Int. J. Appl. Math. Comput. Sci., 2005, Vol. 15, No. 1, 115–123

THE UD RLS ALGORITHM FOR TRAINING FEEDFORWARD NEURAL NETWORKS

JAROSŁAW BILSKI

Department of Computer Engineering
Technical University of Częstochowa
ul. Armii Krajowej 36, 42–200 Częstochowa, Poland
e-mail: bilski@kik.pcz.czest.pl

A new algorithm for training feedforward multilayer neural networks is proposed. It is based on recursive least squares procedures and U-D factorization, which is a well-known technique in filter theory. It will be shown that due to the U-D factorization method, our algorithm requires fewer computations than the classical RLS applied to feedforward multilayer neural network training.

Keywords: neural networks, learning algorithms, recursive least squares method, UD factorization

1. Introduction

Feedforward multilayer neural networks (FMNNs) are widely used to solve various problems in system modelling and identification, prediction, nonlinear signal processing and pattern classification. In practice, the classical method for training FMNNs is the back-propagation, its momentum version and some modifications (Abid, *et al.*, 2001; Chen, 1992; Joost and Schiffmann, 1998; Korbicz *et al.*, 1994; Perantonis and Karras, 1995). Since back-propagation may converge to local minima, in the past decade several other methods have been proposed for training FMNN's. Conjugate gradient-based algorithms (Bishop, 1995; Moller, 1993), second order-algorithms (Ampazis and Perantonis, 2002; Bojarczak and Stodolski, 1996; Lera and Pinzolas, 2002), recursive least-squares methods (Azimi-Sadjadi and Liou, 1992; Bilski and Rutkowski, 1998; 2003) and extended Kalman filter (EKF) techniques (Leung *et al.*, 2001; Sum *et al.*, 1998; 1999; Zhang and Li, 1999) should be mentioned here. Despite so many techniques, a further improvement is highly desirable as regards learning accuracy, computational complexity, numerical stability and generalization capability.

In this paper a new algorithm for training FMNNs is proposed. It is based on recursive least squares-procedures and U-D factorization, which is a well-known technique in filter theory (Wellstead and Zarrop, 1991). It will be shown that due to the U-D factorization method, our algorithm requires fewer computations than the classical RLS (Rutkowski, 1994; Strobach, 1990) applied to FMNN training. Moreover, it outperforms the classical RLS in terms of the convergence rate. In the paper the algorithm is derived for two different cases: the error is determined in the linear part of the neurons (Error Transferred Back – ETB) and, as usual, in back-propagation neural networks. Simulation results will be given to demonstrate the efficiency and effectiveness of the proposed learning algorithm. The paper is organized as follows: In Section 2 the terminology used in the paper is introduced. In Section 3 the UD RLS algorithm for FMNNs with a linear activation function is derived. In Section 4 the results are easily generalized to FMNNs with nonlinear activation functions. In Section 5 the performance of the new learning algorithms is investigated on typical benchmarks.

2. Terminology

In the sequel, the following terminology will be used:

L – the number of layers in the network,

N_k – the number of neurons in the k-th layer, $k = 1, \ldots, L$,

N_0 – the numbers of inputs of neural networks,

$\boldsymbol{u} = [u_1, \ldots, u_{N_0}]^T$ – the input signal vector of the neural network,

$y_i^{(k)}$ – the output signal of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$$y_i^{(k)}(n) = f\left(s_i^{(k)}(n)\right),$$

$\boldsymbol{y}^{(k)} = [y_1^{(k)}, \ldots, y_{N_k}^{(k)}]^T$ – the output signal vector in the k-th layer, $k = 1, \ldots, L$,

$x_i^{(k)}$ – the i-th input, $i = 0, \ldots, N_{k-1}$, for the k-th layer, $k = 1, \ldots, L$, where

$$x_i^{(k)} = \begin{cases} u_i & \text{for } k = 1, \\ y_i^{(k-1)} & \text{for } k = 2, \ldots, L, \\ +1 & \text{for } i = 0, k = 1, \ldots, L, \end{cases}$$

$\boldsymbol{x}^{(k)} = [x_0^{(k)}, \ldots, x_{N_{k-1}}^{(k)}]^T$ – the input signal vector for the k-th layer, $k = 1, \ldots, L$,

$w_{ij}^{(k)}(n)$ – the weight of the i-th neuron, $i = 1, \ldots, N_k$, of the k-th layer, $k = 1, \ldots, L$, connecting this neuron with the j-th input $x_j^{(k)}$, $j = 0, \ldots, N_{k-1}$,

$\boldsymbol{w}_i^{(k)} = [w_{i0}^{(k)}, \ldots, w_{iN_{k-1}}^{(k)}]^T$ – the weight vector of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$\boldsymbol{W}^{(k)} = [\boldsymbol{w}_1^{(k)}, \ldots, \boldsymbol{w}_{N_k}^{(k)}]$ – the weight matrix in the k-th layer, $k = 1, \ldots, L$,

$s_i^{(k)}(n) = \sum_{j=0}^{N_{k-1}} w_{ij}^{(k)}(n)\, x_j^{(k)}(n)$ – the linear output of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$\boldsymbol{s}^{(k)} = [s_1^{(k)}, \ldots, s_{N_k}^{(k)}]^T$ – the linear output vector in the k-th layer, $k = 1, \ldots, L$,

$d_i^{(k)}$ – the desired output of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$\boldsymbol{d}^{(k)} = [d_1^{(k)}, \ldots, d_{N_k}^{(k)}]^T$ – the desired output vector in the layer k, $k = 1, \ldots, L$,

$b_i^{(k)} = f^{-1}(d_i^{(k)})$ – the desired linear summation output of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$\boldsymbol{b}^{(k)} = [b_1^{(k)}, \ldots, b_{N_k}^{(k)}]^T$ – the desired linear summation output vector in layer k, $k = 1, \ldots, L$,

