The Adaptation of Perceptrons with Applications to Inverse Dynamics Identification of Unknown Dynamic Systems

Hebertt J. Sira-Ramírez, *Senior Member, IEEE,* and Stanislaw H. Żak, *Member, IEEE*

***Abstract*—A new class of adaptation algorithms is proposed for single and multilayer perceptrons with discontinuous nonlinearities. The learning parameters in the proposed algorithms are adjusted to force the error between the actual and desired outputs to satisfy a stable difference error equation. The behavior of the algorithms is illustrated on the application example of inverse system modeling.**

I. Introduction

The DEVELOPMENT of the perceptron can be traced back to the early days of pattern recognition (See [3]–[5], [7] and [10] for more details). Its application as an adaptive system to the control of many degrees of freedom robotic manipulators was proposed by Albus [1] in 1975. More recently, Widrow and Winter [7] discussed numerous applications of perceptrons for adaptive filtering (See also [2] and [10]), adaptive pattern recognition, and adaptive signal processing.

A critical role in advancing the practicality of perceptrons, and neural networks in general, are played by adaptation algorithms, others are hardware, systems approach, etc. In the case of the single perceptron one of the most well known algorithm that minimizes the mean square error between the desired output and the actual output is due to Widrow and Hoff. For the layered perceptron the central role is played by the back-propagation algorithm (see [7] for historical remarks, [5] and [8] for the derivation of this algorithm and [9] for an example of its application). One relatively minor drawback of the back-propagation algorithm is the requirement that the nonlinear activation functions be differentiable, while major drawbacks are learning speed and scalability.

In this paper, we propose a new class of adaptation, or training, algorithms for multilayer perceptrons. Our proposed algorithms, unlike the back-propagation training algorithm, do not require differentiability along the network's signal paths. On the contrary, we consider nondifferentiable activation functions such as hard limiters. In the back-propagation algorithm one presents an input to the network and calculates the output corresponding to the current set of learning parameters. One then compares the actual network output with the desired output and calculates the Euclidean distance between the actual and desired outputs, called the error function. The learning procedure aims at minimizing the error function by suitable adjustments of the learning parameters. In particular, one calculates the gradient of the error function with respect to the learning parameters starting at the output nodes and working back towards the input nodes through the hidden layers. Once the gradient is calculated, the learning parameters are adjusted using the gradient descent method. A substantial departure from the back-propagation procedure is proposed in our new training algorithms. Here, the learning parameters are adjusted to force the error between the actual and desired outputs to satisfy a stable difference error equation, rather than to minimize an error function. This approach allows one to better control the stability and speed of convergence by appropriate choice of parameters of the error difference equation. The behavior of the proposed algorithms is illustrated via the application example. In Section II we briefly review the celebrated Widrow–Hoff adaptation rule and propose a new adaptation algorithm for the single perceptron along with its geometrical interpretation. In Section III we formulate new training algorithms for multilayer perceptrons which can be viewed as an extension of the rule proposed for the single perceptron. In Section IV we test the proposed training algorithms on the application example of the inverse system modeling. Inverse system modeling is an important problem in control theory. The reason for this is that a number of robust control algorithms utilize the inverse model of the plant to be controlled. Conclusions are found in Section V. Simulations were performed using the SIMNON package and the programs are listed in the Appendixes.

II. The Adaptation of the Weights of a Single Perceptron

The single perceptron as an adaptive threshold element is shown in Fig. 1.

One can use the Widrow–Hoff delta rule (see [7] for its discussion) to adjust the weights $\omega_i (i = 1, 2, \cdots, n)$. The algorithm can be written as

$$W(k+1) = W(k) + \frac{\alpha e(k) X}{X^T X} \tag{1}$$

Manuscript received March 16, 1990; revised December 22, 1990. This work was supported by the Consejo de Desarrollo Cientifico, Humanistico y Tecnologico (CDCHT) of the Universidad de Los Andes Merida-Venezuela, under Research Grant I-325-90, by the Consejo Nacional de Investigaciones Cientificas y Tecnologicas (CONICIT) of Venezuela and by the School of Electrical Engineering of Purdue University, West Lafayette, IN.

H. J. Sira-Ramírez is with the Departamento Sistemas de Control, Escuela de Ingenieria de Sistemas, Universidad de Los Andes, Merida, Venezuela.

S. H. Żak is with the School of Electrical Engineering, Purdue University, W. Lafayette, IN 47907.

IEEE Log Number 9042703.

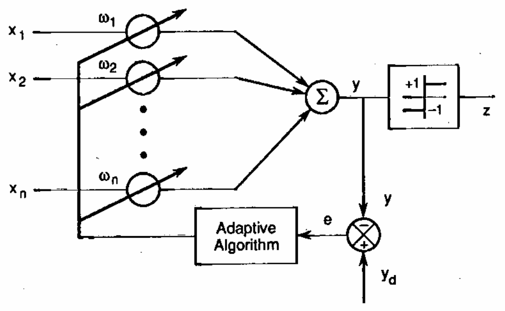


Fig. 1. Single-layer perceptron.

where k is the time index or the adaptation cycle number, $W(k) = [\omega_1(k), \cdots, \omega_n(k)]^T$ is the value, at time at time k, of the weight vector, $X = [x_1, \cdots, x_n]^T$ is the present input pattern, $e(k) = y_d - y(k)$ is the present error, and α is the reduction factor whose practical range is (0.1, 1.0).

