Neural Network-Based Control by Inverting Neural Models

Vasile PALADE, Gheorghe PUSCASU, Daniel-Ciprian NEAGU

Department of Applied Informatics, "Dunarea de Jos" University of Galati, Str. Domneasca Nr. 47, 6200 Galati, Romania, Fax: (40) 36 461353, E-mail: pvasile@cs.ugal.ro; gpuscasu@ac.ugal.ro; dneagu@cs.ugal.ro

Abstract: The success of the use of neural networks in control problems is based on the capabilities of the neural networks to cope with three main difficulties in control: complexity, nonlinearity and uncertainty. First, this paper presents the most common schemes of control which incorporate neural network techniques, and makes some considerations on these neural control structures. Then, it is proposed a method of inverting a neural network which represents the forward model of a process. The inverting method can be used in inverse neural control structure and neural model-based control structures, where an inverse model of the process is required. The method helps the user to determine the controller output using only the forward neural model, instead of training another neural network which implements the inverse dynamics of the process. Two case studies were developed in order to prove the inversion method.

Keywords: neural networks, neural control, inverse model.

1. INTRODUCTION

Neural networks with their inherent parallelism and their ability to learn, has been seen by many authors in the field of system controlling, as an exciting possibility to design adaptive controllers, when the dynamics of the system is deeply nonlinear, complex or unknown. The main advantages of the neural networks, which make them an important tool in order to enhance the capabilities of conventional controllers and to create robust controllers, able to better adapt the controller parameters to different plants and to environmental changes, are:

- neural networks can approximate any linear or nonlinear mapping between the input and the output of the system.
- they are able to learn in order to performe this approximation.
- robustness to partially network destruction, noise tolerance, and generalization ability to situations not contained in the training data set.
- computationally fastness once trained.

The problem of capturing the nonlinearity of the process to be modelled and controlled is to match the nonlinearity of the process with that of the network, by learning, neural networks being nonlinear systems themselves. It was shown in the literature [2][3][4][5][6][9] that neural networks can solve complex and difficult control tasks, where traditional control methods fails, neural networks being also able to work in the presence of noise.

The paper is structured as follows. Section 2 presents the most common neural network-based control structures from the literature, and discusses some of their aspects and characteristics. Section 3 proposes a method of inverting a neural network which represents the forward model of a process. The inversion method developed in the paper is the most computationally fast inversion method, optimising the searching of the network inputs which produce the desired network output. The calculus required by the inversion operation can be performed on-line with a regular computer. In order to test the inversion method, in section 4, two case studies were developed. The first case study is represented by the problem of controlling the level in a liquid tank, and the second case study is a DC motor control problem. The paper is ending with some conclusions.

2. NEURAL NETWORK BASED CONTROL METHODS

The goal of the use of the neural networks in control is to determine the controller outputs (process inputs), given the current state of the process to be controlled.

There are four principal aproaches [9] in the use of the neural networks for control tasks, which can be seen in the literature:

- direct neural network based control,
- inverse neural network based control,
- model - based control,
- supervisory control.

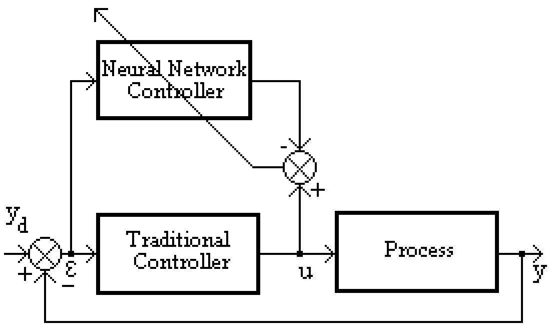


Figure 1. Training a direct neural controller

By direct neural network control, we mean the situation when the control operation is performed by a neural network, which replaces the traditional controller in the general scheme of feedback control. The network will determine the controller action u that reduces the error value ε ($y-y_d$), which represents the input in the network. Before acting as a controller, the network must be trained with data. One such control method is used in [7]. The authors trained a network to mimic a traditional controller (a PID controller for example), the process being controlled by the traditional controller, as shown in figure 1. The input of the traditional controller is the input in the network, and the difference between the output of the traditional controller (control action) and the output of the network is the error signal used for weights adjustment. After this learning phase, the neural network will replace the traditional controller, acting as on-line controller. The neural controller can be made adaptive if the neural network will further learn on-line, by weight adjustment, in order to cope with the drifts in the process.

