Comparison of BPA and LMA methods for Takagi-Sugeno type MIMO Neuro-Fuzzy Network to forecast Electrical Load Time Series

Felix Pasila
Electrical Engineering Department, Petra Christian University
Surabaya - Indonesia
E-mail: felix@petra.ac.id

ABSTRACT
This paper describes an accelerated Backpropagation algorithm (BPA) that can be used to train the Takagi-Sugeno (TS) type multi-input multi-output (MIMO) neuro-fuzzy network efficiently. Also other method such as accelerated Levenberg-Marquardt algorithm (LMA) will be compared to BPA. The training algorithm is efficient in the sense that it can bring the performance index of the network, such as the sum squared error (SSE), Mean Squared Error (MSE), and also Root Mean Squared Error (RMSE), down to the desired error goal much faster than that the simple BPA or LMA. Finally, the above training algorithm is tested on neuro-fuzzy modeling and forecasting application of Electrical load time series.

Keywords: TS type MISO neuro-fuzzy network, accelerated Levenberg-Marquardt algorithm, accelerated Backpropagation algorithm, time-series forecasting

INTRODUCTION
This paper proposes two extended-training algorithms for a multi-input multi-output (MIMO) Takagi-Sugeno type neuro-fuzzy (NF) network and neuro-fuzzy approach for modeling and forecasting application of electrical load time series. The neuro-fuzzy approach attempts to exploit the merits of both neural network and fuzzy logic based modeling techniques. For example, the fuzzy models are based on fuzzy if-then rules and are to a certain degree transparent to interpretation and analysis. Whereas, the neural networks based model has the unique learning ability. Here, TS type MIMO neuro-fuzzy network is constructed by multilayer feedforward network representation of the fuzzy logic system as described in section 2, whereas their training algorithms are described in section 3. Simulation experiments and results are shown in Section 4, and finally brief concluding remarks are presented in section 5.

NEURO-FUZZY SYSTEMS SELECTION FOR FORECASTING
The common approach to numerical data driven for neuro-fuzzy modeling and identification is using Takagi-Sugeno (TS) type fuzzy model along with continuously differentiable membership functions such as Gaussian function and differentiable operators for constructing the fuzzy inference mechanism and defuzzification of output data using the weighted average defuzzifier. The corresponding output inference can then be represented in a multilayer feedforward network structure. In principle, the neuro-fuzzy network’s architecture used here is identical to the multi input single output architecture of ANFIS [1], which is shown in Figure 1a. Jang introduced 2 inputs 1 output architecture with Takagi-Sugeno type fuzzy model with two rules. The Neuro-Fuzzy model ANFIS incorporates a five-layer network to implement a Takagi-Sugeno type fuzzy system. Figure 1a shows the proposed model of ANFIS and it can process a large number of fuzzy rules. It uses the least mean square method to determine the linear consequents of the Takagi-Sugeno rules. As continuity from ANFIS structure, feedforward multi input multi output is proposed by Palit and Babuška [2] and Palit and Popovic [3] as shown in Figure 1b.

Note: Discussion is inspected before December, 1st 2007, and will be published in the “Jurnal Teknik Elektro” volume 8, number 1 March 2008.

Figure 1a. ANFIS architecture with Takagi-Sugeno type fuzzy model with two rules, 2 Inputs, 1 Output
functions (GMFs), product inference rule, and a weighted average defuzzifier can be defined as (1)-(4) (see [4]).

$$f_j = \sum_{i=1}^{M} y_j^i \cdot h^i$$  \hspace{1cm} (1)$$

where,

$$y_j^i = W_{o,j}^i + W_{1,j}^i x_1 + W_{2,j}^i x_2 + \ldots + W_{n,j}^i x_n$$  \hspace{1cm} (2)$$

$$h^i = \left(\frac{z^i}{b}\right), \quad b = \sum_{i=1}^{M} z^i$$  \hspace{1cm} (3)$$

$$z^i = \prod_{m} \mu_{G_i}(x_i); \quad G_i(x_i) = \exp \left(-\left(\frac{x_i - c_i^j}{\sigma_i^j}\right)^2\right)$$  \hspace{1cm} (4)$$

The corresponding $i^{th}$ rule from the above fuzzy logic system (FLS) can be written as

$$R^i': \text{IF } x_i \text{ is } G_i^1 \text{ AND } \ldots \text{ AND } x_n \text{ is } G_i^n \text{ THEN }$$

$$y_j^i = W_{o,j}^i + W_{1,j}^i x_1 + \ldots + W_{n,j}^i x_n,$$  \hspace{1cm} (5)$$

where, $x_i$ with $i = 1, 2, \ldots, n$ are the $n$ system inputs, $f_j$ with $j = 1, 2, \ldots, m$ are its $m$ outputs, and $G_i^l$ with $i = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, m$ are the Gaussian membership functions of form (4) with the corresponding mean and variance parameters $c_i^l$ and $\sigma_i^l$ respectively and with $y_j^i$ as the output consequent of the $i^{th}$ rule. It must be remembered that the Gaussian membership functions $G_i^l$ actually represent linguistic terms such as low, medium, high, very high, etc. The rules as written in (5) are known as Takagi-Sugeno rules.