$\varepsilon_i^{(k)}(n) = d_i^{(k)}(n) - y_i^{(k)}(n)$ – the error of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$\varepsilon^{(k)} = [\varepsilon_1^{(k)}, \ldots, \varepsilon_{N_k}^{(k)}]^T$ – the error signal vector in the k-th layer, $k = 1, \ldots, L$,

$e_i^{(k)}(n) = b_i^{(k)}(n) - f^{-1}(y_i^{(k)}(n))$ – the error of the linear part of the i-th neuron, $i = 1, \ldots, N_k$, in the k-th layer, $k = 1, \ldots, L$,

$\boldsymbol{e}^{(k)} = [e_1^{(k)}, \ldots, e_{N_k}^{(k)}]^T$ – the linear error vector in the k-th layer, $k = 1, \ldots, L$,

λ – the forgetting factor in the RLS algorithm,

μ – the learning coefficient of the back-propagation (BP) algorithm,

α – the momentum coefficient of the momentum back-propagation (MBP) algorithm,

δ – a positive constant.

In Fig. 1 a model of the i-th neuron in the k-th layer is shown.

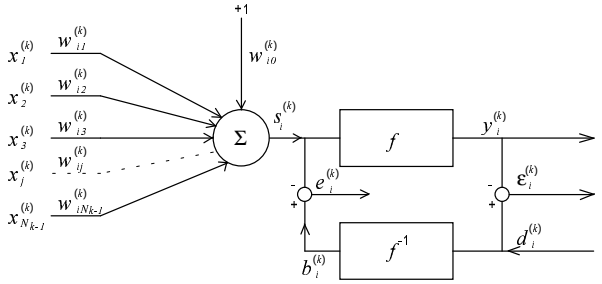

Fig. 1. Model of the i-th neuron in the k-th layer.

3. UD RLS for Multilayer Networks with Linear Activation Functions

In this section it is assumed that the activation function has the following form:

$$f_1(s) = a\, s, \quad a > 0. \tag{1}$$

The minimization criterion for the multilayer network is

$$\boldsymbol{Q}(n) = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \varepsilon_j^{(L)^2}(t)$$

$$= \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \left[d_j^{(L)}(t) - a\boldsymbol{x}^{(L)^T}(t)\, \boldsymbol{w}_j^{(L)}(n) \right]^2. \tag{2}$$

The parameter λ allows us to discard the oldest data (see, e.g., Strobach, 1990). From this criterion, by solving normal equations, the conventional RLS algorithm (Bilski, 1995) is obtained:

$$\varepsilon_i^{(k)}(n) = d_i^{(k)}(n) - a\boldsymbol{x}^{(k)^T}(n)\, \boldsymbol{w}_i^{(k)}(n-1)$$
$$= d_i^{(k)}(n) - y_i^{(k)}(n), \tag{3}$$

$$\boldsymbol{g}^{(k)}(n) = \frac{a\boldsymbol{P}^{(k)}(n-1)\, \boldsymbol{x}^{(k)}(n)}{\lambda + a^2 \boldsymbol{x}^{(k)^T}(n)\, \boldsymbol{P}^{(k)}(n-1)\, \boldsymbol{x}^{(k)}(n)}, \tag{4}$$

$$\boldsymbol{P}^{(k)}(n) = \lambda^{-1} \left[I - a\boldsymbol{g}^{(k)}(n)\, \boldsymbol{x}^{(k)^T}(n) \right] \times \boldsymbol{P}^{(k)}(n-1), \tag{5}$$

$$\boldsymbol{w}_i^{(k)}(n) = \boldsymbol{w}_i^{(k)}(n-1) + \boldsymbol{g}^{(k)}(n)\, \varepsilon_i^{(k)}(n). \tag{6}$$

Now, the algorithm (3)–(6) can be modified by making the assumption that the matrix $\boldsymbol{P}$ is factorized as the product of upper triangular and diagonal matrices (Strobach, 1990; Wellstead and Zarrop, 1991). For simplicity, all transformations will be derived for a single network layer, and therefore the layer index (k) is omitted. The factorization is given as follows:

$$\boldsymbol{P}(n) = \boldsymbol{U}(n)\, \boldsymbol{D}(n)\, \boldsymbol{U}^T(n), \tag{7}$$

where

$$\boldsymbol{U}(n)=\begin{bmatrix} 1 & u_{01}(n) & u_{02}(n) & \cdots & u_{0N_0}(n) \\ 0 & 1 & u_{12}(n) & \cdots & u_{1N_0}(n) \\ 0 & 0 & 1 & & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{N_0-1\,N_0}(n) \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{8}$$

and

$$\boldsymbol{D}(n)=\begin{bmatrix} c_0 & 0 & \cdots & 0 \\ 0 & c_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{N_0} \end{bmatrix}. \tag{9}$$

$\boldsymbol{U}$ is a triangular matrix with zeros in bottom elements and ones on the main diagonal, whereas $\boldsymbol{D}$ is a diagonal matrix.

