Advanced Learning Algorithms

Bogdan M. Wilamowski
Electrical and Computer Engineering, Auburn University, Alabama, USA
wilam@ieee.org

Abstract— **The comparisons of various learning algorithms were presented and it was shown that most popular neural network topologies (MLP) and most popular training algorithm (EBP) are not giving optimal solution. Instead MLP networks much simpler neural network topologies can be used to produce similar or better results. Instead of popular EBP more advance algorithms such as LM or NBN should be used. They not only produce results in couple order of magnitude shorter time, but also good solutions can be found for networks where EBP algorithm fails. Eventually EBP can find solution if number of neurons in the network increase, but this solution in most cases will be far from the optimum one.**

I. INTRODUCTION

It is easy to train neural networks with excessive number of neurons. Such complex architectures for given patterns can be trained to very small errors, but such networks do not have generalization abilities. These networks are not able to deliver a correct response to new patterns, which were not used for training [1]. This way the main purpose of using neural networks is missed. In order to properly utilize neural networks its architecture should be as simple as possible to perform the required function, but in order to train them more advanced algorithms should be used [2-4]

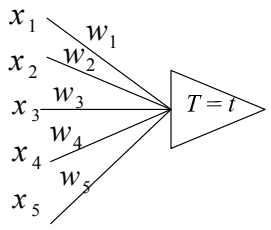


Figure 1. A single neuron with several inputs

II. MAIN PRINCIPLE OF NEURAL NETWORKS LEARNING

Let us consider a single neuron with several inputs (Fig. 1) and let us assume that the net value on the summing input of the neuron is

$$net = \sum_{i=1}^{n} w_i x_i \qquad (1)$$

One may question what is maximum value of the *net* and when it is achieved if both inputs and weights may have binary bipolar (-1,+1) values. Notice that if both weights and inputs have the same values, for example:

$$\mathbf{w} = [-1,+1,-1,+1,-1]] \qquad (2)$$

$$\mathbf{x} = [-1,+1,-1,+1,-1]] \qquad (3)$$

Then *net* = *n*, where n is the number of inputs. If there is mismatch of input and weight on one of the inputs then the *net* value will be reduced by 2. In general,

$$net = n - 2HD(\mathbf{w}, \mathbf{x}) \qquad (4)$$

where *HD* is the Hamming Distance between input pattern **x** and the weight vector **w**. In other words the neuron receives maximum excitation if input pattern and weight vector are equal. This is true for binary bipolar values, but it is also true if both input patterns and weight vectors are normalized. Therefore, the main learning principle is that the required weight change Δw should be

$$\Delta w = c \cdot LR \cdot x \qquad (5)$$

where *c* is learning constant and LR is a learning rule which distinguished different learning methods.

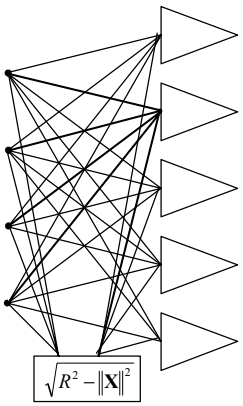


Figure 2. Input pattern transformation in order to preserve all information

Unfortunately, the requirement of normalization of weights and patterns often leads to removal of important information. In order to preserve all information instead of normalization, all weights and patterns could be projected on the hypersphere in the n+1 dimensions (see Fig. 2). With this input pattern transformation the input neurons are also gaining ability for separation of entire

978-1-4244-4113-6/09/$25.00 ©2009 IEEE

clusters. This way for example in two dimensional space 3 neurons can separate three clusters (Figure 3).

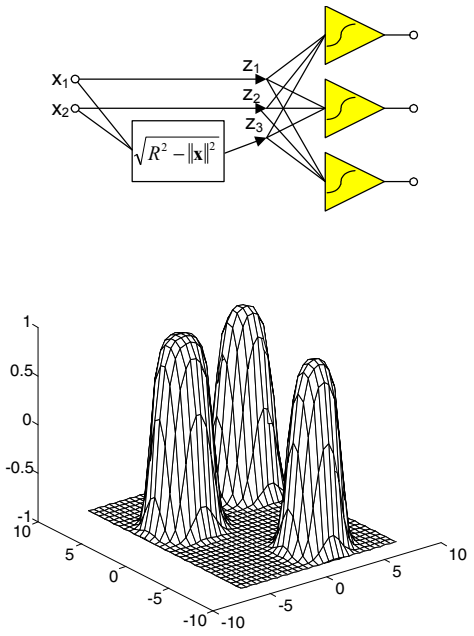


Figure 3. By increasing dimensionality of the problem it is much easier to separate clusters

III. Issues with Learning Algorithms

Similarly to the biological neurons, the weights in artificial neurons are adjusted during a training procedure. Most algorithms require supervision but there are algorithms which can train neural networks without supervision (the desired outcome is not known). Common learning rules are described below[4] [5].

A. Hebbian learning rule

The Hebb learning rule [6] is based on the assumption that if two neighbor neurons must be activated and deactivated at the same time, then the weight connecting these neurons should increase. For neurons operating in the opposite phase, the weight between them should decrease. If there is no correlation, the weight should remain unchanged. This assumption can be described by the formula

$$\Delta w_{ij} = c\, x_i\, o_j \qquad (6)$$

where w_{ij} is the weight from *i-th* to *j-th* neuron, c is the learning constant, x_i is the signal on the *i-th* input and o_j is the output signal. The training process starts usually with values of all weights set to zero. This learning rule can be used for both soft and hard threshold neurons. Since desired responses of neurons are not used in the learning procedure, this is the unsupervised learning rule.

B. Correlation learning rule

The correlation learning rule is based on a similar principle as the Hebb learning rule. It assumes that weights between simultaneously responding neurons should be largely positive, and weights between neurons with opposite reaction should be largely negative. Mathematically, it can be written that weights should be proportional to the product of states of connected neurons. In contrary to the Hebb rule, the correlation rule is the supervised training. Instead of actual response, the desired response is used for weight change calculation

$$\Delta w_{ij} = c\, x_i\, d_j \qquad (7)$$

This training algorithm starts usually with initialization of weights to zero values.