Figure 1b shows that the FLS can be represented as a three layer feedforward network. Because of the neuro implementation of the Takagi-Sugeno-type FLS, this figure represents a Takagi-Sugeno-type of MIMO neuro-fuzzy network, where instead of the connection weights and the biases in neural network, we have here the mean $c_i^l$ and also the variance $\sigma_i^l$ parameters of Gaussian membership functions, along with $w_{o,j}^i, w_{1,j}^i$ parameters from the rules consequent, as the equivalent adjustable parameters of the network. If all these parameters of NF network are properly selected, then the FLS can correctly approximate any nonlinear system based on given data.

**TRAINING ALGORITHM FOR FEEDFORWARD NEURAL NETWORK**

The MIMO Feedforward NF network that is represented in Figure 1b can generally be trained using suitable training algorithms. Some standard training algorithms are Backpropagation Algorithm
(BPA) and Levenberg-Marquardt Algorithm (LMA). BPA, the simplest algorithm for neural network (NN) training, is a supervised learning technique used for training artificial neural networks. It was first described by Paul Werbos in 1974, and further developed by David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams in 1986. This algorithm is a learning rule for multi-layered neural networks, credited to Rumelhart. It is most useful for feed-forward networks (networks without feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". BPA is used to calculate gradient of error of the network with respect to the network's modifiable weights. This gradient is almost always used in a simple stochastic gradient descent algorithm to find weights that minimize the error.

In forecasting application, pure BPA, has slow convergence speed in comparison to other second order training [2]. That is why this algorithm needs to be improved using momentum and modified error index. Alternatively, the more efficient training algorithm, such as Levenberg-Marquardt algorithm (LMA) can also be used for training the MIMO Feedforward NN systems [3].

**BPA and Accelerated BPA**

Let us assume that data pairs input-output \( x^p, d^p \), with \( x^p \) as the number of inputs and \( d^p \) as the number of outputs in XIO matrix observation, is given.

The goal is to find a FLS \( f_j(x^p) \) in equations 6-8, so that the performance, Sum Squared Error (SSE), all the equations explained by Palit andPopovic ([3], p.234-p.245), can be defined as

\[
SSE_j = 0.5 \cdot \sum_{p=1}^{N} (e_j^p)^2 = 0.5 \cdot E_j^T E_j , \tag{6}
\]

with \( E_j^T, E_j \) are error transpose and errors for each output \( e_j^p \) in column vector from FLS.

For total performance

\[
SSE_{Total} = \sum_{j=1}^{m} SSE_j = (SSE_1 + SSE_2 + \ldots + SSE_m) , \tag{7}
\]

is minimized.

\( e_j = f_j(x^p) - d^p \) or for convenience \( e_j = f_j - d_j \) \( \tag{8} \)

The problem is, how BPA works to adjust parameters \( \{W_{ij}, c_j\} \) from the rules consequent and the mean \( c'_j \) and variance \( \tau_j^2 \) parameters from the Gaussian membership functions, so that SSE is minimized.

Furthermore, the gradient steepest descent rule for training of feedforward neural network is based on the recursive expressions:

\[
W'_{0j}(k+1) = W_{0j}(k) - \eta \cdot \frac{\partial SSE}{\partial W'_{0j}} \tag{9}
\]

\[
W'_{ij}(k+1) = W_{ij}(k) - \eta \cdot \frac{\partial SSE}{\partial W'_{ij}} \tag{10}
\]

\[
c'_j(k+1) = c'_j(k) - \eta \cdot \frac{\partial SSE}{\partial c'_j} \tag{11}
\]

\[
\tau'^2_j(k+1) = \tau'^2_j(k) - \eta \cdot \frac{\partial SSE}{\partial \tau'^2_j} \tag{12}
\]

Where SSE is the performance function at the \( k^{th} \) iteration step and \( W'_{0j}(k+1), W'_{ij}(k+1), c'_j(k+1), \tau'^2_j(k+1) \) are the free parameters update at the next \( k^{th} \) iteration step, the starting values of them are randomly selected. Constant step size \( \eta \) or learning rate should be chosen properly, usually \( \eta << 1 \), \( i = 1,2,\ldots,n \) (n = number of inputs), \( j = 1,2,\ldots,m \) (m=number of outputs), and \( l = 1,2,\ldots,M \) (M=number of Gaussian membership functions).