After some manipulations, one can conclude that the error is reduced by a factor of α at each new learning iteration as the weights are changed while holding the input pattern X fixed [7]. More specifically, the error obeys the following difference equation

$$e(k+1) = (1-\alpha)e(k).$$

As one can see from this equation, the choice of α controls the speed of convergence towards zero of the learning error signal e.

After discussing the Widrow–Hoff delta rule, we can now propose a new adaptation algorithm for the single perceptron shown in Fig. 1.

In order to proceed, we need the following notation:

$$\text{SGN } X = \begin{bmatrix} \text{sgn } x_1 \\ \vdots \\ \text{sgn } x_n \end{bmatrix}$$

where

$$\text{sgn } x_i = \begin{cases} +1, & \text{if } x_i > 0 \\ -1, & \text{if } x_i < 0, \ i = 1, 2, \cdots, n \end{cases}$$

We shall present the new algorithm in the following theorem.

Theorem 1: If the weights ω_i of the single perceptron, shown in Fig. 1, are adapted according to the rule:

$$W(k+1) = W(k) + \frac{\alpha e(k) \text{SGN } X}{X^T \text{SGN } X} \tag{2}$$

with $0 < \alpha < 2$ (practical range of α is (0.1, 1.0)), then the error $e(k)$ tends asymptotically to zero with the rate of convergence $(1-\alpha)$.

Proof: Note that

$$\begin{aligned} e(k+1) - e(k) &= y_d - y(k+1) - [y_d - y(k)] \\ &= y(k) - y(k+1) \\ &= \sum_{i=1}^{n} \omega_i(k)x_i - \sum_{i=1}^{n} \omega_i(k+1)x_i \\ &= -\sum_{i=1}^{n} [\omega_i(k+1) - \omega_i(k)]x_i \\ &= -[W(k+1) - W(k)]^T X \\ &= -X^T[W(k+1) - W(k)]. \end{aligned}$$

We can now use the proposed update rule to obtain

$$\begin{aligned} e(k+1) - e(k) &= -X^T \frac{\alpha e(k) \text{ SGN } X}{X^T \text{ SGN } X} \\ &= -\alpha e(k), \qquad \text{if } X \neq 0. \end{aligned}$$

Hence

$$e(k+1) = (1-\alpha)e(k).$$

Thus, if $0 < \alpha < 2$ then

$$\lim_{k \to \infty} e(k) = 0.$$

Note that in the new adaptation algorithm, as well as in the Widrow–Hoff algorithm, the error is reduced by a factor of α. There are other similarities between these two algorithms. As we shall show below, they can both be derived using the theory of quasi-sliding modes in discrete-time dynamical systems. For this task, we will need the following notation and definitions.

Consider a dynamical system modeled by the following controlled difference equations of the state vector $W(k)$, with a single output signal $e(k)$ defined by the mapping h:

$$S \begin{cases} W(k+1) = f(W(k), U(k)) \\ \quad e(k) = h(W(k)) \end{cases}$$

where $W(k) \in \mathbf{R}^n, U(k) \in \mathbf{R}^m, e(k) \in \mathbf{R}$.

Consider now the following level curve of the output map:

$$h^{-1}(0) = \{W \in \mathbf{R}^n | e = h(W) = 0\}.$$

Definition 1: A quasi-sliding mode is said to exist on $h^{-1}(0)$ if there exists a control law $U(k)$ such that the motion of S satisfies the relation:

$$|e(k+1)e(k)| < e^2(k)$$

for $e(k) \neq 0$.

One can show that the previous condition is equivalent to the condition of Sarpturk *et al.* [11] of the form:

$$|e(k+1)| < |e(k)|$$

for $e(k) \neq 0$. (See also [12] for the discussion of the previous conditions.)

We can now look at the proposed adaptation rule (2) and the Widrow–Hoff delta rule (1) as discrete-time controlled dynamical systems of the form S, where the weight vector $W(k)$ is viewed as a state vector at time k, and the correction terms of the present value of the weight vector are viewed as controllers $U(k)$ (where $m = n$), and the error signal $e(k)$ is viewed as the single output signal.

Duly armed with the previous definitions we can now prove the following statement.

Theorem 2: The update correction terms for the weights in the controlled dynamical systems represented by the adaptation rules (1) and (2) guarantee the existence of a quasi-sliding mode on the zero learning error set $h^{-1}(0)$.

Proof: Note that in both algorithms (1) and (2), the error satisfies the difference equation

$$e(k+1) = (1-\alpha)e(k)$$

from where it follows that $e(k) \neq 0$ and $0 < \alpha < 2$ (practical range (0.1, 1.0))

$$|e(k+1)e(k)| = |(1-\alpha)e^2(k)| = |(1-\alpha)|e^2(k) < e^2(k).$$

The proof is a sufficient condition for a quasi-sliding mode to occur.

In what follows, we shall give a geometrical interpretation of the convergence of the proposed adaptation algorithm (2). Note that

$$W(k+1) = W(k) + U(k)$$

where

$$U(k) = \frac{\alpha e(k) \text{ SGN } X}{X^T \text{ SGN } X}.$$

Thus, the correction vector $U(k)$ always updates the weight vector $W(k)$ in the direction of the line passing through the vertices of the hypercube $H = \{W \in \mathbf{R}^n | -1 \leq w_i \leq +1;\ i = 1, 2, \cdots, n\}$ corresponding to the vectors SGN X and SGN$(-X)$ with the coefficient of proportionality given by

$$\frac{\alpha e(k)}{X^T \text{ SGN } X}$$

even if X does not lie on a vertex of the hypercube H. The correction term $U(k)$ is always pointing to the same orthant where X lies, provided $e(k) > 0$ and it points to the opposite orthant where X lies whenever $e(k) < 0$. This is illustrated in Fig. 2 for $n = 2$.