C. Instar learning rule

If input vectors, and weights, are normalized, or they have only binary bipolar values (*-1* or *+1*), then the *net* value will have the largest positive value when the weights have the same values as the input signals. Therefore, weights should be changed only if they are different from the signals

$$\Delta w_i = c(x_i - w_i) \qquad (8)$$

Note, that the information required for the weight change is only taken from the input signals. This is a very local and unsupervised learning algorithm.

D. WTA - Winner Takes All

The WTA is a modification of the instar algorithm where weights are modified only for the neuron with the highest *net* value. Weights of remaining neurons are left unchanged. Sometimes this algorithm is modified in such a way that a few neurons with the highest *net* values are modified at the same time. This unsupervised algorithm (because we do not know what are desired outputs) has a global character. The *net* values for all neurons in the network should be compared in each training step. The WTA algorithm, developed by Kohonen [7] is often used for automatic clustering and for extracting statistical properties of input data.

E. Outstar learning rule

In the outstar learning rule it is required that weights connected to the certain node should be equal to the desired outputs for the neurons connected through those weights

$$\Delta w_{ij} = c(d_j - w_{ij}) \qquad (9)$$

where d_j is the desired neuron output and c is small learning constant which further decreases during the learning procedure. This is the *supervised training* procedure because desired outputs must be known. Both instar and outstar learning rules were developed by Grossberg [8].

F. Perceptron learning rule

$$\Delta \mathbf{w}_i = c\, \delta\, \mathbf{x}_i \qquad (10)$$

$$LR = d - o \qquad (11)$$

$$\Delta \mathbf{w}_i = \alpha\, \mathbf{x}_i \left(d - \mathrm{sign}(net)\right) \qquad (12)$$

$$net = \sum_{i=1}^{n} w_i \mathbf{x}_i \qquad (13)$$

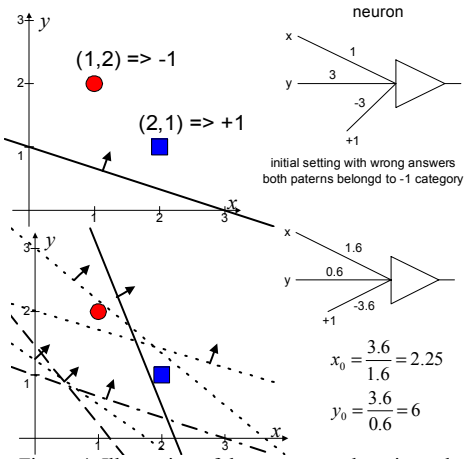


Figure 4. Illustration of the perceptron learning rule

G. Widrow-Hoff (LMS) learning rule

Widrow and Hoff [9] developed a supervised training algorithm which allows to train a neuron for the desired response. This rule was derived so the square of the difference between *net* and output value is minimized.

$$Error_j = \sum_{p=1}^{P} \left(net_{jp} - d_{jp} \right)^2 \qquad (14)$$

where $Error_j$ is the error for *j-th* neuron, P is the number of applied patterns, d_{jp} is the desired output for *j-th* neuron when *p-th* pattern is applied, and *net* is given by equation (1). This rule is also known as the LMS (Least Mean Square) rule. By calculating a derivative of (14) with respect to w_i, one can find a formula for the weight change.

$$\Delta w_{ij} = c\, x_{ij} \sum_{p=1}^{P} \left(d_{jp} - net_{jp} \right) \qquad (15)$$

Note, that weight change Δw_{ij} is a sum of the changes from each of the individual applied patterns. Therefore, it is possible to correct weight after each individual pattern is applied. If the learning constant c is chosen to be small, then both methods gives the same result. The LMS rule works well for all type of activation functions. This rule tries to enforce the *net* value to be equal to desired value. Sometimes, this is not what we are looking for. It is usually not important what the *net* value is, but it is important if the *net* value is positive or negative.

H. Linear regression

The LMS learning rule requires hundreds or thousands of iterations before it converges to the proper solution. Using the linear regression the same result can be obtained in only one step.

Considering one neuron and using vector notation for a set of the input patterns $\boldsymbol{x}$ applied through weights $\boldsymbol{w}$ the value *net* is calculated using

$$\mathbf{xw} = net \qquad (16)$$

or

$$\begin{bmatrix} \sum_{p=1}^{P} x_{p1}x_{p1} & \sum_{p=1}^{P} x_{p1}x_{p2} & \cdots & \sum_{p=1}^{P} x_{p1}x_{pN} \\ \sum_{p=1}^{P} x_{p2}x_{p1} & \sum_{p=1}^{P} x_{p2}x_{p2} & \cdots & \sum_{p=1}^{P} x_{p2}x_{pN} \\ \cdots & \cdots & \cdots & \cdots \\ \sum_{p=1}^{P} x_{pN}x_{p1} & \sum_{p=1}^{P} x_{pN}x_{p2} & \cdots & \sum_{p=1}^{P} x_{pN}x_{pN} \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_N \end{bmatrix} = \begin{bmatrix} \sum_{p=1}^{P} d_p x_{p1} \\ \sum_{p=1}^{P} d_p x_{p2} \\ \cdots \\ \sum_{p=1}^{P} d_p x_{pN} \end{bmatrix} \qquad (17)$$

Note that the size of the input patterns is always augmented by one, and this additional weight is responsible for the threshold. This method, similar to the LMS rule, assumes a linear activation function, so the *net* value should be equal to desired output values $\mathbf{d}$

$$\mathbf{xw} = \mathbf{d} \qquad (18)$$

Usually $p > n+1$, and the above equation can be solved only in the least mean square error sense

$$\mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{d} \qquad (19)$$

I. Delta learning rule

The LMS method assumes linear activation function $o=net$, and the obtained solution is sometimes far from optimum. If error is defined as

$$Error_j = \sum_{p=1}^{P} \left(o_{jp} - d_{jp} \right)^2 \qquad (20)$$

Then the derivative of the error with respect to the weight w_{ij} is $\frac{d\, Error_j}{d\, w_{ij}} = 2 \sum_{p=1}^{P} \left(o_{jp} - d_{jp} \right) \frac{df(net_{jp})}{d\, net_{jp}} x_i \qquad (21)$

Note, that this derivative is proportional to the derivative of the activation function $f'(net)$.