Now, let us construct again (1-2) based on Figure 1b, It is sure that SSE depends on output network \( f_j \), therefore performance function SSE, depends on \( W'_{0j}, W'_{ij} \) only through (14), and also depends on \( c'_j, \tau_j^2 \) only through (13)-(15)

\[
f_j = \sum_{l=1}^{M} y_l^j \cdot h^l \tag{13}
\]

\[
y_l^j = \sum_{i=1}^{N} W_{ij}x_i + W_{ij}x_i + \ldots + W_{ij}x_n \tag{14}
\]

\[
h^l = \left( \frac{z^l}{b} \right), \text{ and } b = \sum_{l=1}^{M} z^l \tag{15}
\]

\[
z^l = \prod_{j=1}^{n} \exp \left( -\frac{(x_i - c'_j)^2}{\sigma'^2_j} \right) \tag{16}
\]

Therefore, the corresponding chain rules are

\[
\frac{\partial SSE}{\partial W_{0j}} = \left( \frac{\partial SSE}{\partial f_j} \right) \left( \frac{\partial f_j}{\partial c'_j} \right) \left( \frac{\partial c'_j}{\partial W_{0j}} \right) \tag{17}
\]

\[
\frac{\partial SSE}{\partial W_{ij}} = \left( \frac{\partial SSE}{\partial f_j} \right) \left( \frac{\partial f_j}{\partial c'_j} \right) \left( \frac{\partial c'_j}{\partial W_{ij}} \right) \tag{18}
\]

\[
\frac{\partial SSE}{\partial c'_j} = \left( \frac{\partial SSE}{\partial f_j} \right) \left( \frac{\partial f_j}{\partial c'_j} \right) \left( \frac{\partial c'_j}{\partial c'_j} \right) \tag{19}
\]

\[
\frac{\partial SSE}{\partial \tau'^2_j} = \left( \frac{\partial SSE}{\partial f_j} \right) \left( \frac{\partial f_j}{\partial c'_j} \right) \left( \frac{\partial c'_j}{\partial \tau'^2_j} \right) \tag{20}
\]
And equations (17) – (20) can be written as

\[
\frac{\partial \text{SSE}}{\partial W'_{o_j}} = (f_{j} - d_{j}) \cdot (z_{i} / b)
\]  
(21)

\[
\frac{\partial \text{SSE}}{\partial W'_{o_j}} = (f_{j} - d_{j}) \cdot (z_{i} / b) \cdot x_{i}
\]  
(22)

\[
\frac{\partial \text{SSE}}{\partial c'_{l}} = A \cdot \frac{2 \cdot (z_{i} / b) \cdot (x_{i} - c'_{l})}{(\sigma'_{l})^3}
\]  
(23)

\[
\frac{\partial \text{SSE}}{\partial c'_{l}} = A \cdot \frac{2 \cdot (z_{i} / b) \cdot (x_{i} - c'_{l}) \cdot f_{l} (\sigma'_{l})^3}{(\sigma'_{l})^3}
\]  
(24)

For simplicity, we substitute the term A with

\[
A = \left( \sum_{j=1}^{n} (y'_{j} - f_{j}) \cdot (f_{j} - d_{j}) \right)
\]  
(25)

By substituting equation (21-24) into the (9-12), updated rules for free parameters can be written as

\[
W'_{o_j}(k + 1) = W'_{o_j}(k) - \eta \cdot (f_{j} - d_{j}) \cdot (z_{i} / b)
\]  
(26)

\[
W'_{o_j}(k + 1) = W'_{o_j}(k) - \eta \cdot (f_{j} - d_{j}) \cdot (z_{i} / b) \cdot x_{i}
\]  
(27)

\[
c'_{l}(k + 1) = c'_{l}(k) - \eta \cdot \frac{2 \cdot A \cdot h' \cdot (x_{i} - c'_{l})}{(\sigma'_{l})^3}
\]  
(28)

\[
\sigma'_{l}(k + 1) = \sigma'_{l}(k) - \eta \cdot \frac{2 \cdot A \cdot h' \cdot (x_{i} - c'_{l})}{(\sigma'_{l})^3}
\]  
(29)

where \( h' \) term is the normalized degree of fulfillment of the \( l \)th rule.