In the inverse control, the network is trained to learn the inverse dynamics of the process. In this scheme, the output of the plant is the input to the neural network, and the plant's input is the target output of the neural network (see figure 2). Generally, the network can have as input, the past (y(k-i)), current (y(k)) and future (y(k+i)) outputs of the process, as well as the past inputs (u(k-i)), having as the network output, the current input of the process (u(k)). An example of creating an inverse model is shown in [3]. Another alternative to the creation of an inverse model of the process is the inverting of the forward model of the process [9][5], by searching the necessary inputs of the forward model that produce the desired outputs. The forward model of a plant is obtained by training the neural network with the plant's inputs as inputs in the network, and with the plant's outputs as outputs of the network.

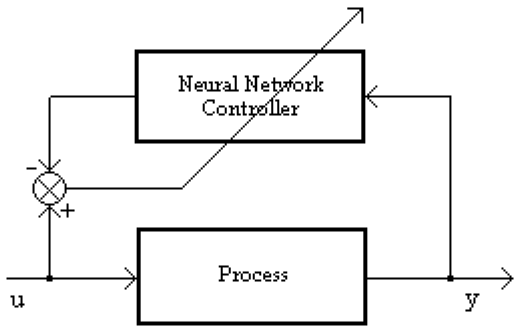


Figure 2. Training of an inverse neural controller

The model - based control strategies predict the future output of the process, using a model of that process, and then try to minimize the error between the model and the process. Traditionally, it is used a linear model of the process, with the performance degrading when attempts to control nonlinear systems. Due to the nonlinear nature of the neural networks, it is possible to identify good models, and to integrate these neural models, with better results, in a general model-based control scheme. Manchanda et al. [4] introduced neural network models in an IMC (Internal Model Control) scheme, developed by Garcia and Morari (see figure 3). They used a model of the process in parallel with the process, in order to produce the feedback signal to the controller. The result of incorporating neural networks into this structure is that the controller must be the inverse of the model of the process [4][9].

$$u = G_m^{-1} * [y_d-(y-y_m)]$$

where G_m is the forward neural model of the process. The inverse model G_m^{-1} (the controller) can be learned by training, or can be obtained by inverting the forward model G_m. The forward model can be updated on - line to improve the modelling performance, which leads, by an inversion technique, to an updated inverse model, and then better control performances.

The development of a neural model for a process implies the collection of a representative set of input - output data from the process. The most efficient way to create a neural model for a process is to train the neural network off-line, while the process is controlled by a traditional controller. When the difference between the process and the model decreases sufficiently, the neural model can be integrated in a model - based control scheme and, eventually, can be adjusted on-line after that.

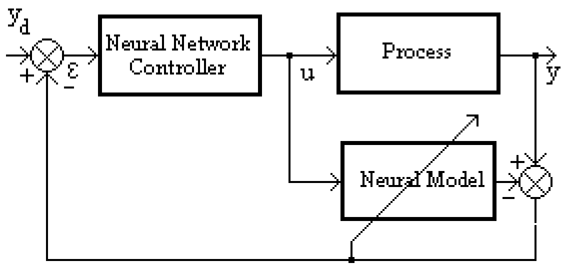


Figure 3. Internal model control scheme

In supervisory control, one controller (supervisory controller) sets the parameters of another controller (subordinate controller), in order to optimize the control performance. Many authors [8][10] used a neural network to control the parameters of a traditional controller, such a PD, PI, or PID controller. Given the error signal and the current state of the process, the neural network will compute the parameters K_c, T_I, T_D of the traditional controller. In this way, the controller becomes adaptive. The advantage of this structure is that the confidence in the controller is enhanced, comparatively with other neural control schemes, in this case the controller being in a more familiar form. Additionally, a PI controller, for example, is a robust controller, since it uses a rough model of the process, represented by its tuning parameters As a result, it will give better performance when the mismatch between the process and the model increases, in comparison with model - based controllers.

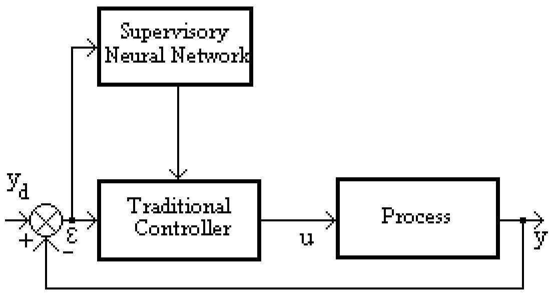


Figure 4. Supervisory control

3. THE INVERSION METHOD

As described in section 2, both for inverse neural control and internal model neural control, the inversion of the forward model of the process is needed. Two inversion methods were presented in [9]. This section presents a more computationally efficient inversion method of a neural network than previously reported methods in the literature.