We can now present new adaptation algorithms for the three-layer perceptron. The proposed algorithm is an extension of the training algorithm for the two-layer perceptrons presented in [6].

III. Adaptation Algorithms for Multilayer Perceptrons

In this section we will be concerned with multilayer perceptrons that are feedforward networks with one or more layers of nodes between the input and output nodes [4]. It is generally acknowledged that the applicability of multilayer neural networks is highly dependent on the efficiency of the training algorithms. One of the best known training algorithms for multilayer perceptrons is the back-propagation algorithm [8]. A disadvantage of the back-propagation algorithm is the inherent requirement of continuous differentiability of the nonlinearities.

In this section we propose a new class of training algorithms for the multilayer perceptrons with discontinuous nonlinearities. In our analysis we shall utilize the structure depicted in Fig. 3.

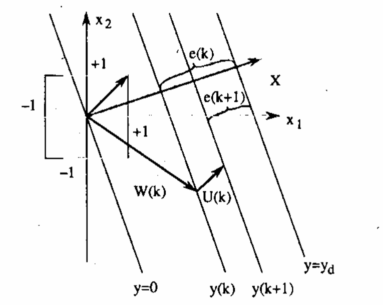


Fig. 2. Geometrical interpretation of the dynamical behavior of algorithm (2) when $e(k) > 0$. (Notice that when $e(k) < 0, U(k)$ will be directed in the opposite direction.)

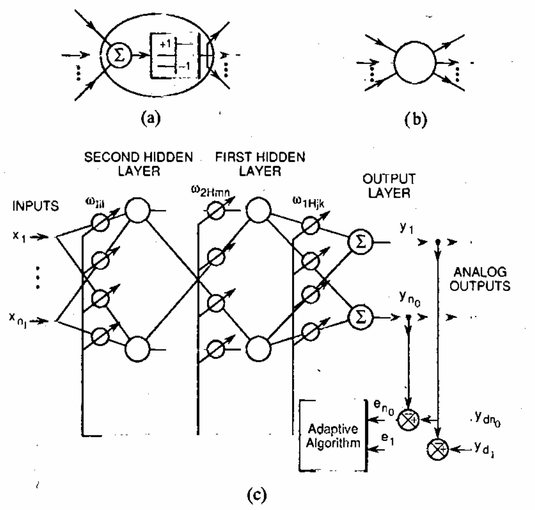


Fig. 3. (a) Model of a single neuron. (b) Symbol for a single neuron. (c) A three-layer adaptive perceptron.

A. Notation and Some Definitions

The κth output of the network in Fig. 3 is obtained as the weighted sum of the outputs of the first hidden layer

$$y_{o\kappa} = \sum_{j=1}^{n_{1H}} \omega_{1Hj\kappa}(k) z_{1Hj}(k), \qquad \kappa = 1, 2, \cdots, n_o$$

where n_{1H} is the number of neurons in the first hidden layer, n_o is the number of the network outputs, and z_{1Hj} are the binary outputs of the neurons in the first hidden layer. The time varying quantities $\omega_{1Hj\kappa}$ represent the interconnection weights between the first hidden layer and the outputs of the net. In vector notation

$$y_{o\kappa} = [W_{1H}^{\kappa}(k)]^T Z_{1H}(k), \qquad \kappa = 1, 2, \cdots, n_o$$

where

$$W_{1H}^{\kappa}(k) = \begin{bmatrix} \omega_{1H1\kappa}(k) \\ \omega_{1H2\kappa}(k) \\ \vdots \\ \omega_{1Hn_{1H}\kappa}(k) \end{bmatrix}$$

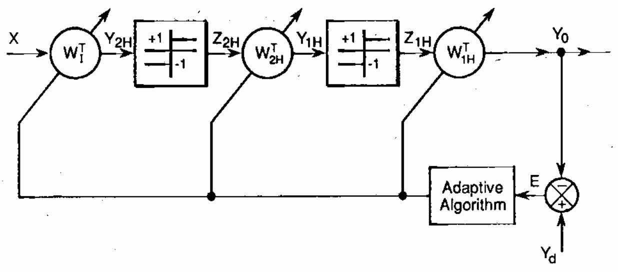


Fig. 4. Schematic representation of the three-layer adaptive perceptron from Fig. 3.

and

$$Z_{1H}(k) = \begin{bmatrix} z_{1H_1}(k) \\ z_{1H_2}(k) \\ \vdots \\ z_{1Hn_{1H}}(k) \end{bmatrix} = \begin{bmatrix} \operatorname{sgn} y_{1H1}(k) \\ \operatorname{sgn} y_{1H2}(k) \\ \vdots \\ \operatorname{sgn} y_{1Hn_{1H}} \end{bmatrix}$$
$$=: \text{SGN } Y_{1H}(k)$$

where

$$\operatorname{sgn} y_{1H\kappa} = \begin{cases} 1, & \text{if} \quad y_{1H\kappa} > 0 \\ -1, & \text{if} \quad y_{1H\kappa} < 0, \quad \kappa = 1, 2, \cdots, n_0. \end{cases}$$

If we denote by W_{1H} the matrix of column vectors W_{1H}^{κ}, then the vector $Y_o(k)$ of the output components $y_{o\kappa}(k)$ is obtained as (see Fig. 4)

$$Y_o(k) = [W_{1H}(k)]^T Z_{1H}(k). \tag{3}$$

Note that $W_{1H} \in \mathbf{R}^{n_{1H} \times n_o}$.