\[
h' = z' / b, \text{ with } b = \sum_{i=1}^{M} z_{i} \]  
(30)

BPA training algorithm for TS-type MIMO networks is represented from equations (6) to (30), equivalent with linear fuzzy rules from TS

\[
y'_{j} = W'_{o_j} + W'_{i_1} x_{1} + W'_{i_2} x_{2} + \ldots + W'_{i_{N_{x}}} x_{N_{x}}
\]  
(31)

In case of BPA, to avoid the possible oscillation in final phase of training, very small learning rate \( \eta \) is chosen. Therefore, BPA training needs a large number of training epochs. For acceleration of BPA, momentum (adaptive version of learning rate) and modified error index for performance function update can be applied. Additional momentum (mo) can be seen in equation (32-35) with momentum constant is usually less than one (mo < 1).

\[
W'_{o_j}(k + 1) = W'_{o_j}(k) - \eta \cdot (1 - mo) \cdot (f_{j} - d_{j}) \cdot (z_{i} / b) \]  
(32)

\[
W'_{o_j}(k + 1) = W'_{o_j}(k) - \eta \cdot (1 - mo) \cdot (f_{j} - d_{j}) \cdot (z_{i} / b) \cdot x_{i}
\]  
(33)

\[
c'_{l}(k + 1) = c'_{l}(k) - \eta \cdot (1 - mo) \cdot \frac{2 \cdot A \cdot z' \cdot (x_{i} - c'_{l})}{(\sigma'_{l})^3}
\]  
(34)

\[
\sigma'_{l}(k + 1) = \sigma'_{l}(k) - \eta \cdot (1 - mo) \cdot \frac{2 \cdot A \cdot z' \cdot (x_{i} - c'_{l}) \cdot f_{l}}{(\sigma'_{l})^3}
\]  
(35)

To improve the training performance of feedforward NN, additional modified error index version should be added to the modified BPA with momentum, as proposed by Xiao et al. [5]. This approach can be seen in equations (36–41).

\[
\text{SSE}_{m}(w) = 0.5 \cdot \gamma \cdot \sum_{r=1}^{N} (e_{r}(w) - e_{avg})^2
\]  
(36)

\[
e_{avg} = \frac{1}{N} \sum_{r=1}^{N} e_{r}(w)
\]  
(37)

Thus the new performance index can be written as

\[
\text{SSE}_{\text{new}}(w) = \text{SSE}(w) + \text{SSE}_{m}(w)
\]  
(38)

where \( \text{SSE}(w) \) the unmodified error performance as is defined in (7) and variable \( w \) represents the network free parameter vector in general. The corresponding gradient now becomes to

\[
\nabla \text{SSE}(w) = \sum_{r=1}^{N}(e_{r}(w)) \cdot (\partial e_{r}(w) / \partial w)
\]  
(39)

\[
\nabla \text{SSE}_{m}(w) = \sum_{r=1}^{N} \left( \gamma \cdot (e_{r}(w) - e_{avg}) \right) \cdot (\partial e_{r}(w) / \partial w)
\]  
(40)

\[
\nabla \text{SSE}_{\text{new}}(w) = \sum_{r=1}^{N}(e_{r}(w) + \gamma \cdot (e_{r}(w) - e_{avg})) \cdot (\partial e_{r}(w) / \partial w)
\]  
(41)

The constant gamma \( \gamma \) should be defined properly. In practice, gamma also small, \( \gamma < 1 \)

**Accelerated LMA**

To accelerate the convergence speed on neuro-fuzzy network training that happened in BPA, the Levenberg-Marquardt algorithm (LMA) was proposed and proved [4].

If a function \( V(w) \) is meant to minimize with respect to the parameter vector \( w \) using Newton’s method, the update of parameter vector \( w \) is defined as:

\[
\Delta w = -[\nabla^2 V(w)]^{-1} \cdot \nabla V(w)
\]  
(42)

\[
w(k + 1) = w(k) + \Delta w
\]  
(43)

From equations (42-43), \( \nabla^2 V(w) \) is the Hessian matrix and \( \nabla V(w) \) is the gradient of \( V(w) \). If the function \( V(w) \) is taken to be SSE function as follows:

\[
V(w) = 0.5 \cdot \sum_{r=1}^{N} e_{r}^2(w)
\]  
(44)