Defining

$$\boldsymbol{f}=\boldsymbol{U}^T(n-1)\,\boldsymbol{x}(n), \tag{10}$$

$$\boldsymbol{h}=\boldsymbol{D}(n-1)\,\boldsymbol{f}, \tag{11}$$

and denoting the denominator of (4) by β, we have

$$\begin{aligned}\beta &= \lambda + a^2\boldsymbol{x}^T(n)\boldsymbol{P}(n-1)\boldsymbol{x}(n) \\ &= \lambda + a^2\boldsymbol{x}^T(n)\boldsymbol{U}(n-1)\boldsymbol{D}(n-1)\boldsymbol{U}^T(n-1)\boldsymbol{x}(n) \\ &= \lambda + a^2\boldsymbol{f}^T\boldsymbol{D}(n-1)\boldsymbol{f} = \lambda + a^2\boldsymbol{f}^T\boldsymbol{h},\end{aligned} \tag{12}$$

and Eqns. (4) and (5) can be expressed as

$$\begin{aligned}\boldsymbol{g}(n) &= a\boldsymbol{P}(n-1)\,\boldsymbol{x}(n)\,\beta^{-1} \\ &= a\boldsymbol{U}(n-1)\,\boldsymbol{D}(n-1)\,\boldsymbol{U}^T(n-1)\,\boldsymbol{x}(n)\,\beta^{-1} \\ &= a\boldsymbol{U}(n-1)\,\boldsymbol{D}(n-1)\,\boldsymbol{f}\beta^{-1} \\ &= a\boldsymbol{U}(n-1)\,\boldsymbol{h}\beta^{-1}\end{aligned} \tag{13}$$

and

$$\begin{aligned}\boldsymbol{P}(n) &= \lambda^{-1}\Big[I - a^2\beta^{-1}\boldsymbol{U}(n-1)\,\boldsymbol{h}\boldsymbol{x}^{\mathrm{T}}(n)\Big]\boldsymbol{P}(n-1) \\ &= \lambda^{-1}\Big[\boldsymbol{P}(n-1) \\ &\quad - a^2\beta^{-1}\boldsymbol{U}(n-1)\,\boldsymbol{h}\boldsymbol{x}^{\mathrm{T}}(n)\,\boldsymbol{P}(n-1)\Big] \\ &= \lambda^{-1}\Big[\boldsymbol{U}(n-1)\,\boldsymbol{D}(n-1)\,\boldsymbol{U}^T(n-1) \\ &\quad - a^2\beta^{-1}\boldsymbol{U}(n-1)\,\boldsymbol{h}\boldsymbol{x}^{\mathrm{T}}(n)\,\boldsymbol{U}(n-1) \\ &\quad \times \boldsymbol{D}(n-1)\,\boldsymbol{U}^T(n-1)\Big] \\ &= \lambda^{-1}\boldsymbol{U}(n-1)\Big[\boldsymbol{D}(n-1) \\ &\quad - a^2\beta^{-1}\boldsymbol{h}\boldsymbol{f}^T\boldsymbol{D}(n-1)\Big]\boldsymbol{U}^T(n-1) \\ &= \lambda^{-1}\boldsymbol{U}(n-1)\Big[\boldsymbol{D}(n-1) - a^2\beta^{-1}\boldsymbol{h}\boldsymbol{h}^T\Big] \\ &\quad \times \boldsymbol{U}^T(n-1) \\ &= \boldsymbol{U}(n)\,\boldsymbol{D}(n)\,\boldsymbol{U}^T(n).\end{aligned} \tag{14}$$

Hence, through the substitution

$$\bar{\boldsymbol{U}}\bar{\boldsymbol{D}}\bar{\boldsymbol{U}}^T = \boldsymbol{D}(n-1) - a^2\beta^{-1}\boldsymbol{h}\boldsymbol{h}^T, \tag{15}$$

the following formulae are obtained:

$$\boldsymbol{U}(n) = \boldsymbol{U}(n-1)\bar{\boldsymbol{U}}, \tag{16}$$

$$\boldsymbol{D}(n) = \bar{\boldsymbol{D}}\lambda^{-1}, \tag{17}$$

where

$$\begin{aligned}\bar{\boldsymbol{U}}(n) &= \begin{bmatrix} \bar{\boldsymbol{u}}_0 & \bar{\boldsymbol{u}}_1 & \cdots & \bar{\boldsymbol{u}}_{N_0} \end{bmatrix} \\ &= \begin{bmatrix} 1 & \bar{u}_{01}(n) & \bar{u}_{02}(n) & \cdots & \bar{u}_{0N_0}(n) \\ 0 & 1 & \bar{u}_{12}(n) & \cdots & \bar{u}_{1N_0}(n) \\ 0 & 0 & 1 & & \vdots \\ \vdots & \vdots & \vdots & \ddots & \bar{u}_{N_0-1N_0}(n) \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}\end{aligned} \tag{18}$$

and

$$\bar{\boldsymbol{D}}(n)=\begin{bmatrix} \bar{c}_0 & 0 & \cdots & 0 \\ 0 & \bar{c}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \bar{c}_{N_0} \end{bmatrix}. \tag{19}$$

The new algorithm is obtained by solving (15)–(17). Equation (15) can be written in the form

$$\sum_{i=0}^{N_0}\bar{c}_i\bar{\boldsymbol{u}}_i\bar{\boldsymbol{u}}_i^T = \sum_{i=0}^{N_0}c_i\boldsymbol{e}_i\boldsymbol{e}_i^T - a^2\beta^{-1}\boldsymbol{h}\boldsymbol{h}^T, \tag{20}$$

where $c_i = c_i(n-1)$ and $\boldsymbol{e}_i$ is the i-th versor. Introducing the symbols

$$\beta_{N_0} = \beta, \qquad \beta_m = \lambda + a^2\sum_{i=0}^{m} f_i h_i, \tag{21}$$

$$\boldsymbol{v}_{N_0} = \boldsymbol{h}, \qquad \boldsymbol{v}_{m-1} = \begin{bmatrix} v_{m0} \\ \vdots \\ v_{mm-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{22}$$