The analog outputs of the neurons in the first hidden layer are given by

$$y_{1Hn}(k) = \sum_{m=1}^{n_{2H}} \omega_{2Hmn}(k) z_{2Hm}(k)$$
$$= [W_{2H}^n(k)]^T Z_{2H}(k), \quad n = 1, 2, \cdots, n_{1H}$$

where

$$W_{2H}^n(k) = \begin{bmatrix} \omega_{2H1n}(k) \\ \omega_{2H2n}(k) \\ \vdots \\ \omega_{2Hn_{2H}n}(k) \end{bmatrix}$$

and

$$Z_{2H}(k) = \begin{bmatrix} z_{2H1}(k) \\ z_{2H2}(k) \\ \vdots \\ z_{2Hn_{2H}}(k) \end{bmatrix}.$$

Denote by W_{2H} the matrix of column vectors W_{2H}^n. Then the vector of the analog neuron outputs Y_{1H} of the first hidden layer is (see Fig. 4)

$$Y_{1H}(k) = [W_{2H}(k)]^T Z_{2H}(k). \tag{4}$$

Note that $W_{2H}(k) \in \mathbf{R}^{n_{2H} \times n_{1H}}$. Observe that

$$Z_{2H}(k) = \text{SGN } Y_{2H}(k).$$

The components of the vector $Y_{2H}(k)$ are the analog outputs of the neurons of the second hidden layer given by

$$y_{2Hl}(k) = \sum_{i=1}^{n_I} \omega_{Iil}(k) x_i$$
$$= [W_I^l(k)]^T X, \qquad i = 1, \cdots, n_{2H}$$

where

$$W_I^l(k) = \begin{bmatrix} \omega_{I1l}(k) \\ \omega_{I2l}(k) \\ \vdots \\ \omega_{In_Il}(k) \end{bmatrix}$$

and

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_I} \end{bmatrix}.$$

If we denote by W_I the matrix of column vectors $W_I^l(k)$ then the vector $Y_{2H}(k)$ of the analog neuron outputs of the second hidden layer is (see Fig. 4)

$$Y_{2H}(k) = [W_I(k)]^T X \tag{5}$$

where

$$W_I(k) \in \mathbf{R}^{n_I \times n_{2H}}.$$

Let Y_d denotes the vector of desirable output values with components, $Y_{d\kappa}; \kappa = 1, 2, \cdots, n_o$. The error vector $E(k)$ at time k is then obtained as

$$E(K) = \begin{bmatrix} e_1(k) \\ \vdots \\ e_{n_o}(k) \end{bmatrix} = Y_d - Y_o(k). \tag{6}$$

Let the weights update law be represented in general by the following equations:

$$\omega_{1Hj\kappa}(k+1) = \omega_{1Hj\kappa}(k) + u_{1Hj\kappa}(k)$$
$$\omega_{2Hmn}(k+1) = \omega_{2Hmn}(k) + u_{2Hmn}(k)$$
$$\omega_{Iil}(k+1) = \omega_{Iil}(k) + u_{Iil}(k)$$

or, in matrix notation:

$$W_{1H}(k+1) = W_{1H}(k) + U_{1H}(k)$$
$$W_{2H}(k+1) = W_{2H}(k) + U_{2H}(k)$$
$$W_{1}(k+1) = W_{I}(k) + U_{I}(k)$$

where $U_{1H}(k) \in \mathbb{R}^{n_{1H}\times n_o}, U_{2H}(k) \in \mathbb{R}^{n_{2H}\times n_{1H}}$, and $U_I(k) \in \mathbb{R}^{n_I\times n_{2H}}$ are the correction matrices updating the weight matrices at time k. In Fig. 4 we represent schematically the three-layer adaptive perceptron using the above notation. The proposed matrix representation of the considered neural network greatly facilitates its further analysis.

B. Description of the Algorithms

The following lemma will be used in the subsequent considerations.

Lemma 1: The error vector $E(k)$ satisfies the following difference equation as a function of the input and hidden layers matrix update control weights $U_{1H}(k), U_{2H}(k)$, and $U_I(k)$:

$$\begin{aligned} E(k+1) - E(k) &= [W_{1H}(k)]^T\Big\{\text{SGN } Y_{1H}(k) - \text{SGN}\Big\{[W_{2H}(k) \\ &\quad + U_{2H}(k)]^T\ \text{SGN}\Big(Y_{2H}(k) + [U_I(k)]^T X\Big)\Big\}\Big\} \\ &\quad - [U_{1H}(k)]^T\ \text{SGN}\Big\{[W_{2H}(k) + U_{2H}(k)]^T \\ &\quad \text{SGN}\big(Y_{2H}(k) + [U_I(k)]^T X\big)\Big\} \end{aligned} \tag{7}$$