Then the gradient of \( V(w) \) and the Hessian matrix \( \nabla^2 V(w) \) are generally defined as:

\[
\nabla V(w) = J^T(w) \cdot e(w)
\]  
(45)
\[ \nabla^2 V(w) = J^T(w) \cdot J(w) + \sum_{j=1}^{N} e_j(w) \cdot \nabla^2 e_j(w) \tag{46} \]

where the Jacobian matrix \( J(w) \) as follows:

\[
J(w) = \begin{bmatrix}
\frac{\partial e_1(w)}{\partial w_1} & \frac{\partial e_1(w)}{\partial w_2} & \cdots & \frac{\partial e_1(w)}{\partial w_N} \\
\frac{\partial e_2(w)}{\partial w_1} & \frac{\partial e_2(w)}{\partial w_2} & \cdots & \frac{\partial e_2(w)}{\partial w_N} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial e_N(w)}{\partial w_1} & \frac{\partial e_N(w)}{\partial w_2} & \cdots & \frac{\partial e_N(w)}{\partial w_N}
\end{bmatrix}
\tag{47}
\]

From (47), it is seen that the dimension of the Jacobian matrix is \( (N \times N_p) \), where \( N \) is the number of training samples and \( N_p \) is the number of adjustable parameters in the network. For the Gauss-Newton method, the second term in (46) is assumed to be zero. Therefore, the update equations according to (42) will be:

\[ \Delta w = -[J^T(w) \cdot J(w)]^{-1} \cdot J^T(w) \cdot e(w) \tag{48} \]

Now let us see the LMA modifications of the Gauss-Newton method.

\[ \Delta w = -[J^T(w) \cdot J(w) + \mu \cdot I]^{-1} \cdot J^T(w) \cdot e(w) \tag{49} \]

where dimension of \( I \) is the \( (N_p \times N_p) \) identity matrix, and the parameter \( \mu \) is multiplied or divided by some factor whenever the iteration steps increase or decrease the value of \( V(w) \).

Here, the updated equation according to (43)

\[ w(k+1) = w(k) - [J^T(w) \cdot J(w) + \mu \cdot I]^{-1} \cdot J^T(w) \cdot e(w) \]

This is important to know that for large \( \mu \), the algorithm becomes the steepest descent algorithm with step size \( 1/\mu \), and for small \( \mu \), it becomes the Gauss-Newton method.

For faster convergence reason and also to overcome the possible trap at local minima and to reduce oscillation during the training \[6\], like in BPA, a small momentum term \( mo \) (practically in electrical load forecasting, adding \( mo \) around 5% to 10% will give better results) also can be added, so that final update (50) becomes

\[ w(k+1) = w(k) - [J^T(w) \cdot J(w) + \mu \cdot I]^{-1} \cdot J^T(w) \cdot e(w) + mo \cdot (w(k) - w(k-1)) \tag{50} \]

Furthermore, like in BPA, Xiaosong et al. \[5\] also proposed to add modified error index (MEI) term in order to improve training convergence. Similar to equations (39-41), the corresponding gradient with MEI can now be defined by using a Jacobian matrix as:

\[ \nabla \text{SSE}_{\text{me}}, (w) = J^T(w) \cdot [e(w) + \gamma \cdot (e(w) - e_{\text{ave}})] \tag{52} \]

where \( e(w) \) is the column vector of errors, \( e_{\text{ave}} \) is sum of error of each column divided by number of training, while \( \gamma \) is a constant factor, \( \gamma \ll 1 \) has to be chosen appropriately.

Now, comes to the computation of Jacobian Matrices. These are the most difficult step in implementing LMA. We describe a simple technique to compute the Jacobian matrix from the Backpropagation results, by taking into account equation (46), the gradient

\[ \nabla V(W_{ij}^l) = \text{SSE} \] can be written as

\[ \nabla V(W_{ij}^l) = \gamma \cdot (e(w) - e_{\text{ave}}) \]

Where \( f_j \) and \( d_j \) are the actual output of the Takagi-Sugeno type MIMO and the corresponding desired output from matrix input-output training data. And then by comparing (53) to (45), where the gradient \( \nabla V(w) \) is expressed with the transpose of the Jacobian matrix multiplied with the network’s error vector

\[ \nabla V(w) = J^T(w) \cdot e(w) \]

then the Jacobian matrix, the transpose of Jacobian matrix for the parameter \( W_{ij}^l \) of the NF network can be written by

\[ J^T(W_{ij}^l) = \left( z_j^l / b \right) \]

\[ J(W_{ij}^l) = \left[ J^T(W_{ij}^l) \right]^T = \left[ z_j^l / b \right]^T \]

with prediction error of fuzzy network

\[ e_j \equiv (f_j - d_j) \]