Eqn. (20) can be transformed to the form

$$\sum_{i=0}^{N_0} \bar{c}_i \bar{\boldsymbol{u}}_i \bar{\boldsymbol{u}}_i^T - \sum_{i=0}^{N_0} c_i \boldsymbol{e}_i \boldsymbol{e}_i^T + a^2 \beta^{-1} \boldsymbol{v}_{N_0} \boldsymbol{v}_{N_0}^T = \boldsymbol{0}, \quad (23)$$

$$\sum_{i=0}^{N_0-1} \bar{c}_i \bar{\boldsymbol{u}}_i \bar{\boldsymbol{u}}_i^T - \sum_{i=0}^{N_0-1} c_i \boldsymbol{e}_i \boldsymbol{e}_i^T + \bar{c}_{N_0} \bar{\boldsymbol{u}}_{N_0} \bar{\boldsymbol{u}}_{N_0}^T - c_{N_0} \boldsymbol{e}_{N_0} \boldsymbol{e}_{N_0}^T + a^2 \beta_{N_0}^{-1} \boldsymbol{v}_{N_0} \boldsymbol{v}_{N_0}^T = \boldsymbol{0}, \quad (24)$$

$$\sum_{i=0}^{N_0-1} c_i \boldsymbol{u}_i \boldsymbol{u}_i^T - \sum_{i=0}^{N_0-1} c_i \boldsymbol{e}_i \boldsymbol{e}_i^T + \boldsymbol{M}_{N_0} = \boldsymbol{0}. \quad (25)$$

It can be easily noticed that matrices under the summation sign have zeros in the N_0-th row and in the N_0-th column. So, in order to meet (25), an identical situation for the following matrix $\boldsymbol{M}_{N_0}$ must occur:

$$\boldsymbol{M}_{N_0} = \bar{c}_{N_0} \bar{\boldsymbol{u}}_{N_0} \bar{\boldsymbol{u}}_{N_0}^T - c_{N_0} \boldsymbol{e}_{N_0} \boldsymbol{e}_{N_0}^T + a^2 \beta_{N_0}^{-1} \boldsymbol{v}_{N_0} \boldsymbol{v}_{N_0}^T. \quad (26)$$

It can be obtained by setting

$$\bar{u}_{N_0 N_0} = 1, \quad (27)$$

$$\bar{c}_{N_0} = c_{N_0} - a^2 \beta_{N_0}^{-1} v_{N_0 N_0}^2, \quad (28)$$

$$\bar{c}_{N_0} \bar{u}_{i N_0} \bar{u}_{N_0 N_0} = -a^2 \beta_{N_0}^{-1} v_{N_0 N_0} v_{N_0 i}. \quad (29)$$

From (27) and (29) we thus get

$$\bar{u}_{i N_0} = \frac{-a^2 v_{N_0 N_0} v_{N_0 i}}{\beta_{N_0} \bar{c}_{N_0}}. \quad (30)$$

By substituting (27), (28) and (30) into (26), the matrix $\boldsymbol{M}_{N_0}$ takes the form

$$\boldsymbol{M}_{N_0} = \bar{c}_{N_0} \frac{a^4 v_{N_0 N_0}^2}{\bar{c}_{N_0}^2 \beta_{N_0}^2} \boldsymbol{v}_{N_0-1} \boldsymbol{v}_{N_0-1}^T + a^2 \beta_{N_0}^{-1} \boldsymbol{v}_{N_0-1} \boldsymbol{v}_{N_0-1}^T = \left[\frac{a^4 v_{N_0 N_0}^2}{\bar{c}_{N_0} \beta_{N_0}^2} + \frac{a^2}{\beta_{N_0}} \right] \boldsymbol{v}_{N_0-1} \boldsymbol{v}_{N_0-1}^T \quad (31)$$

or, after transformations (cf. (21), (22) and (28)),

$$\frac{a^4 v_{N_0 N_0}^2}{\bar{c}_{N_0} \beta_{N_0}^2} + \frac{a^2}{\beta_{N_0}} = a^2 \frac{a^2 v_{N_0 N_0}^2 + c_{N_0} \beta_{N_0} - a^2 \beta_{N_0}^{-1} v_{N_0 N_0}^2 \beta_{N_0}}{\bar{c}_{N_0} \beta_{N_0}^2} = a^2 \frac{c_{N_0} \beta_{N_0}}{\bar{c}_{N_0} \beta_{N_0}^2} = a^2 \frac{c_{N_0}}{c_{N_0} \beta_{N_0} - a^2 \beta_{N_0}^{-1} v_{N_0 N_0}^2 \beta_{N_0}} = a^2 \frac{1}{\beta_{N_0} - \frac{a^2}{c_{N_0}} v_{N_0 N_0}^2} = a^2 \frac{1}{\beta_{N_0-1}}, \quad (32)$$

the matrix $\boldsymbol{M}_{N_0}$ can be expressed in the simplified form

$$\boldsymbol{M}_{N_0} = \frac{a^2}{\beta_{N_0-1}} \boldsymbol{v}_{N_0-1} \boldsymbol{v}_{N_0-1}^T, \quad (33)$$

and (23) can be written as

$$\sum_{i=1}^{N_0-1} \bar{c}_i \bar{\boldsymbol{u}}_i \bar{\boldsymbol{u}}_i^T - \sum_{i=1}^{N_0-1} c_i \boldsymbol{e}_i \boldsymbol{e}_i^T + a^2 \beta_{N_0-1}^{-1} \boldsymbol{v}_{N_0-1} \boldsymbol{v}_{N_0-1}^T = \boldsymbol{0}. \quad (34)$$

Observe that Eqns. (23) and (32) differ from each other only by the summation range and indices. Repeating the same arguments for indices changing from $N_0 - 1$ to 1, as those in the transformations (23)–(34), it is possible to calculate all the values of c_i and u_{ij}.