Proof: One can check that the following string of equalities is satisfied

$$\begin{aligned} E(k+1) - E(k) &= Y_o(k) - Y_o(k+1) \\ &= [W_{1H}(k)]^T Z_{1H}(k) - [W_{1H}(k+1)]^T Z_{1H}(k+1) \\ &= [W_{1H}(k)]^T Z_{1H}(k) \\ &\quad - \big[W_{1H}(k) + U_{1H}(k)\big]^T Z_{1H}(k+1) \\ &= [W_{1H}(k)]^T [Z_{1H}(k) - Z_{1H}(k+1)] \\ &\quad - [U_{1H}(k)]^T Z_{1H}(k+1) \\ &= [W_{1H}(k)]^T [\text{SGN } Y_{1H}(k) - \text{SGN } Y_{1H}(k+1)] \\ &\quad - [U_{1H}(k)]^T \text{SGN } Y_{1H}(k+1). \end{aligned} \tag{8}$$

Note that

$$\begin{aligned} Y_{1H}(k+1) &= [W_{2H}(k+1)]^T Z_{2H}(k+1) \\ &= [W_{2H}(k) + U_{2H}(k)]^T \text{SGN } Y_{2H}(k+1) \\ &= [W_{2H}(k) + U_{2H}(k)]^T \text{SGN}\Big([W_I(k+1)]^T X\Big) \\ &= [W_{2H}(k) + U_{2H}(k)]^T \\ &\quad \cdot \text{SGN}\Big([W_I(k) + U_I(k)]^T X\Big). \end{aligned}$$

Substituting this equation into (8) yields

$$\begin{aligned} &E(k+1) - E(k) \\ &\quad = [W_{1H}(k)]^T\Big\{\text{SGN } Y_{1H}(k) - \text{SGN}\Big\{(W_{2H}(k) \\ &\quad + U_{2H}(k))^T\ \text{SGN}([W_I(k) + U_I(k)]^T X)\Big\}\Big\} \\ &\quad - [U_{1H}(k)^T \text{SGN}\Big\{[W_{2H}(k) + U_{2H}(k)]^T \\ &\quad \cdot \text{SGN}([W_I(k) + U_I(k)]^T X)\Big\}. \end{aligned}$$

Upon substitution of (5), the last equation is seen to coincide with (7).

We shall now present the new training algorithm for the three-layer perceptron (see Fig. 3) in the following theorem.

Theorem 3: If the weight correction matrices $U_I(k), U_{2H}(k)$, and $U_{1H}(k)$ are, respectively, chosen as

$$U_I(k) = -\frac{2[\text{SGN } X][Y_{2H}(k)]^T}{X^T \text{SGN } X} \tag{9}$$

$$U_{2H}(k) = -\frac{2[\text{SGN } Z_{2H}(k)][Y_{1H}(k)]^T}{Z_{2H}^T(k)\ \text{SGN } Z_{2H}(k)} \tag{10}$$

and

$$U_{1H}(k) = \frac{[\text{SGN } Z_{1H}(k)][AE(k)]^T}{Z_{1H}^T(k)\ \text{SGN } Z_{1H}(k)} \tag{11}$$

then the learning error vector $E(k)$ satisfies the following asymptotically stable difference equation

$$E(k+1) = (I - A)E(k) \tag{12}$$

where A is an $n_o \times n_o$ diagonal matrix chosen as

$$A = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{n_o} \end{bmatrix}$$

such that $|1 - \alpha_\kappa| < 1$, $\kappa = 1, 2, \cdots, n_o$.(Practical range for α_k is (0.1,1.0)).

Proof: Note that the transpose of the weight correction matrix $U_I(k)$ is given by

$$[U_I(k)]^T = -\frac{2Y_{2H}(k)[\text{SGN } X]^T}{[\text{SGN } X]^T X}. \tag{13}$$

Substituting this last expression into the error difference equation (7) gives

$$\begin{aligned} &E(k+1) - E(k) \\ &\quad = [W_{1H}(k)]^T\Big\{\text{SGN } Y_{1H}(k) + \text{SGN}\Big\{[W_{2H}(k) \\ &\quad + U_{2H}(k)]^T\ \text{SGN } Y_{2H}(k)\Big\}\Big\} \\ &\quad + [U_{1H}(k)]^T \text{SGN}\Big\{[W_{2H}(k) \end{aligned}$$

$$
\begin{aligned}
&+U_{2H}(k]^T)\mathrm{SGN}\ Y_{2H}(k)\Big\} \\
&=[W_{1H}(k)]^T\Big\{Z_{1H}(k)+\mathrm{SGN}(Y_{1H}(k) \\
&+\ [U_{2H}(k)]^T Z_{2H}(k))\Big\} \\
&+\ [U_{1H}(k)]^T\mathrm{SGN}\Big\{Y_{1H}(k) \\
&+[U_{2H}(k)]^T Z_{2H}(k)\Big\} \qquad (14)
\end{aligned}
$$

Upon substituting the transpose of the weight correction matrix $U_{2H}(k)$ into (14) yields

$$
\begin{aligned}
E(k+1)-E(k) &= -[U_{1H}(k)]^T\mathrm{SGN}\ Y_{1H}(k) \\
&= -[U_{1H}(k)]^T Z_{1H}(k). \qquad (15)
\end{aligned}
$$

Finally, substituting the transpose of the weight correction matrix $U_{1H}(k)$ into (15) we obtain (12).

Remark 1: Notice that A may also be chosen as an arbitrary nondiagonal matrix such that the matrix $[I-A]$ has its eigenvalues in the open unit circle of the complex plane.