But if the normalized prediction error on NF network is considered, then instead of equations (55) and (56), the equations will be

\[ J^T(W_{ij}^l) = \left( z_j^l \right) \]

\[ J(W_{ij}^l) = \left[ J^T(W_{ij}^l) \right]^T = \left[ z_j^l \right]^T \]

This is because the normalized prediction error of the MIMO-NF network is

\[ e_j (\text{normalized}) \equiv (f_j - d_j) / b \]

By using similar technique and taking into account equation (54), the transpose of Jacobian matrix and Jacobian matrix for the parameter \( W_{ij}^l \) of the NF network can be written as
\[ J^T(W_{ij}^t) = (z' / b) \cdot x_i \] (62)

\[ J(W_{ij}^t) = [J^T(W_{ij}^t)]^T = [(z' / b) \cdot x_i]^T \] (63)

Also, by considering normalized prediction error from (61), equations (62-63) then become:

\[ J^T(W_{ij}^t) = (z' \cdot x_i) \] (64)

\[ J(W_{ij}^t) = [J^T(W_{ij}^t)]^T = [(z' \cdot x_i)]^T \] (65)

Now, comes to the computation of the rest parameters \( c_i' \) and \( \tau_i' \). Using the similar equation to compute the term \( A \) in equation (25), the equation has to be reorganized as follows

\[ D_j \equiv (y_j' - f_j) \] (66)

By combining equations (57) and (66), the term \( A \) which is explained before can be rewritten as

\[ A = \sum_{j=1}^{N} (D_j \cdot e_j) = (D_1 \cdot c_1 + D_2 \cdot c_2 + \cdots + D_m \cdot e_m) \] (67)

Let us define the terms \( D_{eqv} \) and \( e_{eqv} \) as

\[ A = D_{eqv} \cdot e_{eqv} = (D_1 \cdot e_1 + D_2 \cdot e_2 + \cdots + D_m \cdot e_m) \] (68)

With \( e_{eqv} \) as the same amount of sum squared error that can be found by all the errors \( e_j \) from the MIMO network.

\[ e_{eqv}^p = \sqrt{(e_1^p + e_2^p + \cdots + e_m^p)} \] (69)

Where, \( p = 1, 2, 3, \ldots, N \); corresponding to \( N \) as number of training data. From (68), the term \( D_{eqv} \) can be determined as

\[ D_{eqv} = A \cdot \left(e_{eqv}\right)^{-1} \] (70)

This can also be written in matrix form using pseudo inverse as

\[ D_{eqv} = A \cdot \left(E_{eqv} \cdot E_{eqv}^T\right)^{-1} \] (71)

The terms \( E_{eqv} \) (is the equivalent error vector), \( D_{eqv} \) and \( A \) are matrices of size \((N \times 1)\), \((M \times N)\) and \((M \times 1)\) respectively. Now matrix \( A \) can be replaced with scalar product of \( e_{eqv} \) and \( D_{eqv} \)

\[ A = D_{eqv} \cdot e_{eqv} \] (72)

With equation (72), we can write equations (23) and (24) as

\[ \left(\operatorname{cSE} / \partial c_i'\right) = \left(D_{eqv} \cdot e_{eqv}\right) \cdot \left(2 \cdot \left(z' / b\right) \cdot \left(x_i - c_i'\right) / \left|\sigma_i'\right|^2\right) \] (73)

\[ \left(\operatorname{cSE} / \partial \sigma_i'\right) = \left(D_{eqv} \cdot e_{eqv}\right) \cdot \left(2 \cdot \left(z' / b\right) \cdot \left(x_i - c_i'\right) / \left|\sigma_i'\right|^3\right) \] (74)

Now, by considering normalized equivalent error like in (61), taking into account the equation (54) and comparing it respectively with (73) and (74), the transposed Jacobian matrix, the Jacobian for the parameters \( c_i' \) and \( \tau_i' \) can be computed as:

\[ J^T(c_i') = \left[2 \cdot D_{eqv} \cdot z'(x_i - c_i') / \left|\sigma_i'\right|^2\right] \] (75)

\[ J(c_i') = \left[2 \cdot D_{eqv} \cdot z'(x_i - c_i') / \left|\sigma_i'\right|^3\right] \] (76)

\[ J^T(\sigma_i') = \left[2 \cdot D_{eqv} \cdot z'(x_i - c_i') / \left|\sigma_i'\right|^2\right] \] (77)

\[ J(\sigma_i') = \left[2 \cdot D_{eqv} \cdot z'(x_i - c_i') / \left|\sigma_i'\right|^3\right] \] (78)

It is to be noted that normalized prediction error is considered for computation of Jacobian matrices for the free parameters \( W_{ij}^t \) and \( W_{ij}^t \). Meanwhile normalized equivalent error has been considered for the computation of transposed Jacobian matrices and their Jacobian matrices respectively for the free parameters mean \( c_i' \) and variance \( \sigma_i' \).