Equation (30) is transformed using (21), (22) and (28) as follows:

$$\bar{u}_{i N_0} = \frac{-a^2 v_{N_0 N_0} v_{N_0 i}}{\beta_{N_0} \bar{c}_{N_0}} = \frac{-a^2 v_{N_0 N_0} v_{N_0 i}}{\beta_{N_0} \left(c_{N_0} - \frac{a^2}{\beta_{N_0}} v_{N_0 N_0}^2 \right)} = \frac{-a^2 v_{N_0 N_0} v_{N_0 i}}{c_{N_0} \left(\beta_{N_0} - \frac{a^2}{c_{N_0}} v_{N_0 N_0}^2 \right)} = \frac{-a^2 v_{N_0 N_0} v_{N_0 i}}{c_{N_0} \beta_{N_0-1}}. \quad (35)$$

Substituting

$$\mu_{N_0} = \frac{-a^2 v_{N_0 N_0}}{c_{N_0} \beta_{N_0-1}} = \frac{-a^2 f_{N_0}}{\beta_{N_0-1}}, \quad (36)$$

a simpler form,

$$\bar{u}_{i N_0} = \mu_{N_0} v_{N_0 i}, \quad (37)$$

is obtained. The values of the matrix $\boldsymbol{D}$ are calculated as follows (cf. (21), (22) and (28)):

$$c_i(n) = \bar{c}_i \lambda^{-1} = \left[c_i - \frac{a^2 v_{ii}^2}{\beta_i} \right] \lambda^{-1} = c_i \frac{1}{\beta_i} \left[\beta_i - \frac{a^2 v_{ii}^2}{c_i} \right] \lambda^{-1} = c_i \frac{\beta_{i-1}}{\beta_i \lambda}. \quad (38)$$

The numerator of (4) or (13) takes the following form:

$$\boldsymbol{k}(n) = a \boldsymbol{U}(n-1) \boldsymbol{h} = a \boldsymbol{U}(n-1) \boldsymbol{v}_{N_0}. \quad (39)$$

Hence

$$k_i = \sum_{m=i}^{N_0} u_{im}(n-1) v_{N_0 m} = v_{N_0 i} + \sum_{m=i+1}^{N_0} u_{im}(n-1) v_{N_0 m} = \sum_{m=i}^{N_0-1} u_{im}(n-1) v_{N_0 m} + u_{i N_0}(n-1) v_{N_0 N_0} \quad (40)$$

or, in a recurrent form,

$$k_{i,\text{new}} = k_{i,\text{old}} + u_{iN_0}(n-1)\, v_{N_0 N_0}. \tag{41}$$

The following is calculated from (16):

$$\begin{aligned} u_{ij}(n) &= \sum_{m=i}^{j} u_{im}(n-1)\,\bar{u}_{mj} \\ &= u_{ij}(n-1) + \sum_{m=i}^{j-1} u_{im}(n-1)\,\mu_j v_{jm} \\ &= u_{ij}(n-1) + \mu_j \sum_{m=i}^{j-1} u_{im}(n-1)\, v_{jm} \\ &= u_{ij}(n-1) + \mu_j k_i. \end{aligned} \tag{42}$$

All the transformations that have been carried out lead to a new UD RLS algorithm for a single layer neural network. For a multilayer neural network, the learning algorithm differs only by the index (k), which refers to the number of the k-th layer. Hence, by analogy, the corresponding UD RLS algorithm for a multilayer neural network can be written. First, errors and required values are calculated for all neurons in the network by using the backpropagation method. Next, all the weights for subsequent layers are updated using the algorithm (43)–(54). We have

$$\begin{aligned} \varepsilon_i^{(k)}(n) &= d_i^{(k)}(n) - a\boldsymbol{x}^{(k)^T}(n)\,\boldsymbol{w}_i^{(k)}(n-1) \\ &= d_i^{(k)}(n) - y_i^{(k)}(n), \end{aligned} \tag{43}$$

$$\boldsymbol{f} = \boldsymbol{U}^{(k)^T}(n-1)\,\boldsymbol{x}^{(k)}(n), \tag{44}$$

$$\boldsymbol{h} = \boldsymbol{D}^{(k)}(n-1)\,\boldsymbol{f}, \tag{45}$$

$$\beta_{-1} = \lambda. \tag{46}$$

For j from 0 to N_{k-1} we set

$$\beta_j = \beta_{j-1} + a^2 f_j h_j, \tag{47}$$

$$c_j^{(k)}(n) = c_j^{(k)}(n-1)\,\frac{\beta_{j-1}}{\beta_j \lambda}, \tag{48}$$

$$k_j = h_j, \tag{49}$$

$$\mu_j = \frac{-a^2 f_j}{\beta_{j-1}}. \tag{50}$$

For m from 0 to $j-1 (j>0)$ we write

$$u_{mj}^{(k)}(n) = u_{mj}^{(k)}(n-1) + \mu_j k_m, \tag{51}$$

$$k_m = k_m + u_{mj}^{(k)}(n-1)\, k_j. \tag{52}$$

Finally,

$$\boldsymbol{g}^{(k)}(n) = \frac{\left[k_0, \ldots, k_{N_{k-1}}\right]^T}{\beta_{N_{k-1}}}, \tag{53}$$

$$\boldsymbol{w}_i^{(k)}(n) = \boldsymbol{w}_i^{(k)}(n-1) + \boldsymbol{g}^{(k)}(n)\,\varepsilon_i^{(k)}(n). \tag{54}$$

The initial values of matrices are set as

$$\begin{aligned} &\boldsymbol{D}^{(k)}(0) = \delta \boldsymbol{I}, \qquad \delta > 0, \\ &\boldsymbol{U}^{(k)}(0) = \boldsymbol{I}, \end{aligned} \tag{55}$$

where δ is a positive constant. The initial values of weights $\boldsymbol{w}_i^{(k)}(n)$ are chosen randomly.