Remark 2: Observe that

$$\mathrm{SGN}\ Z_{iH}(k)=Z_{iH}(k), \qquad i=1,2$$

and that

$$
\begin{aligned}
[Z_{iH}(k)]^T\mathrm{SGN}\ Z_{iH}(k) &= Z_{iH}^T(k)Z_{iH}(k) \\
&= n_{iH}, \qquad i=1,2.
\end{aligned}
$$

Utilizing the above relations we can represent the weight correction matrices $U_{2H}(k)$ and $U_{1H}(k)$ in the following form:

$$U_{2H}(k)=-\frac{2Z_{2H}(k)Y_{1H}^T(k)}{n_{2H}} \qquad (10')$$

$$U_{1H}(k)=\frac{Z_{1H}(k)[AE(k)]^T}{n_{1H}}. \qquad (11')$$

Observe also that instead of the weight correction matrix $U_I(k)$ given by (9) one may utilize the following alternative form:

$$U_I(k)=-\frac{2XY_{2H}^T(k)}{X^TX}. \qquad (9')$$

IV. Application Example

In this section we test the proposed training algorithms given in Theorems 1 and 3 on an application example. In this example we will be dealing with the problem of the inverse dynamics identification of an unknown dynamical system according to the scheme shown in Fig. 5. For more details about the inverse modeling problem, the reader is referred to [7, p. 30]. In the simulations, the "unknown" dynamical system is described by the following equations:

$$
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -2x_1-2x_2+u \\
y &= x_1.
\end{aligned}
$$

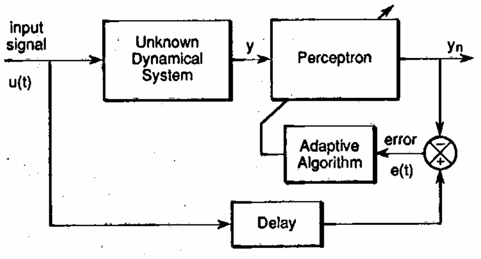


Fig. 5. Block diagram of the inverse dynamics identification scheme.

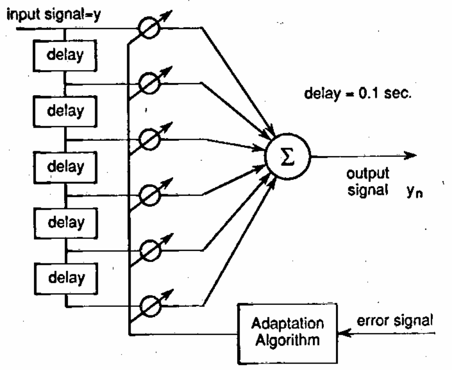


Fig. 6. Structure of the single-layer perceptron used to identify the inverse dynamics of the unknown system.

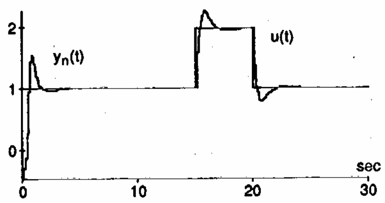


Fig. 7. Time history of $(u)t$ and $y_n(t)$ for the single perceptron. The program where the three-layer perceptron is used as an inverse dynamics identifier in Appendix II.

The input signal to the perceptron is provided by means of the delay elements arranged in a transversal filter scheme as depicted in Fig. 6.

The input signal to the system $u(t)$, and the output $y_n(t)$ of the perceptron are shown in Fig. 7. The program for the inverse dynamics identification using single perceptron is attached in Appendix I.

The structure of the three-layer perceptron used to identify the inverse dynamics is depicted in Fig. 8. For more details see Appendix II.

In Fig. 9 we show a discontinuous input signal $u(t)$, and the perceptron output signal $y_n(t)$ corresponding to the output of the dynamical system inverse which tries to reconstruct the original input signal.

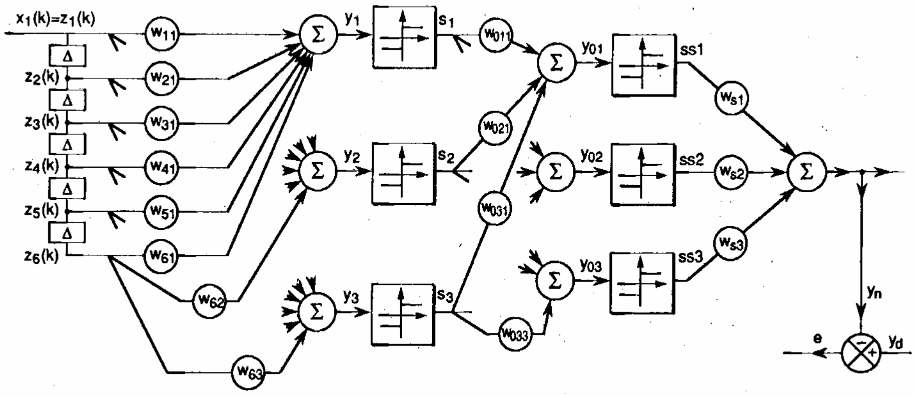


Fig. 8. A three-layer perceptron used in the simulation experiment (see Appendix II for more details).

The program where the three-layer perceptron is used as an inverse dynamics identifier is listed in Appendix II.

V. Conclusion

In this paper we have proposed new adaptation algorithms for single and multilayer perceptrons with discontinuous nonlinearities. The behavior of the proposed algorithms was tested on the application example and simulation results were included. The simulations shown in this paper were performed using the SIMNON package developed by the Swedish researchers for the purpose of simulation of nonlinear systems.