**Modelling of non-linear dynamics data time-series**

The time series with lead time predicts the values at time \((t + L)\), based on the available time series data up to the point \( t \). To forecast the electrical load time series using the neuro-fuzzy approach, the time series data \( X = \{X_1, X_2, X_3, \ldots, X_p\} \) can be rearranged in a MIMO - XIO like structure. XIO stands for the time series \( X \) is represented in Input and Output form. For the given time series modeling and forecasting application the MIMO neuro-fuzzy predictor to be developed is supposed to operate with four inputs \((n = 4)\) and three outputs \((m = 3)\). Now if the sampling interval and the lead time of forecast both are taken to be 1 time unit, it means if 1 time unit means 1 hour (this is only an example, because another 1 time unit can equal to 15 minutes), so this MIMO system was taking 4 hours as an input data and predicted 3 hours in advance, then for each \( t \geq 4 \), the input data in this case represent a four dimensional vector and output data a three dimensional vector as described below.

\[ XI = [X(t - 3), X(t - 2), X(t - 1), X(t)] \] (79)

\[ XO = [X(t + 1), X(t + 2), X(t + 3)] \] (80)

Thus, in order to have sequential output in each row the values of \( t \) run as \( 4, 7, 10, 13 \ldots, (p-6) \); and so on, so that the XIO matrix will propose an sort-term and offline/Batch mode scenario, which is look like in equation (81). In this equation, the first four columns of the XIO matrix represent the four inputs to the network whereas, last three columns represent the output from the neuro-fuzzy network.

\[ \begin{bmatrix}
X_1 & X_2 & X_3 & X_4 & X_1 & X_2 & X_3 & X_4 & \ldots & \ldots & \ldots & \ldots \\
X_4 & X_5 & X_6 & X_7 & X_4 & X_5 & X_6 & X_7 & \ldots & \ldots & \ldots & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\
X_{p-4} & X_{p-5} & X_{p-6} & X_{p-7} & X_{p-4} & X_{p-5} & X_{p-6} & X_{p-7} & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix} \] (81)
EXPERIMENTS AND RESULTS OF BPA AND LMA

The training and forecasting performances, which produced by NF-network, are illustrated in Figures 2a-2c and 3a-3c where the XIO matrix is shown in equation (81). There are 3450 electrical load data from electrical load data used in training and forecasting, for which 1500 data are for training and the remaining 1950 data are for forecasting. Because the first parameters are random, the initial $SSE$ in BPA starts from 979.98 and the final $SSE$ train becomes 11.8270 by using 1000 epochs. In the other case of LMA, the initial $SSE$ starts with 558.02 and goes to final $SSE$ train=9.2764 using only 200 epochs.

Figure 2a. Graphic $SSE$ vs. Epochs of TS-type MIMO NF networks with BPA, $n= 4$ Inputs, $m= 3$ outputs, $M=15$, Epochs= 1000

Figure 2b. Training Performance of TS-type MIMO NF network with BPA, $n= 4$ Inputs, $m= 3$ outputs, $M=15$, Epochs= 1000, Lead Time $d=1$, Learning Rate $\eta=0.0012$, Wildness Factor WF=1.005, Gamma $\gamma=0.0001$, Momentum $\mu= 0.05$, Final SSEtrain=28.0664 with 3450 data Training and Forecasting

Figure 2c. Training and Forecasting Performance of TS-type MIMO NF network with BPA, $n= 4$ Inputs, $m= 3$ outputs, $M=15$, Epochs= 1000, Lead Time $d=1$, Learning Rate $\eta=0.0012$, Wildness Factor WF=1.005, Gamma $\gamma=0.0001$, Momentum $\mu= 0.05$, Final SSEtotal=28.0664 with 3450 data Training and Forecasting

Figure 3a. SSE training vs. Epochs of TS-type MIMO NF networks with LMA, $n= 4$ Inputs, $m= 3$ outputs, $M=15$, Epochs= 200

Figure 3b. illustrates the proposed accelerated LMA which brings the performance similar to the proposed BPA using only 200 epochs, indicating higher convergence speed in comparison with the BPA. In addition, the performances in both Figures 2 and 3 shows that the training has small oscillation because of the implementation of Wildness Factor (WF) which only allow 0.1% of oscillation.

