inputs and output of this neuron are lying on the unit circle, but since they are continuous, this case corresponds to the infinite-valued logic.

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

It is also possible to consider a hybrid MVN (either discrete inputs $\rightarrow$ continuous output or continuous inputs $\rightarrow$ discrete output). We will use in this paper exactly MVN with the continuous inputs and a discrete output. It is important that MVN learning does not depend on type of the neuron. It is reduced to the movement along the unit circle. This movement does not require a derivative of the activation function. The learning process is based on the following error correction rule [3], [4]

$$W_{r+1} = W_r + \frac{C_r}{(n+1)}(T - Y)X, \quad (4)$$

where $X$ is an input vector, $n$ is a number of neuron's inputs, $X$ is a vector with the components complex conjugated to the components of vector $X$, $r$ is the number of iteration, $W_r$ is a current weighting vector, $W_{r+1}$ is a weighting vector after correction, $T$ is a desired neuron's output, $Y$ is an actual neuron's output, and $C_r$ is a learning rate.

### 2.2. MVN-based Multilayer Feedforward Neural Network (MLMVN)

A multilayer architecture of the network with a feedforward dataflow through nodes that requires full connection between consecutive layers and an idea of a backpropagation learning algorithm was proposed in [5] by D. E. Rumelhart and J. L. McClelland. A classical example of such a network is a multilayer perceptron (MLP) [5], [6]. Its learning is based on the algorithm of error backpropagation. The error is being sequentially distributed form the "right hand" layers to the "left hand" ones. A crucial point of the MLP backpropagation is that the error of each neuron of the network is proportional to the derivative of the activation function. Usually MLP is based on the neurons with the sigmoid activation function [6].

However, it is possible to use different neurons as the basic ones for a network with the feedforward architecture. A multilayer feedforward neural network based on multi-valued neurons (MLMVN) has been recently proposed in [1]-[3]. This network has at least two principal advantages in comparison with an MLP: higher functionality (an MLMVN with the smaller number of hidden neurons outperforms an MLP with the larger number of hidden neurons [1]-[3]) and simplicity of learning (MLMVN learning does not require differentiability of the activation function).

Let us consider $m$-layer MLMVN with $n$ inputs ($m$-1 hidden layers and one output layer (the $m$\textsuperscript{th} one) based on the MVN with the continuous inputs and a discrete output. Let $w_j^i$ be the weight corresponding to the $i$\textsuperscript{th} input of the $j$\textsuperscript{th} neuron ($k$\textsuperscript{th} neuron of the $j$\textsuperscript{th} layer), $Y_j^i$ be the actual output of the $j$\textsuperscript{th} neuron from the $j$\textsuperscript{th} layer ($j=1,\ldots,m$), and $N_j$ be the number of the neurons in the $j$\textsuperscript{th} layer. It means that the neurons from the $j$\textsuperscript{th} one to the $m$\textsuperscript{th} one have exactly $N_j$ inputs. Let $x_{1j},\ldots,x_{N_j}$ be the network inputs. The backpropagation learning algorithm for the MLMVN is described as follows [3].

The global error of the network taken from the $k$\textsuperscript{th} neuron of the $m$\textsuperscript{th} (output) layer is calculated as follows:

$$\delta_{km}^o = T_{km} - Y_{km}. \quad (5)$$

The backpropagation of the global errors $\delta_{km}^o$ through the network is used (from the $m$\textsuperscript{th} (output) layer to the $m$\textsuperscript{th}-1 one, from the $m$\textsuperscript{th}-1 one to the $m$\textsuperscript{th}-2 one, $\ldots$, from the $m$\textsuperscript{th}-$i$ one to the $m$\textsuperscript{th}-($i+1$) one) in order to express the error of each neuron $\delta_{ij}, j = 1,\ldots,m; i = 1,\ldots,N_j$ by means of the global errors $\delta_{km}^o$ of the entire network.

The errors of the $m$\textsuperscript{th} (output) layer neurons are:

$$\delta_{km}^i = \frac{1}{s_m} \delta_{km}^o,$$  \hspace{1cm} (6)

where $km$ specifies the $k$\textsuperscript{th} neuron of the $m$\textsuperscript{th} (output) layer; $s_m = N_{m-1}+1$, i.e. the number of all neurons on
the previous layer (layer \( m-1 \), which the error is backpropagated to) incremented by 1, \( \delta_{\text{in}} \) is the global error of the entire network (5) taken from the \( k \text{th} \) neuron of the \( m \text{th} \) (output) layer.