Using the cumulative approach, the neuron weight w_{ij} should be changed with a direction of gradient

$$\Delta w_{ij} = c\, x_i \sum_{p=1}^{P} \left(d_{jp} - o_{jp} \right) f_{j'p} \qquad (22)$$

in case of the incremental training for each applied pattern

$$\Delta w_{ij} = c\, x_i\, f_{j'} \left(d_j - o_j \right) \qquad (23)$$

the weight change should be proportional to input signal x_i, to the difference between desired and actual outputs d_{jp}-o_{jp}, and to the derivative of the activation function f'_{jp}. Similar to the LMS rule, weights can be updated in both the incremental and the cumulative methods. In comparison to the LMS rule, the delta rule always leads to a solution close to the optimum.

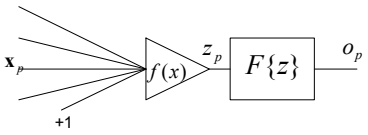
J. Error Backpropagation learning

The delta learning rule can be generalized for multilayer networks [10-11]. Using a similar approach, as it is described for the delta rule, the gradient of the global error can be computed in respect to each weight in the network.

$$o_p = F\left\{ f\left(w_1 x_{p1} + w_2 x_{p2} + \cdots + w_n x_n \right) \right\} \qquad (24)$$

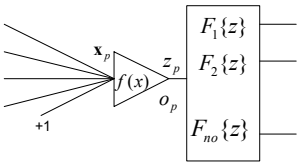
$$Total\_Error = \sum_{p=1}^{np} \left[d_p - o_p \right]^2 \qquad (25)$$

$$\frac{d(TE)}{dw_i} = -2 \sum_{p=1}^{np} \left[\left(d_p - o_p \right) F'\left\{ z_p \right\} f'\left(net_p \right) x_{pi} \right] \qquad (26)$$



$$\Delta \mathbf{w}_p = \alpha \sum_{p=1}^{np} \left[\left(d_p - o_p \right) F' \left\{ z_p \right\} f' \left(net_p \right) \mathbf{x}_p \right]$$

Figure 5. Error backpropagation for neural networks with one output



$$\Delta \mathbf{w}_p = \alpha \sum_{o=1}^{no} \sum_{p=1}^{np} \left[\left(d_{op} - o_{op} \right) F' \left\{ z_p \right\} f' \left(net_p \right) \mathbf{x}_p \right]$$

Figure 6. Error backpropagation for neural networks with multiple output

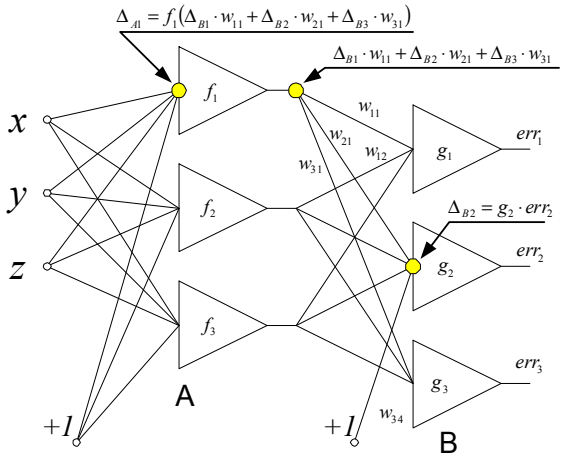


Figure 7. Calculation errors in neural network using error backpropagation algorithm

IV. Heuristic Approach to EBP

The backpropagation algorithm has many disadvantages which leads to very slow convergence. One of the most painful is that in the backpropagation algorithm the learning process almost perishes for neurons responding with the maximally wrong answer (Figure 8).

For example, if the value on the neuron output is close to *+1* and desired output should be close to *-1*, then the neuron gain *f'(net)=0* and the error signal cannot backpropagate, so the learning procedure is not effective. To overcome this difficulty, a modified method for derivative calculation was introduced by Wilamowski and Torvik [12]. The derivative is calculated as the slope of a line connecting the point of the output value with the point of the desired value as shown in Fig. 8.

$$f_{modif} = \frac{o_{desired} - o_{actual}}{net_{desired} - net_{actual}} \qquad (31)$$

Note, that for small errors, equation (31) converges to the derivative of activation function at the point of the output value. With an increase of the system dimensionality, a chance for local minima decrease. It is believed that the described above phenomenon, rather than a trapping in local minima, is responsible for convergence problems in the error backpropagation algorithm.

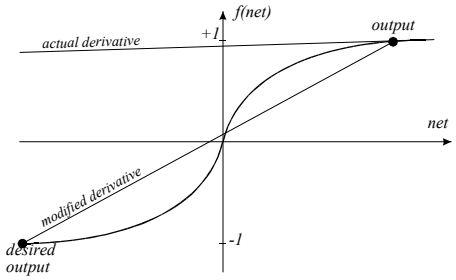


Figure 8. In traditional EBP algorithm very large errors are not being propagated through the network

By introduction new way of defining derivative of the activation function the learning speed could be significantly increased

$$f'(net) = k\left[1 - o^2\right] \qquad (27)$$

$$f'(net) = k\left[1 - o^2\left(1 - \left(\frac{err}{2}\right)^2\right)\right] \qquad (28)$$

For small errors: $f'(net) = k\left[1 - o^2\right]$ (29)

For large errors: $f'(net) = k$ (30)

A. Momentum term

The backpropagation algorithm has a tendency for oscillation. In order to smooth up the process, the weights increment Δw_{ij} can be modified by introduction of the momentum terms:

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) + \alpha \Delta w_{ij}(n-1) \qquad (32)$$

or

$$w_{ij}(n+1) = w_{ij}(n) + (1 - \alpha)\,\Delta w_{ij}(n) + \alpha \Delta w_{ij}(n-1) \qquad (33)$$