4. UD RLS for Multilayer Networks with Non-Linear Activation Functions

In this section the results of Section 3 are generalized to the case of non-linear activation functions. The UD RLS algorithms are derived with the assumption that the errors are determined in the linear part of the neurons (the algorithm will be called ETB UD RLS, cf. Section 4.1) and, as usually, in the back-propagation method (cf. Section 4.2). A similar algorithm, based on UD factorization and Kalman filters, was studied in (Zhang and Li, 1999).

4.1. UD RLS with the Error Transferred Back (ETB) to the Linear Part of the Neuron

For any invertible activation function

$$y_i(n) = f\big(s_i(n)\big), \tag{56}$$

the desired output signal can be transferred back to the linear part of a neuron and then denoted by

$$b_i(n) = f^{-1}\big(d_i(n)\big). \tag{57}$$

In this case, the minimization criterion for the multilayer network takes the form

$$\begin{aligned} Q(n) &= \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} e_j^{(L)^2}(t) \\ &= \sum_{t=1}^{n} \lambda^{n-t} \\ &\quad \times \sum_{j=1}^{N_L} \left[b_j^{(L)}(t) - \boldsymbol{x}^{(L)^T}(t)\,\boldsymbol{w}_j^{(L)}(n) \right]^2. \end{aligned} \tag{58}$$

The errors of the linear part are determined by

$$e_p^{(k)}(t) = \sum_{j=1}^{N_{k+1}} w_{jp}^{(k+1)}(n)\,\frac{\partial y_p^{(k)}(t)}{\partial s_p^{(k)}(t)}\, e_j^{(k+1)}(t). \tag{59}$$

Finally, the ETB UD RLS method is obtained, as expressed by Eqns. (60) – (71). We compute

$$\varepsilon_i^{(k)}(n) = b_i^{(k)}(n) - \boldsymbol{x}^{(k)^T}(n)\,\boldsymbol{w}_i^{(k)}(n-1) = b_i^{(k)}(n) - s_i^{(k)}(n), \quad (60)$$

$$\boldsymbol{f} = \boldsymbol{U}^{(k)^T}(n-1)\,\boldsymbol{x}^{(k)}(n), \quad (61)$$

$$\boldsymbol{h} = \boldsymbol{D}^{(k)}(n-1)\,\boldsymbol{f}, \quad (62)$$

$$\beta_{-1} = \lambda. \quad (63)$$

For j from 0 to N_{k-1} we set

$$\beta_j = \beta_{j-1} + f_j h_j, \quad (64)$$

$$c_j^{(k)}(n) = c_j^{(k)}(n-1)\,\frac{\beta_{j-1}}{\beta_j \lambda}, \quad (65)$$

$$k_j = h_j, \quad (66)$$

$$\mu_j = \frac{-f_j}{\beta_{j-1}}. \quad (67)$$

For m from 0 to $j-1$ $(j>0)$ we write

$$u_{mj}^{(k)}(n) = u_{mj}^{(k)}(n-1) + \mu_j k_m, \quad (68)$$

$$k_m = k_m + u_{mj}^{(k)}(n-1)\,k_j. \quad (69)$$

Finally,

$$\boldsymbol{g}^{(k)}(n) = \frac{\left[k_0, \ldots, k_{N_{k-1}}\right]^T}{\beta_{N_{k-1}}}, \quad (70)$$

$$\boldsymbol{w}_i^{(k)}(n) = \boldsymbol{w}_i^{(k)}(n-1) + \boldsymbol{g}^{(k)}(n)\,\varepsilon_i^{(k)}(n). \quad (71)$$

The initial values are given by (55).

4.2. UD RLS with the Approximation of the Activation Function

For any differentiable activation function

$$y_i(n) = f\big(s_i(n)\big) \quad (72)$$

and the minimization criterion

$$Q(n) = \sum_{t=1}^{n} \lambda^{n-j} \sum_{j=1}^{N_L} {\varepsilon_j^{(L)}}^2(t) = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \Big[d_j^{(L)}(t) - f\left(\boldsymbol{x}^{(L)^T}(t)\,\boldsymbol{w}_j^{(L)}(n)\right)\Big]^2, \quad (73)$$

the conventional RLS algorithm can be modified as previously, leading to the UD RLS method. We start by setting

$$\varepsilon_i^{(k)}(n) = d_i^{(k)}(n) - y_i^{(k)}(n), \quad (74)$$

$$\boldsymbol{f}_i^{(k)} = \boldsymbol{U}_i^{(k)^T}(n-1)\,\boldsymbol{x}^{(k)}(n), \quad (75)$$

$$\boldsymbol{h}_i^{(k)} = \boldsymbol{D}_i^{(k)}(n-1)\,\boldsymbol{f}_i^{(k)}, \quad (76)$$

$$\beta_i^{(k)}{}_{-1} = \lambda. \quad (77)$$

For $j = 0, \ldots, N_{k-1}$, we compute

$$\beta_{i,j}^{(k)} = \beta_{i,j-1}^{(k)} + f'^2\left(s_i^{(k)}(n)\right) f_{i,j}^{(k)} h_{i,j}^{(k)}, \quad (78)$$

$$c_{i,j}^{(k)}(n) = c_{i,j}^{(k)}(n-1)\,\frac{\beta_{i,j-1}^{(k)}}{\beta_{i,j}^{(k)}\lambda}, \quad (79)$$

$$k_{i,j}^{(k)} = h_{i,j}^{(k)}, \quad (80)$$

$$\mu_{i,j}^{(k)} = -f'^2\left(s_i^{(k)}(n)\right)\frac{f_{i,j}^{(k)}}{\beta_{i,j-1}^{(k)}}. \quad (81)$$

For $m = 0, \ldots, j-1$ $(j>0)$, we set

$$u_{i,mj}^{(k)}(n) = u_{i,mj}^{(k)}(n-1) + \mu_{i,j}^{(k)} k_{i,m}^{(k)}, \quad (82)$$

$$k_{i,m}^{(k)} = k_{i,m}^{(k)} + u_{i,mj}^{(k)}(n-1)\,k_{i,j}^{(k)}. \quad (83)$$

Finally,

$$\boldsymbol{g}_i^{(k)}(n) = \frac{\left[k_{i,0}^{(k)}, \ldots, k_{i,N_{k-1}}^{(k)}\right]^T}{\beta_{i_{N_{k-1}}}}, \quad (84)$$

$$\boldsymbol{w}_i^{(k)}(n) = \boldsymbol{w}_i^{(k)}(n-1) + \boldsymbol{g}_i^{(k)}(n)\,\varepsilon_i^{(k)}(n). \quad (85)$$

The initial values are given by (55).