The errors of the hidden layers’ neurons are:

\[
\delta_{ij} = \frac{1}{s_j} \sum_{j=1}^{s_k} \delta_{kji} (w_{kj}^{d+1})^{-1},
\]

where \( kj \) specifies the \( k \text{th} \) neuron of the \( j \text{th} \) layer \((j=1,...,m-1)\); \( s_j = N_{j+1} + 1 \), \( j=2,...,m \); \( s_1 = 1 \) is the number of all neurons on the layer \( j-1 \) (the previous layer \( j \) which error is backpropagated to) incremented by 1. The weights for all neurons of the network are corrected after calculation of the errors. In order to do this, the learning rule (4) is used. Hence, the following correction rules are used for the weights [3]:

\[
\hat{w}_{ij} = w_{ij} + \frac{C_{ij}}{(N_{i+1}+1)} \delta_{ij} \nabla \hat{m}_{i-1}, \quad i = 1,...,n,
\]

\[
\hat{w}_{0j} = w_{0j} + \frac{C_{0j}}{(N_{j+1}+1)} \delta_{0j},
\]

for the neurons from the \( m \text{th} \) (output) layer (\( k \text{th} \) neuron of \( m \text{th} \) layer),

\[
\hat{w}_{ij}^{d+1} = w_{ij}^{d+1} + \frac{C_{ij}}{(N_{i+1}+1)} \delta_{ij} \nabla \hat{m}_{i-1}, \quad i = 1,...,n,
\]

\[
\hat{w}_{0j}^{d+1} = w_{0j}^{d+1} + \frac{C_{0j}}{(N_{j+1}+1)} \delta_{0j},
\]

for the neurons from the \( 2 \text{nd} \) till \( m-1 \text{th} \) layer (\( k \text{th} \) neuron of the \( j \text{th} \) layer \( j=2, ..., m-1 \)), and

\[
\hat{w}_{ij}^{d+1} = w_{ij}^{d+1} + \frac{C_{ij}}{(n+1)} | z_{ij} | \delta_{ij} \nabla \hat{m}_{i-1}, \quad i = 1,...,n,
\]

\[
\hat{w}_{0j}^{d+1} = w_{0j}^{d+1} + \frac{C_{0j}}{(n+1)} \delta_{0j},
\]

for the neurons of the \( 1 \text{st} \) hidden layer, where \( C_{kj} \) is a constant part of the learning rate (it should be mentioned that in our experiments for all the neurons we took \( C_{kj} = 1, k = 1,...,N_j; j = 1,...,m \)). The factor \( 1 / z_{kj} \), where \( z_{kj} \) is a weighted sum of the \( kj \text{th} \) neuron on the previous learning iteration, is a variable self-adaptive part of the learning rate, which is used only for the hidden neurons, as it is recommended in [3].

In general, the learning process should continue until the following condition is satisfied:

\[
E = \frac{1}{N} \sum_{s=1}^{S} \sum_{s} (\delta_{s}^*_{\text{in}})^2(W) = \frac{1}{N} \sum_{s=1}^{S} E_s \leq \lambda,
\]

where \( E_s \) is a square error of the network for the \( s \text{th} \) pattern from the learning set \((E_s = \sum_{s} (\delta_{s}^*_{\text{in}})^2(W))\), \( N \) is the number of patterns in the learning set, and \( \lambda \) determines the precision of learning. In particular, in the case when \( \lambda = 0 \) the equation (11) is transformed to \( \forall k, \forall s \delta_{s}^*_{\text{in}} = 0 \). We will use exactly the last case in our experiments.

3. DATA DESCRIPTION

We use two well-known microarray gene expression data sets; “Novartis” and “Lung”. Both publicly available datasets consist of multiple classes. The “Lung” data set includes 197 samples with 419 features (genes) that represent the four known classes. The “Novartis” data set includes 103 samples with 697 features that also represent the four known classes. Though feature selection is left outside the scope of this study, it should be noted that any screening or selection of features prior to classification can have significant effect on the result. For a more detailed description of the data sets applied in this study, see [7]-[9].

Since using MLMVN we have to put the inputs on the unit circle, the gene expression data was not used in classification as such. We used a simple linear transform (see Section 4) to convert the data to the points on the unit circle. Actually this transform simply changes linearly a range of the data and completely preserves the data nature.