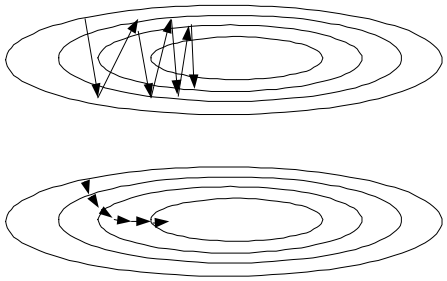


Figure 9. Solution process without and with momentum term

B. Gradient direction search

The backpropagation algorithm can be significantly sped up, when after finding components of the gradient, weights are modified along the gradient direction until a minimum is reached. This process can be carried on without the necessity of computational intensive gradient calculation at each step. The new gradient components are calculated once a minimum on the direction of the previous gradient is obtained. This process is only possible for cumulative weight adjustment. One method to find a minimum along the gradient direction is the tree step process of finding error for three points along gradient direction and then, using a parabola approximation, jump directly to the minimum.

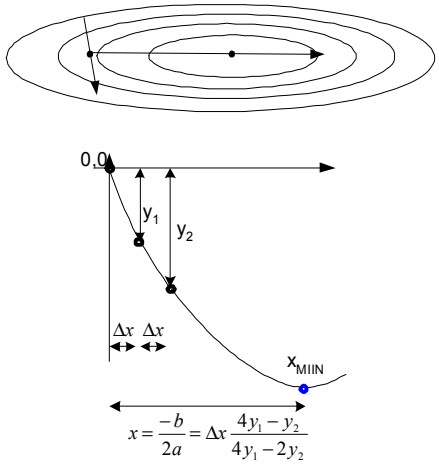


Figure 10. Search on the gradient direction before a new calculation of gradient components.

C. Quickprop algorithm by Fahlman

The fast learning algorithm using the above approach was proposed by Fahlman [14] and it is known as the *quickprop*.

$$\Delta w_{ij}(t) = -\alpha S_{ij}(t) + \gamma_{ij} \Delta w_{ij}(t-1) \qquad (34)$$

$$S_{ij}(t) = \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} + \eta w_{ij}(t) \qquad (35)$$

Where:

α learning constant

γ memory constant (small 0.0001 range) leads to reduction of weights and limits growth of weights

η momentum term selected individually for each weight

$0.01 < \alpha < 0.6$ when $\Delta w_{ij} = 0$ or sign of Δw_{ij}

$\alpha = 0$ otherwise

$$S_{ij}(t) \Delta w_{ij}(t) > 0 \qquad (36)$$

$$\Delta w_{ij}(t) = -\alpha S_{ij}(t) + \gamma_{ij} \Delta w_{ij}(t-1) \qquad (37)$$

momentum term selected individually for each weight is very important part of this algorithm. Quickprop algorithm sometimes reduces computation time a hundreds times

Later this algorithm was simplified:

$$\beta_{ij}(t) = \frac{S_{ij}(t)}{S_{ij}(t-1) - S_{ij}(t)} \qquad (38)$$

Modified Quickprop algorithm is simpler and often gives better results than the original one.

D. RPROP Resilient Error Back Propagation

Very similar to EBP, but weights adjusted without using values of the propagated errors, but only its sign [15]. Learning constants are selected individually to each weight based on the history

$$\Delta w_{ij}(t) = -\alpha_{ij} \operatorname{sgn}\left(\frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}(t)} \right) \qquad (39)$$

$$S_{ij}(t) = \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} + \eta w_{ij}(t) \qquad (40)$$

$$\alpha_{ij}(t) = \begin{cases} \min(a \cdot \alpha_{ij}(t-1), \alpha_{\max}) & \text{for} \quad S_{ij}(t) \cdot S_{ij}(t-1) > 0 \\ \max(b \cdot \alpha_{ij}(t-1), \alpha_{\min}) & \text{for} \quad S_{ij}(t) \cdot S_{ij}(t-1) < 0 \\ \alpha_{ij}(t-1) & \text{otherwise} \end{cases}$$

E. Back Percolation

Error is propagated as in EBP and then each neuron is "trained" using an algorithm to train one neuron such as pseudo inversion. Unfortunately pseudo inversion may lead to errors, which are sometimes larger than 2 for bipolar or larger than 1 for unipolar

G. Delta-bar-Delta

For each weight the learning coefficient is selected individually [16]. It was developed for quadratic error functions

$$\Delta \alpha_{ij}(t) = \begin{cases} a & \text{for} \quad S_{ij}(t-1) D_{ij}(t) > 0 \\ -b \cdot \alpha_{ij}(t-1) & \text{for} \quad S_{ij}(t-1) D_{ij}(t) < 0 \\ 0 & \text{otherwise} \end{cases} \qquad (41)$$

$$D_{ij}(t) = \frac{\partial E(t)}{\partial w_{ij}(t)} \qquad (42)$$

$$S_{ij}(t) = (1-\xi) D_{ij}(t) + \xi S_{ij}(t-1) \qquad (43)$$

V. Second Order Algorithms

A. Levenberg-Marquardt Algorithm (LM)

The Levenberg-Marquardt method was sucesfull applied to NN training [17]. In the steepest descent method (error backpropagation)

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \mathbf{g} \qquad (44)$$

where **g** is gradient vector

$$\textit{gradient} \quad \mathbf{g} = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{bmatrix} \qquad (45)$$