5. Performance Evaluations

The performance of RLS algorithms was tested on two typical benchmarks. In all simulations the learning algorithms run 100 times. The results are depicted in tables with entries showing both the average number of epochs required to meet a stopping criterion and the percentage of successful runs. In all these cases the best results are presented together with the corresponding parameters. In the case of unsuccessful runs, nothing is shown. In the tables, as was explained in Section 2, the following notation is used: λ – forgetting factor in the RLS algorithm, δ – initialization constant, μ – learning coefficient of the back-propagation (BP) algorithm, α – momentum coefficient of the momentum back-propagation (MBP) algorithm. The RLS and UD-RLS algorithms were compared

Table 1. Numerical results, part 1.

Structure / Algorithm	10–4–10		10–5–10		10–6–10	
BP	591.79 96	$\mu = 0.05$	373.3 100	$\mu = 0.04$	235.02 100	$\mu = 0.06$
MBP	570.86 98	$\mu = 0.03$ $\alpha = 0.35$	263.95 100	$\mu = 0.01$ $\alpha = 0.85$	193.88 100	$\mu = 0.01$ $\alpha = 0.85$
RLS	323.1 93	$\lambda = 0.997$ $\delta = 10$	223.51 85	$\lambda = 0.995$ $\delta = 1$	177.92 83	$\lambda = 0.994$ $\delta = 1$
UD-RLS	237.82 97	$\lambda = 0.9992$ $\delta = 1$	108.94 100	$\lambda = 0.9997$ $\delta = 0.45$	149.38 98	$\lambda = 0.9997$ $\delta = 1$

Table 2. Numerical results, part 2.

Structure / Algorithm	221		241		261	
BP	—	—	3513.81 31	$\mu = 0.02$	181.73 100	$\mu = 0.1$
MBP	—	—	906.29 95	$\mu = 0.15$ $\alpha = 0.25$	161.31 100	$\mu = 0.1$ $\alpha = 0.25$
ETB RLS	—	—	367.01 96	$\alpha = 0.97$ $\delta = 1000$	485.65 81	$\alpha = 0.98$ $\delta = 1000$
ETB UD-RLS	196.2 93	$\alpha = 0.95$ $\delta = 100$	157.19 100	$\alpha = 0.92$ $\delta = 100$	164.41 100	$\alpha = 0.95$ $\delta = 1000$

Table 3. Numerical results, part 3.

Structure / Algorithm	2221		2441		2661	
BP	3644.2 10	$\mu = 0.02$	272.62 100	$\mu = 0.152$	146.96 100	$\mu = 0.15$
MBP	3487.3 23	$\mu = 0.01$ $\alpha = 0.25$	245.05 100	$\mu = 0.09$ $\alpha = 0.55$	139.4 100	$\mu = 0.1$ $\alpha = 0.35$
ETB RLS	198.75 8	$\lambda = 0.92$ $\delta = 100$	212.68 84	$\lambda = 0.99$ $\delta = 100$	—	—
ETB UD-RLS	297.45 67	$\delta = 0.97$ $\delta = 100$	148.19 99	$\lambda = 0.995$ $\delta = 10$	81.75 100	$\lambda = 0.99$ $\delta = 10$

with the backpropagation (BP) and momentum backpropagation (MBP) algorithms.

In the first experiment, a multilayer neural network was trained to function as a 10–to–10 encoder (see, e.g., Karayiannis and Venetsanopoulos, 1993). This problem was considered in other papers as a benchmark. The 10–to–10 encoder is implemented by a neural network with 10 inputs and 10 output units trained to map 10 input patterns into the output. In this case 10–4–10, 10–5–10 and 10–6–10 structures with hyperbolic tangent transfer functions were investigated. The results are depicted in Table 1. It is easily seen that the performance improves with an increase in the number of neurons in the hidden layer and the best results are obtained by the UD-RLS algorithm followed by the RLS algorithm.

In the second experiment, the nonlinear function

$$f(x, y) = \left(1 + x^{-2} + y^{-1.5}\right)^2$$

was approximated. By sampling the input range $x, y \in [1, 5]$, 50 input-output patterns were obtained. The results for the 2–2–1, 2–4–1, 2–6–1 and 2–2–2–1, 2–4–4–1, 2–6–6–1 architectures are shown in Tables 2 and 3, respectively. From these tables it follows that all the algorithms perform better for structures with two hidden lay-

ers. Moreover, the performance improves with an increase in the number of neurons in hidden layers. Note that some simulations were unsuccessful and the best performance was obtained for the ETB UD-RLS algorithm.

6. Conclusions

The UD RLS algorithm requires a lower number of iterations than the traditional back-propagation one and than the RLS method applied to FMNN training. Moreover, the presented algorithms are computationally more efficient than the classical RLS method. It would be interesting to compare the computational load required to run learning algorithms from the RLS family. The appropriate results are depicted in Tables 4 and 5 for the RLS, and UD RLS algorithms, respectively.

Table 4. Computational load of the RLS algorithm.