The obtained results in this paper can be used to control unknown dynamic systems using neural controllers. Indeed, many robust control algorithms utilize the inverse dynamics of

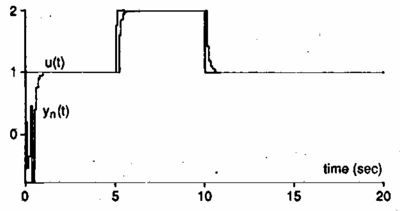


Fig. 9. Time history of $u(t)$ and $y_n(t)$ for the three-layer perceptron.

the plant to be controlled. Thus, the proposed structures where the perceptrons are the inverse system model identifiers should constitute a part of the controller. Research in this direction is now underway.

Appendix I

SIMNON program for the inverse dynamics identification using single perceptron.

Discrete System Perceptron-1

```
input u y
state w1 w2 w3 w4 w5 w6 z1 z2 z3 z4 z5 z6
new nw1 nw2 nw3 nw4 nw5 nw6 nz1 nz2 nz3 nz4 nz5 nz6
time t
tsamp ts
nz1 = z2
nz2 = z3
nz3 = z4
nz5 = z6
nz6 = y
nw1 = w1 + u1
nw2 = w2 + u2
nw3 = w3 + u3
nw4 = w4 + u4
nw5 = w5 + u5
nw6 = w6 + u6
```

```
u1 = a*e*sign(z6)/d
u2 = a*e*sign(z5)/d
u3 = a*e*sign(z4)/d
u4 = a*e*sign(z3)/d
u5 = a*e*sign(z2)/d
u6 = a*e*sign(z1)/d
d = z1*sign(z1) + z2*sign(z2) + z3*sign(z3) + z4*sign(z4) + z5*sign(z5) + z6*sign(z6)
e = yd - y
y = w1*z6 + w2*z5 + w3*z4 + w4*z3 + w5*z2 + w6*z1
yd = u
ts = t + h
h : 0.1
a : 0.6
end
```

Continuous system Plant

```
output u y
state x1 x2
der dx1 dx2
time t
dx1 = x2
dx2 = -2*x1 - 2*x2 + u
y = x1
u = if t < 5 then 1 else if t < 10 then 2 else 1
end
```

Connecting system Invcon 1

```
time t
u[Perceptron-1] = u[Plant]
y[Perceptron-1] = y[Plant]
end
```

Appendix II

SIMNON program for the inverse dynamics identification using three-layer perceptron.

Continuous system Perceptron-3

```
input u y
state w11 w12 w13 w21 w22 w23 w31 w32 w33 w41 w42 w43 w51 w52 w53
state w61 w62 w63 wo11 wo12 wo13 wo21 wo22 wo23 wo31 wo32 wo33
state z2 z3 z4 z5 z6 ws1 ws2 ws3
new nw11 nw12 nw13 nw21 nw22 nw23 nw31 nw32 nw33 nw41 nw42 nw43 nw51 nw52 nw53
new nw61 nw62 nw63 nwo11 nwo12 nwo13 nwo21 nwo22 nwo23 nwo31 nwo32 nwo33
new nz2 nz3 nz4 nz5 nz6 nws1 nws2 nws3
time t
tsamp ts
z1 = v
nz2 = z1
nz3 = z2
nz4 = z3
nz5 = z4
nz6 = z5
nw11 = w11 + u11
nw12 = w12 + u12
nw13 = w13 + u13
nw21 = w21 + u21
nw22 = w22 + u22
nw23 = w23 + u23
nw31 = w31 + u31
nw32 = w32 + u32
nw33 = w33 + u33
nw41 = w41 + u41
nw42 = w42 + u42
nw43 = w43 + u43
nw51 = w51 + u51
nw52 = w52 + u52
nw53 = w53 + u53
nw61 = w61 + u61
```

(continued)

```
u42 = -2*(y2*sign(z4))/d
u43 = -2*(y3*sign(z4))/d
u51 = -2*(y1*sign(z5))/d
u52 = -2*(y2*sign(z5))/d
u53 = -2*(y3*sign(z5))/d
u61 = -2*(y1*sign(z6))/d
u62 = -2*(y2*sign(z6))/d
u63 = -2*(y3*sign(z6))/d
q1 = (ss1*a*e)/3
q2 = (ss2*a*e)/3
q3 = (ss3*a*e)/3
r11 = -2*yo1*s1/3
r12 = -2*yo2*s1/3
r13 = -2*yo3*s1/3
r21 = -2*yo1*s2/3
r22 = -2*yo2*s2/3
r23 = -2*yo3*s2/3
r31 = -2*yo1*s3/3
r32 = -2*yo2*s3/3
r33 = -2*yo3*s3/3
y1 = w11*z1 + w21*z2 + w31*z3
       +w41*z4 + w51*z5 + w61*z6
```

```
nw62 = w62 + u62
nw63 = w63 + u63
nwo11 = wo11 + r11
nwo12 = wo12 + r12
nwo13 = wo13 + r13
nwo21 = wo21 + r21
nwo22 = wo22 + r22
nwo23 = wo23 + r23
nwo31 = wo31 + r31
nwo32 = wo32 + r32
nwo33 = wo33 + r33
nswl = ws1 + q1
nsw2 = ws2 + q2
nws3 = ws3 + q3
u11 = -2*(y1*sign(z1))/d
u12 = -2*(y2*sign(z1))/d
u13 = -2*(y3*sign(z1))/d
u21 = -2*(y1*sign(z2))/d
u22 = -2*(y2*sign(z2))/d
u23 = -2*(y3*sign(z2))/d
u31 = -2*(y1*sign(z3))/d
u32 = -2*(y2*sign(z3))/d
u33 = -2*(y3*sign(z3))/d
u41 = -2*(y1*sign(z4))/d
y2 = w12*z1 + w22*z2 + w32*z3
     +w42*z4 + w52*z5 + w62*z6
y3 = w13*z1 + w23*z2 + w33*z3
     +w43*z4 + w53*z5 + w63*z6
s1 = sign(y1)
s2 = sign(y2)
s3 = sign(y3)
ss1 = sign(yo1)
ss2 = sign(yo2)
ss3 = sign(yo3)
yo1 = wo11*s1 + wo21*s2 + wo31*s3
yo2 = wo12*s1 + wo22*s2 + wo32*s3
yo3 = wo13*s1 + wo23*s2 + wo33*s3
d1 = z1*sign(z1) + z2*sign(z2) + z3*sign(z3)
d2 = z4*sign(z4) + z5*sign(z5) + z6*sign(z6)
d = d1 + d2
yn = ws1*ss1 + ws2*ss2 + ws3*ss3
yd = u
e = yd - yn
ts = t + h
h : 0.1
a : 0.6
end
```