Newton method

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{A}_k^{-1} \mathbf{g} \qquad (46)$$

where $\mathbf{A}_k$ is Hessian

$$\textit{Hessian} \quad \mathbf{A}_k = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_2 \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_1} \\ \frac{\partial^2 E}{\partial w_1 \partial w_2} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_1 \partial w_n} & \frac{\partial^2 E}{\partial w_2 \partial w_n} & \cdots & \frac{\partial^2 E}{\partial w_n^2} \end{bmatrix} \qquad (47)$$

$$\mathbf{J}=\begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_n} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{M1}}{\partial w_1} & \frac{\partial e_{M1}}{\partial w_2} & \cdots & \frac{\partial e_{M1}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{1P}}{\partial w_1} & \frac{\partial e_{1P}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_N} \\ \frac{\partial e_{2P}}{\partial w_1} & \frac{\partial e_{2P}}{\partial w_2} & \cdots & \frac{\partial e_{2P}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{MP}}{\partial w_1} & \frac{\partial e_{MP}}{\partial w_2} & \cdots & \frac{\partial e_{MP}}{\partial w_N} \end{bmatrix} \quad (48)$$

$$\mathbf{A}=2\mathbf{J}^T\mathbf{J} \quad \text{and} \quad \mathbf{g}=2\mathbf{J}^T\mathbf{e} \quad (49)$$

Gauss-Newton method:

$$\mathbf{w}_{k+1}=\mathbf{w}_k-\left(\mathbf{J}_k^T\mathbf{J}_k\right)^{-1}\mathbf{J}_k^T\mathbf{e} \quad (50)$$

Levenberg - Marquardt method:

$$\mathbf{w}_{k+1}=\mathbf{w}_k-\left(\mathbf{J}_k^T\mathbf{J}_k+\mu\mathbf{I}\right)^{-1}\mathbf{J}_k^T\mathbf{e} \quad (51)$$

The LM algorithm requires computation of the Jacobian J matrix at each iteration step and the inversion of J^TJ square matrix. Note that in the LM algorithm an *N* by *N* matrix must be inverted in every iteration. This is the reason why for large size neural networks the LM algorithm is not practical. Also most of implementations of LM algoritms (like popular MATLAB NN Toolbox) are developed only for MLP (Multi Layer Perceptron) networks which are in most cases far from optimal architectures. The MLP networks not only require larger than minimum number of neurons, but also they are learning slower.

B. Neuron by Neuron Algorithm (NBN)

The Neuron by Neuron Algorithm (NBN) algorithm was developed in order to eliminate many disadvantages of the LM algorithm. Detailed description of the algorithm can be found in [2-4].

VI. Comparison various training algorithms

Experimental comparison of various algorithms can be found in Figures 11 to 18 and in TABLES I to III. For MLP architectures (TABLE I) comparison can be done for all algorithms:

EBP – Error Back Propagation [10]

LM _Levenberg Marquat [17]

NBN Neuron By Neuron [2]

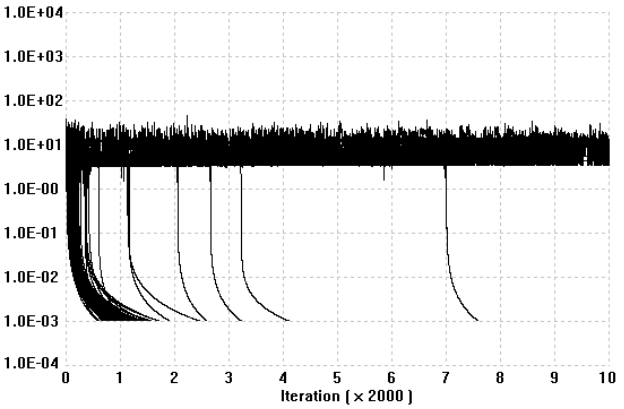


Figure 11. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using EBP algorithm with the success rate of 90% average number if iterations 2438.91, average time 931.92 ms (2-3-3-1 topology)

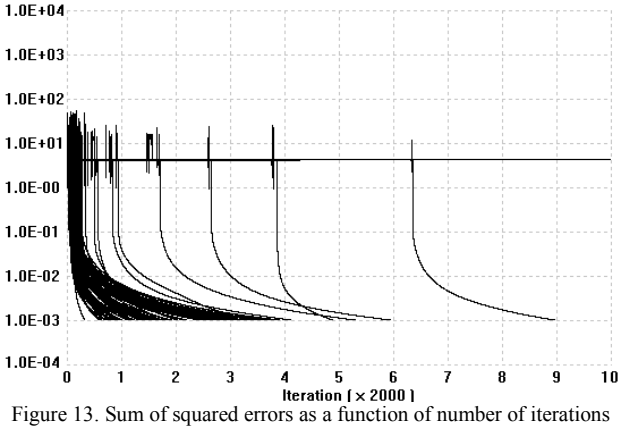


Figure 13. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using EBP algorithm with the success rate of 98%, average number if iterations 3977.15, average time 1382.78 ms (2-1-1-1-1 topology)

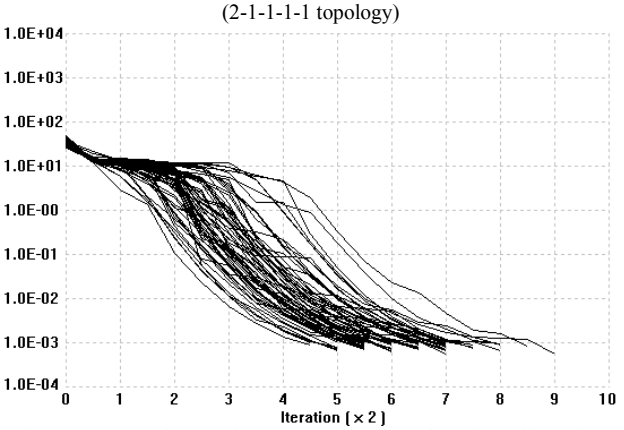


Figure 14. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using NBN algorithm with the success rate of 100%, average number if iterations 12.36, average time 8.15 ms (2-1-1-1-1-1 topology)

TABLE I
COMPARISON OF SOLUTIONS OF PARITY-4 PROBLEM WITH VARIOUS ALGORITHMS ON 1-3-3-1 TOPOLOGY

Type size (pts.)	Averages from 100 runs		
	Success rate	iterations	Computing time
EBP	90%	2438.91	931.92 ms
LM	72%	19.72	19.60 ms
NBN	82%	20.53	19.63 ms