	RLS
Linear	$2x^3 + 10x^2 + 16x + 8 + (2x + 2)y$
Nonlinear ETB	$2x^3 + 10x^2 + 16x + 8 + (2x + 2)y$
Nonlinear	$(2x^3 + 10x^3 + 18x + 10)y$

Table 5. Computational load of the UD RLS algorithm.

	UD RLS
Linear	$3x^2 + 13x + 9 + (2x + 2)y$
Nonlinear ETB	$3x^2 + 13x + 9 + (2x + 2)y$
Nonlinear	$(3x^2 + 15x + 11)y$

For simplicity, the results for one layer in a multilayer network are presented. In Tables 4 and 5, x and y denote the numbers of inputs and neurons, respectively. It can be easily seen that the computational load (the number of operations, i.e., multiplications, additions and function value calculations) is smallest for UD RLS algorithms and largest for RLS algorithms. Moreover, nonlinear ETB neural networks require the same number of operations as linear neural networks. The computational burden is significantly bigger for nonlinear neural networks. In future research it would be interesting to determine the structure and initial weights of the neural network by a combination of UD RLS learning procedures with genetic algorithms (Kitano, 1994; Yao, 1999).

References

Abid S., Fnaiech F. and Najim M. (2001): *A fast feedforward training algorithm using a modified form of the standard backpropagation algorithm*. — IEEE Trans. Neural Netw. Vol. 12, No. 2, pp. 424–434.

Ampazis N. and Perantonis J. (2002): *Two highly efficient second-order algorithms for training feedforward networks*. — IEEE Trans. Neural Netw. Vol. 13, No. 5, pp. 1064–1074.

Azimi-Sadjadi M.R. and Liou R.J. (1992): *Fast learning process of multi-layer neural network using recursive least squares method*. — IEEE Trans. Signal Process. Vol. 40, No. 2, pp. 443–446.

Bilski J. (1995): *Fast learning procedures for neural networks*. — Ph.D. Thesis, AGH University of Science and Technology, (in Polish).

Bilski J. and Rutkowski L. (1996): *The recursive least squares method versus the backpropagation learning algorithms*. — Second Conf. *Neural Networks and Their Applications*, Szczyrk, Poland, pp. 25–31.

Bilski J. and Rutkowski L. (1998): *A fast training algorithm for neural networks*. — IEEE Trans. Circuits Syst. II, Vol. 45, No. 6, pp. 749–753,

Bilski J. and Rutkowski L. (2003): *A family of the RLS neural network learning algorithms*. — Techn. Report, Dept. Comp. Eng., Technical University of Częstochowa, Poland.

Bishop C.M. (1995): *Neural Networks for Pattern Recognition*. — Oxford: Clarendon Press.

Bojarczak O.S.P. and Stodolski M. (1996): *Fast second-order learning algorithm for feedforward multilayer neural networks and its application*. — Neural Netw. Vol. 9, No. 9, pp. 1583–1596.

Chen X.- H.Y.G.- A. (1992): *Efficient backpropagation learning using optimal learning rate and momentum*. — Neural Netw., Vol. 10, No. 3, pp. 517–527.

Joost M. and Schiffmann W. (1998): *Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization*. — Int. J. Uncert. Fuzz. Knowledge-Based Syst., Vol. 6, No. 2, pp. 117–126.

Karayiannis N.B. and Venetsanopoulos A.N. (1993): *Efficient Learning Algorithms for Neural Networks (ELEANNE)*. — IEEE Trans. Syst. Man Cybern., Vol. 23, No. 5, pp. 1372–1383.

Kitano H. (1994): *Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms*. — Physica D., Vol. 75, No. 1–3, pp. 225–238.

Korbicz J., Obuchowicz A., Uciński D. (1994): *Artificial Neural Networks. Fundamentals and Applications*. — Warsaw: Akademicka Oficyna Wydawnicza PLJ, (in Polish).

Lera G. and Pinzolas M. (2002): *Neighbourhood based Levenberg-Marquardt algorithm for neural network training*. — IEEE Trans. Neural Netw. Vol. l3, No. 5, pp. 1200–1203.

Leung Ch.S., Tsoi Ah.Ch. and Chan L. W. (2001): *Two regularizers for recursive least squared algorithms in feedforward multilayered neural networks*. — IEEE Trans. Neural Netw. Vol. 12, No. 6, pp. 1314–1332.

Moller M. (1993): *A scaled conjugate gradient algorithm for fast supervised learning*. — Neural Netw. Vol. 6, No. 4, pp. 525–533.

Perantonis S. and Karras D. (1995): *An efficient constrained learning algorithm with momentum acceleration*. — Neural Netw. Vol. 8, No. 2, pp. 237–249.

Rutkowski L. (1994): *Adaptive Signal Processing: Theory and Applications*. — Warsaw: WNT, (in Polish).

Strobach P. (1990): *Linear Prediction Theory – A Mathematical Basis for Adaptive Systems*. — New York: Springer-Verlag.

Sum J., Chan L.W., Leung C.S. and Young G. (1998): *Extended Kalman filter-based pruning method for recurrent neural networks*. — Neural Comput. Vol. 10, No. 6, pp. 1481–1505.

Sum J., Leung C., Young G.H. and Kan W. (1999): *On the Kalman filtering method in neural-network training and pruning*. — IEEE Trans. Neural Netw., Vol. 10, No. 1, pp. 161–166.

Wellstead P.E. and Zarrop M.B. (1991): *Self-Tuning Systems Control and Signal Processing*. — Chichester Wiley.

Yao X. (1999): *Evolving artificial neural networks*. — Proc. IEEE, Vol. 87, No. 9, pp. 1423–1447.

Zhang Y. and Li R. (1999): *A fast U-D factorization-based learning algorithm with applications to nonlinear system modelling and identification*. — IEEE Trans. Neural Netw. Vol. 10, No. 4, pp. 930–938.

Received: 25 February 2004
Revised: 26 June 2004