Continuous system Plant

```
output u y
state x1 x2
der dx1 dx2
time t
dx1 = x2
dx2 = -2*x1 - 2*x2 + u
y = x1
u = if t < 5 then 1 else if t < 10 then 2 else 1
end
```

Connecting system Invcon3

```
time t
u[Perceptron-3] = u[Plant]
y[Perceptron-3] = y[Plant]
end
```

References

[1] J.S. Albus, "A New approach to manipulator control: The cerebellar model articulation controller (CMAC)" *Trans. ASME, J. Dyn. Syst. Measurement Contr.*, vol. 97, series G, no. 3, pp. 220–227, Sept. 1975.

[2] K.J. Åström and B. Wittenmark, *Adaptive Control.* Reading, MA: Addison-Wesley, 1989.

[3] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks,* vol. 11, no. 1, pp. 17–61, 1988.

[4] R.P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, pp. 4–22, Apr. 1987.

[5] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks.* Reading, MA: Addison-Wesley, 1989.

[6] S.H. Żak and H.J. Sira-Ramirez, "On the adaptation algorithms for generalized perceptrons," in *Proc. 8th Int. Congress of Cybernetics and Systems,* New York, June 11–15, 1990.

[7] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *Computer,* vol. 21, no. 3, pp. 25–39, Mar. 1988.

[8] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, 1986.

[9] S.S. Rangwala and D.A. Dornfeld, "Learning and optimization of machining operations using computing abilities of neural networks," *IEEE Trans. Syst. Man Cybern.*, vol. 19, no. 2, pp. 299–314, Mar./Apr. 1989.

[10] S. Haykin, *Introduction to Adaptive Filters.* New York: Macmillan, 1984.

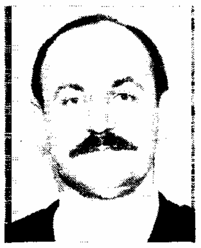
[11] S.Z. Sarpturk, Y. Istefanopulos, and O. Kaynak, "On the stability of discrete-time sliding mode control systems," *IEEE Trans. Automat. Contr.*, vol. AC-32, pp. 930–932, Oct. 1987.

[12] S.K. Spurgeon, "On conditions for the development of stable discrete-time sliding mode control systems," Math. Rep. No. A124, Dept. Math. Sci., Univ. Technol., Loughborough, Leics., U.K., Apr. 1990.

Hebertt Sira-Ramírez (M'75–SM'85) received the Degree of Ingeniero Electricista from the Universidad de Los Andes, Mérida, Venezuela in 1970. He received the M.S.E.E. and the E.E. degrees in 1974 and the Ph.D. degree in electrical engineering in 1977 from the Massachusetts Institute of Technology, Cambridge, MA.

He is currently a full professor in the Control Systems Department of the Escuela de Ingeniería de Sistemas of the Universidad de Los Andes, where he has also held the positions of Head of the Control Systems Department from 1978 to 1980, and was elected Vicepresident of the University for the term 1980–1984. He held visiting positions in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign and in the Departments of Aeronautical and Astronautical Engineering and the Electrical Engineering Department of the same university in 1986. He has also held brief visiting positions at the School of Electrical Engineering of Purdue University. He received the Venezuelan College of Engineers Award (CIV) for Scientific Research in 1987, and the Senior Researcher Scholarship Award from the Venezuelan National Council for Science and Technology (CONICIT) in 1990.

Dr. Sira-Ramírez is interested in the theory and applications of discontinuous control strategies for nonlinear dynamical systems.



Stanislaw H. Żak (M'81) received the Ph.D. degree in automatic control from the Technical University of Warsaw, Warsaw, Poland, in 1977.

He was an Assistant Professor in the Institute of Control and Industrial Electronics, Technical University of Warsaw, from 1977 to 1980. From 1980 until 1983, he was a visiting Assistant Professor in the Department of Electrical Engineering, University of Minnesota. In August, 1983, he joined the School of Electrical Engineering at Purdue University, West Lafayette, IN where he is now an Associate Professor.

Dr. Żak's research interests are in three different fields: an algebraic approach to the analysis and synthesis of linear systems, the control and observation of nonlinear systems, and applications of neural networks to optimization and control problems.