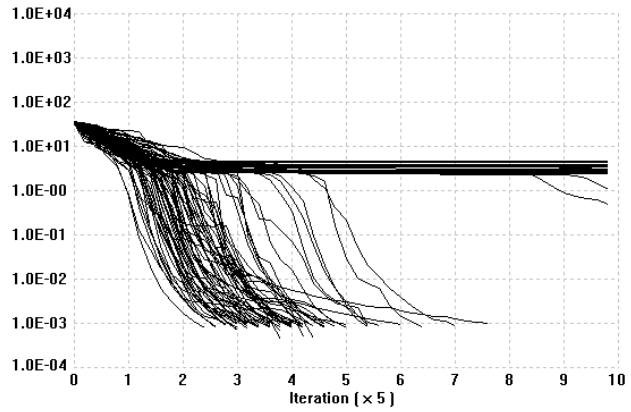


Figure 15. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using EBP algorithm with the success rate of 71% average number if iterations 19.72, average time 19.60 ms (2-1-1-1 topology)

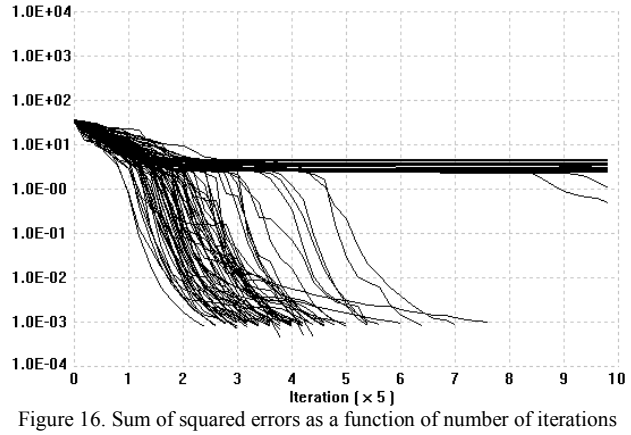


Figure 16. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using LM algorithm with the success rate of 71% average number if iterations 19.72, average time 19.60 ms (2-3-3-1 topology)

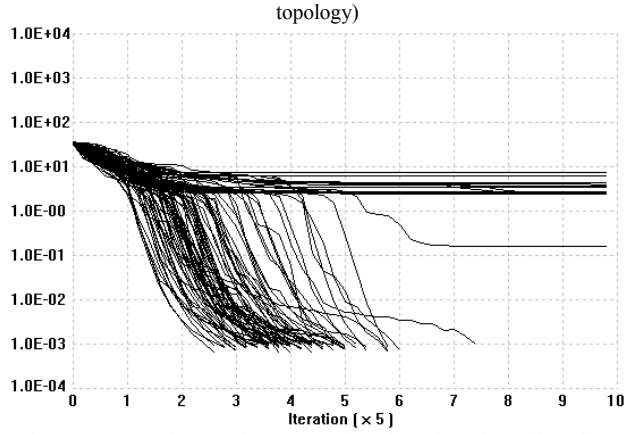


Figure 17. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using NBN algorithm with the success rate of 82% average number if iterations 20.53, average time 19.63 ms (2-3-3-1 topology)

TABLE II
COMPARISON OF SOLUTIONS OF PARITY-4 PROBLEM WITH VARIOUS ALGORITHMS ON 2-1-1-1 TOPOLOGY

Type size (pts.)	Averages from 100 runs		
	Success rate	iterations	Computing time
EBP	88%	7567.81	2985.80ms
LM	N/A	N/A	N/A
NBN	97%	14.59	8.69ms

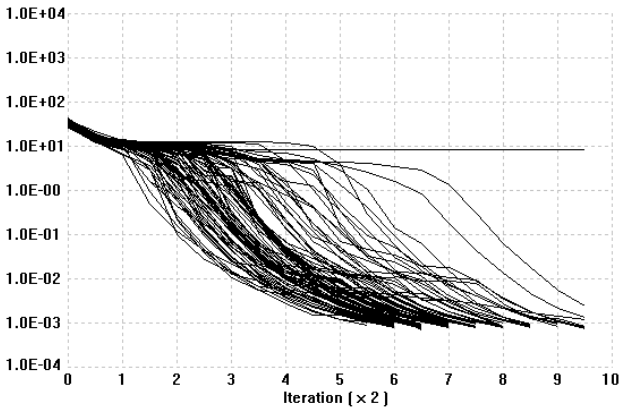


Figure 12. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using NBN algorithm with the success rate of 97% average number if iterations 14.59, average time 8.69 ms (2-1-1-1 topology)

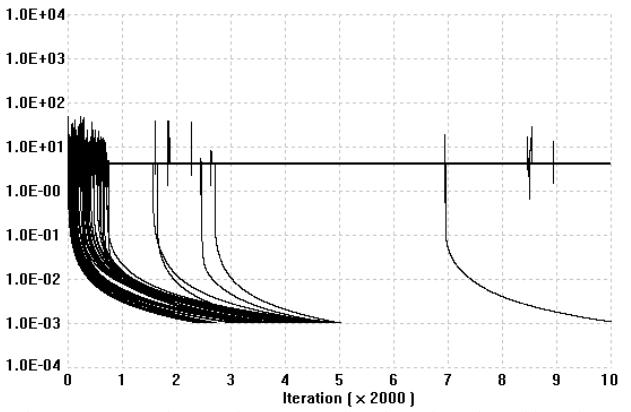


Figure 18. Sum of squared errors as a function of number of iterations for the "Parity-4" problem using EBP algorithm with the success rate of 88% average number if iterations 7567.81, average time 2985.80 ms (2-1-1-1 topology)

TABLE III
COMPARISON OF SOLUTIONS OF PARITY-4 PROBLEM WITH VARIOUS ALGORITHMS ON 2-1-1-1-1 TOPOLOGY

Type size (pts.)	Averages from 100 runs		
	Success rate	iterations	Computing time
EBP	98%	3977.15	1382.78ms
LM	N/A	N/A	N/A
NBN	100%	12.36	8.15ms

For MLP topologies it seems that the EBP algorithm is most robust and has largest success rate for random weight initialization. At the same time the EBP algorithm requires over 1000 times larger number of iterations to converge. Because it is relatively simple; therefore; the time required for each iteration is about 20 times shorter and its actual computation time is only 50 times longer than in the case of other two algorithms.