From Figure 3a, it can be seen that LMA has much faster training compared to BPA. LMA needs only around 100 epochs to achieve the performance, whereas BPA needs 1000 epochs. See Table 1 for detailed comparison between these two methods.
The result shows that $SSE_1$, $SSE_2$ and $SSE_3$ indicate the sum squared error at output 1, 2 and 3, respectively, of the Takagi-Sugeno–type MIMO NF network, and $SSE$ indicates the summing of all $SSE$ values which are contributed by all outputs. The similar definition is also applied to $MSE_1$, $MSE_2$, $MSE_3$, $RMSE_1$, $RMSE_2$, and $RMSE_3$.

**Table 1. Comparison between BPA and LMA on Electrical Load training and forecasting performance of Takagi-Sugeno-type MIMO-NF network with 4 Inputs and 3 Outputs**

| Description | Performance using BPA (Normalized Data) | Performance using LMA (Normalized Data) |
|-------------|-----------------------------------------|----------------------------------------|
| No of Epochs = 1000 | Initial SSE= 979.98 | Initial SSE= 558.02 |
| Training Data | | |
| | SSEtrain = 11.8270 | SSEtrain = 9.2764 |
| | SSE1 = 1.1998 | SSE1 = 0.8777 |
| | SSE2 = 4.0697 | SSE2 = 2.6833 |
| | SSE3 = 6.5575 | SSE3 = 5.7513 |
| | MSEtrain = 0.0158 | MSEtrain = 0.0124 |
| | MSE1 = 0.0048 | MSE1 = 0.0035 |
| | MSE2 = 0.0163 | MSE2 = 0.0107 |
| | MSE3 = 0.0263 | MSE3 = 0.0229 |
| | RMSE1 = 0.0693 | RMSE1 = 0.0592 |
| | RMSE2 = 0.128 | RMSE2 = 0.1034 |
| | RMSE3 = 0.1622 | RMSE3 = 0.1513 |
| Electrical Load | | |
| | SSEtotal = 28.0664 | SSEtotal = 22.2786 |
| | SSE1 = 3.3012 | SSE1 = 2.5575 |
| | SSE2 = 10.1478 | SSE2 = 6.6040 |
| | SSE3 = 14.6173 | SSE3 = 13.1171 |
| | MSEtotal = 0.0163 | MSEtotal = 0.0129 |
| | RMSEtotal = 0.128 | RMSEtotal = 0.1136 |

**CONCLUSION REMARKS**

In the paper neuro-fuzzy approaches with two types of training algorithms have been presented for short-term forecasting of electrical load. Performance results from Section 4 proved that the trained NF network using LMA is found to be very efficient in modeling and prediction of the various nonlinear dynamics, compared to BPA. An efficient training algorithm based on combination of LMA with additional modified error index extension (MEI) and adaptive version of learning rate (momentum) have been developed to train the Takagi-Sugeno type multi-input multi-output (MIMO) and multi-input single-output (MISO) Neuro-fuzzy network, improving the training performance. Several issues need to be addressed in the future works which includes mainly transparency and interpretability of generated fuzzy model (rules). For the latter issue set theoretic similarity measures [7] should be computed for each pair of fuzzy sets and the fuzzy sets which are highly similar should be merged together into a single one.
REFERENCES

[1] Jang JSR., ANFIS: Adaptive network Based Fuzzy Inference System, IEEE Trans. On SMC., 23(3):665-685, 1993.

[2] Palit AK, Babuška R., Efficient training algorithm for Takagi-Sugeno type Neuro-Fuzzy network, Proc. of FUZZ-IEEE, Melbourne, Australia, vol. 3: 1538-1543, 2001.

[3] Palit AK, Popovic D., Nonlinear combination of forecasts using ANN, FL and NF approaches, FUZZ-IEEE, 2:566-57, 2002.

[4] Palit AK, Popovic D., Computational Intelligence in Time Series Forecasting, Theory and Engineering Applications, Springer, 2005.

[5] Xiaosong D, Popovic D, Schulz-Ekloff. Oscillation resisting in the learning of Backpropagation neural networks, Proc. of 3rd IFAC/IFIP, Belgium, 1995.

[6] Pasila Felix, Forecasting of Electrical Load using Takagi-Sugeno type MIMO Neuro-Fuzzy network, Master Thesis, University of Bremen, 2006.

[7] Setnes M, Babuška R, Kaymark U., Similarity measures in Fuzzy rule base simplification, IEEE Transaction on System, Man and Cybernetics, vol.28:771-775, 1998.