We used a K-random subsampling with \( K=15 \) to separate the data on the training and testing sets. Thus \( K=15 \) training and testing sets have been created. For the “Lung” data set 44 samples of 197 were used for training and the rest 153 ones for testing for all \( K=15 \) cases. For the “Novartis” data set 51 samples were used for training and the rest 50 ones for testing for all \( K=15 \) cases.

4. SIMULATION RESULTS

To test the MLMVN as a classifier for solving the presented problems, we used the network with one hidden layer and one output layer containing the same number of neurons as the number of classes. The best results for both test data sets are shown by the network with 6 neurons on a single hidden layer (any increase of the hidden neurons amount does not improve the results; on the other hand, the results are a bit worse for a smaller amount of the hidden neurons). Thus taking into account that we have in both classification problems exactly 4 classes, the network \( n \to 6 \to 4 \) (where \( n \) is the number of inputs) has been used.

We used the MLMVN with the continuous inputs and a discrete output. However, the hidden neurons were continuous-valued, while the output ones combine the continuous inputs with a discrete output. In order to put the original real-valued inputs to the unit circle, their continuous-valued, while the output ones combine the continuous-valued inputs with a discrete output. In order to put the original real-valued inputs to the unit circle, their continuous inputs with a discrete output. In order to put the original real-valued inputs to the unit circle, their continuous inputs with a discrete output.
samples belonging to one of the four considered classes. This means that each neuron has to recognize patterns only from one of the four classes and to reject all other patterns. Hence the activation function of all the output neurons separates the complex plane onto two semiplanes: the upper one is reserved for recognition of the patterns from a particular class, while the bottom one is reserved for the rejection.

During the learning process we directed the weighted sum to the angles \( \pi / 2 \) in the upper semiplane and \( 3\pi / 2 \) in the bottom semiplane. During the learning process the domains \( \pi / 2 \pm \pi / 8 \) and \( 3\pi / 2 \pm \pi / 8 \) were considered as acceptable.

The learning process converges very quickly starting from the random vectors with the real and imaginary parts belonging to \([0, 1]\). It requires 2-3 minutes using a software simulator developed in the Borland Delphi 5 environment on a PC with Pentium IV 3.0 GHz CPU.

To verify the results, as it was mentioned above, a \( K \)-random subsampling cross validation with \( K=15 \) has been used for both problems. For the "Novartis" data set there is 96.35\% classification rate, and for the "Lung" data set there is 94.32\% classification rate. Since the variation of the error for all 15 data splits is very small (0.41 for the "Novartis" data set and 0.39 for the "Lung" data set), this result is very stable. For comparison, the classification results for the "Novartis" data set by using the \( k \) nearest neighbors (\( kNN \)) classifier with \( k=1, 3, \) and 5 were 97.69\%, 97.44\%, and 97.31\%, respectively. For the "Lung" data set, the classification accuracy for 1NN classifier was 89.80\%, for 3NN it was 91.11\%, and for 5NN the accuracy was 92.55\%. Exactly the same data transformation and subsampling partitions were used for all classifiers.

We can conclude from these results that for the multiclass gene expression data classification problem the MLMVN shows the results that are comparable with the \( kNN \) classifier. However, due to the shortcomings of cross validation estimators in small sample settings [10], single results do not necessarily provide a reliable basis for comparison between different classification methods, or for drawing direct conclusions on classifier performance. One classifier shows a bit better result for the one data set, another one shows a bit better result for another data set. However, the microarray test cases should be considered as good examples of possible new application areas of the MLMVN.

5. CONCLUSION

A multilayer neural network based on multi-valued neurons (MLMVN) has proven to be a flexible, accurate and fast algorithm for supervised classification. Here the MLMVN classifier is applied to microarray gene expression data classification. The results for two data sets are comparable with the ones obtained with widely used \( kNN \) classifiers. In the multiclass classification tasks performed for "Novartis" and "Lung" data sets, relatively simple network (\( n \rightarrow 6 \rightarrow 4 \)) provided 96.35\% and 94.32\% classification rates, respectively.

A possible direction for future research is to continue exploring the performance of MLMVN classifier in the context of microarray gene expression data. A more extensive set of results with different error estimators could provide more information on the accuracy and a more reliable basis for comparison with other classification methods.

6. ACKNOWLEDGMENTS

This work was supported by the Academy of Finland, project No. 213462 (Finnish Centre of Excellence program (2006 - 2011)).

7. REFERENCES