When arbitrarily connected topologies are considered (including connections across layers) then a much smaller network can be used to solve the same Parity-4 problem. The minimum neural network topology would require only 3 neurons connected in cascade (2-1-1-1). Unfortunately, the LM Algorithm was adopted only for MLP networks and it cannot be applied for this network so the comparison is done only for EBP and NBN algorithms. From TABLE II

one may notice that NBN algorithm has success rate of 97% in comparison to 88% of EBP. The number of iterations in NBN algorithm is about 30 times smaller. Most importantly the computing time of NBN is about 300 times shorter than in the case of EBP. This is because EBP algorithm cannot handle efficiently arbitrarily connected neural networks.

If number of neurons in the cascade topology is increased from 3 to 4 then both algorithms have a much larger chance for success and NBN algorithm has 100% success rate. One may notice that with increasing of network complexity neural networks are losing their ability for generalization. This issue will be discussed in more details in the next section.

VII. WHY WE SHOULD NOT USE LARGER NEURAL NETWORKS THAN NECESSARY ?

Let us consider a neural network which should replace the fuzzy system with the control surface shown in Figure 19.

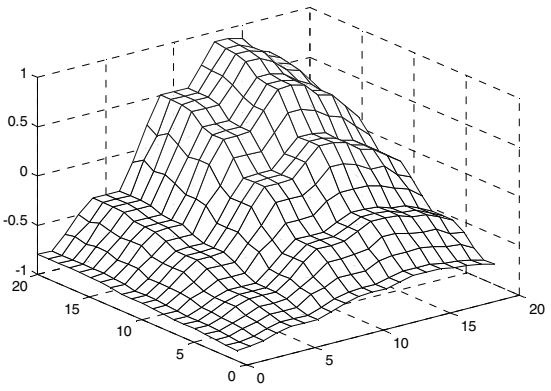


Figure 19. The control surface of a fuzzy controller with 6 and 5 membership functions for both inputs

This control surface can be approximated by neural network topologies with different complexities. Figure 20 shows the control surface obtained from network with 3 neurons (2-1-1-1) and Figure 21 shows the control surface obtained from network with 6 neurons (2-1-1-1-1-1)

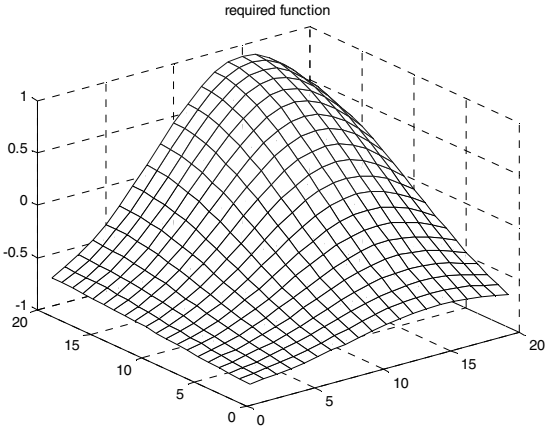


Figure 20. The control surface of a neural controller with 3 neurons

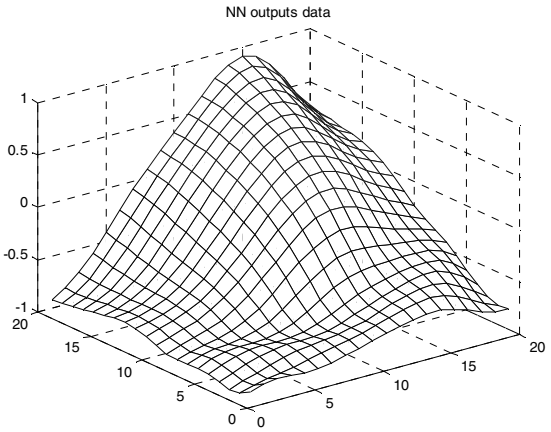


Figure 21. The control surface of a neural controller with 3 neurons

One may notice that if too large neural networks is used the system can find solutions which produces very small error for the training patterns, but for patterns which were not used for training errors actually could be much larger than in the case of much simple network.

What many people are not also aware is that not all popular algorithms can train every neural network. Surprisingly, the most popular EBP (Error Back Propagation) algorithm cannot handle more complex problems while other more advanced algorithms can. Also in most cases neural networks trained with popular algorithms such as EBP produce far from optimum solutions.

For example, training the popular test bench with Wieland two spiral problem can be solved (Fig. 1) with second order using cascade architecture with 8 neurons but in order to solve the same problem with the EBP algorithm (Fig 2) at least 16 neurons and weights in cascade architecture are needed. With only 12 neurons in cascade, the NBN algorithm can produce a very smooth response (Fig. 3) with less than 150 iterations but we were not able to solve this 12 neuron problem with EBP algorithm despite many trials with 1,000,000 iterations limit. More detailed information about the relationship between complexity of network topology and learning algorithms can be found in [1]. The conclusion is that with a better learning algorithm the same problem can be solved with simpler hardware.

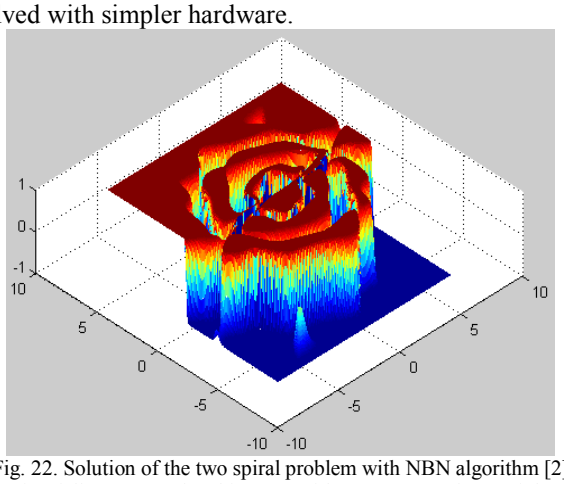


Fig. 22. Solution of the two spiral problem with NBN algorithm [2] using fully connected architecture with 8 neurons and 52 weights

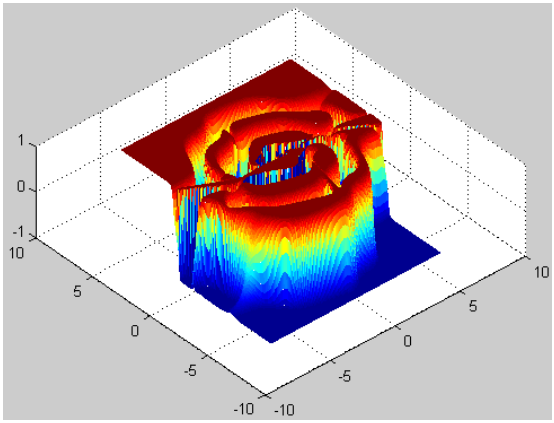


Fig. 23. Solution of the two spiral problem with EBP algorithm using fully connected architecture with 16 neurons and 168 weights

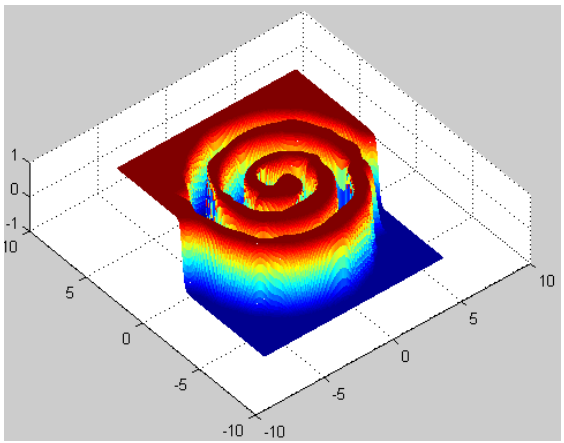


Fig. 24. Solution of the two spiral problem with NBN algorithm [2] using fully connected architecture with only 12 neurons and 102 weights

If we want to take true advantage of neural networks we should use the second order training algorithms such as LM or NBN.

Fully operational software which uses both LM and NBN algorithms can be easy downloaded from *http://www.eng.auburn.edu/~wilambm/nnt*

VIII. Conclusion

The comparisons of various learning algorithms were presented and it was shown that most popular neural network topologies (MLP) and most popular training algorithm (EBP) are not giving optimal solution. Instead MLP much simpler neural network topologies can be used to produce similar or better results. Instead of popular EBP more advanced algorithms such as LM or NBN should be used. They not only produce results in couple order of magnitude shorter time, but also they can find good solutions for networks where EBP algorithm fails. Eventually EBP can find solution if number of neurons in the network increase, but this solution in most cases will be far from the optimum one.

REFERENCES

[1] B. M. Wilamowski, "Special Neural Network Architectures for Easy Implementations for Electronic Control" (keynote), *POWERENG 2009*, Lisbon, Portugal, March 18-20, 2009

[2] B. M. Wilamowski, N. Cotton, O. Kaynak, G. Dundar, "Method of Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks". *Proc. IEEE ISIE*, Vigo, Spain, June, 4-7, pp. 3298-3303, 2007

[3] B. M. Wilamowski, N. Cotton, J. Hewlett, O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms". *Proc. International Conference on Intelligent Engineering Systems*, June 29 2007-July 1 2007, pp. 127-132

[4] Hao Yu and B. M. Wilamowski "C++ Implementation of Neural Networks Trainer" 13th *IEEE Intelligent Engineering Systems Conference, INES 2009,* Barbados, April 16-18, 2009

[5] B. M. Wilamowski "Methods of Computational Intelligence" *ICIT'04 IEEE International Conference on Industrial Technology*, Tunisia, Tunisia, December 8-10, 2004

[6] D. O. Hebb, 1949. *The Organization of Behivior, a Neuropsychological Theory*. John Wiley, New York

[7] T. Kohonen, The Self-organized Map. *Proc. IEEE* 78(9):1464-1480

[8] Grossberg, S. 1969. Embedding Fields: a Theory of Learning with Physiological Implications. *Journal of Mathematical Psychology* 6:209-239

[9] B. Widrow, "Generalization and Information Storage in Networks of Adaline Neurons", *Self-Organizing Systems*, 1962, Spartan Books, pp. 435-461

[10] D. E., Rumenhart, G. E. Hinton, G. E. and R. J, Wiliams, "Learning Representations by Back-Propagating Errors", *Nature*, Vol. **323**, pp. 533-536, 1986

[11] P. J. Werbos "Back-Propagation: Past and future", *Proceeding of International Conference on Neural Networks*, San Diego, CA, **1,** 343-354

[12] B. M. Wilamowski, B. M. and L. Torvik, "Modification of Gradient Computation in the Back-Propagation Algorithm", *ANNIE'93 Intelligent Engineering Systems Through Artificial Neural Networks* Vol. 3, pp. 175-180, ASME PRESS, New York 1993

[13] A. A Miniani and R. D Williams. (1990). "Acceleration of Back-Propagation through Learning Rate and Momentum Adaptation", *Proceedings of International Joint Conference on Neural Networks*, San Diego, CA, **1,** 676-679

[14] S. Fahlman, "An Empirical Study of Learning Speed in Backpropagation Networks", *Tech. Report CMCU-CS-162, Carnegie-Mellon University*, Computer Science Dep., 1988

[15] J. M. Hannan, J. M. Bishop, "A Comparison of Fast Training Algorithms over Two Real Problems", *Fifth International Conf. on Artificial Neural Networks*, 7-9 July, pp. 1-6, 1997

[16] R. A. Jacobs, "Increased Rates of Convergence through Learning Rate Adaptation", *Neural Networks*, Vol. 1, 1988, pp. 295-307

[17] M. T. Hagan and M. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm", *IEEE Transactions on Neural Networks,* Vol. **5**, No. 6, pp. 989-993, 1994