Given a three layered neural network, which represent the direct model of a process, with q the number of the network inputs, h the number of hidden nodes, and r the number of network outputs. Within the q inputs of the network, f inputs are considered to be fixed inputs (past inputs and outputs of the process), and we have to determine, by a searching procedure, the remaining p inputs (p+f=q). Given the output vector y, we have to find the input vector u which produces the output y. By u it is denoted the vector of the p unfixed inputs of the network, and by u_f the vector of the f fixed inputs of the network.

We have the following relations:

$$y' = f^{-1}(y) - \theta^y$$
$$W^{yx} x = y' \qquad (1)$$

where f is the nonlinear activation function of the network nodes, x the output vector of the hidden nodes, W^{yx} is the weight matrix between hidden layer and output layer, θ^y is the bias vector of the output layer.

If W^{yx} is a squared matrix, then it is possible to calculate the input u:

$$x = (W^{yx})^{-1} y'$$
$$x' = f^{-1}(x) - \theta^x$$
$$W^{xu} u = x' - W_f^{xu} u_f \qquad (2)$$

where W^{xu} is the weight matrix between unfixed inputs and hidden layer, W_f^{xu} is the weight matrix between fixed inputs and hidden layer, θ^x is the bias vector of the hidden layer. When the hidden and the output layers have exactly the same number of nodes (h=r), the inversion of the network consists of solving two linear equation systems ((1) and (2)). In the following, consider d the vector d = $W_f^{xu} u_f$.

The general case of most neural models of real-world processes is represented by neural networks with h > r. In this case, it is possible to write the matrix W^{yx}, by Jordan-Gauss elimination, in the following form:

$$
\begin{array}{c}
\begin{array}{cccccc} x_1 & x_2 & x_r & x_{r+1} & x_h & y' \end{array} \\
\left[\begin{array}{ccc|c|c}
1 & 0 \ldots\ldots\ldots & 0 & & \\
0 & 1 \ldots\ldots\ldots & 0 & & \\
\ldots & \ldots\ldots\ldots & \ldots & C & Y^* \\
\ldots & \ldots\ldots\ldots & \ldots & & \\
0 & 0 \ldots\ldots\ldots & 1 & &
\end{array}\right]
\end{array}
$$

Partitioning the weight matrix W^{xu} and the vectors x, θ^x and d, as given below:

$$x = \begin{bmatrix} x' \\ x'' \end{bmatrix} \text{ where}$$

$x' = [x_1, x_2, \ldots\ldots, x_r]^t$ and

$x'' = [x_{r+1}, \ldots., x_h]^t$

$$W^{xu} = \begin{bmatrix} W^{x'u} \\ W^{x''u} \end{bmatrix} \quad \theta^x = \begin{bmatrix} \theta^{x'} \\ \theta^{x''} \end{bmatrix} \quad d = \begin{bmatrix} d^{x'} \\ d^{x''} \end{bmatrix} \text{ where}$$

$W^{x'u}$ is a r x p matrix, and

$W^{x''u}$ is a (h - r) x p matrix

we have the relations:

$$x' = f(W^{x'u}u + d^{x'} + \theta^{x'})$$
$$x'' = f(W^{x''u}u + d^{x''} + \theta^{x''})$$

The input u, which produces the desired output y, is the solution of the equation:

$$x' = y^* - C x''$$

equivalent with:

$$y^* - Cf(W^{x''u}u + d^{x''} + \theta^{x''}) = f(W^{x'u}u + d^{x'} + \theta^{x'})$$

Expanding the nonlinear function f, through a Taylor series around the point u_k, the following relations is obtained:

$$y^* - C(f(W^{x''u}u_k + d^{x''} + \theta^{x''}) + f'(W^{x''u}u_k + d^{x''} + \theta^{x''})$$
$$(W^{x''u}u - W^{x''u}u_k)) = f(W^{x'u}u_k + d^{x'} + \theta^{x'}) +$$
$$f'(W^{x'u}u_k + d^{x'} + \theta^{x'})(W^{x'u}u - W^{x'u}u_k)$$

where:

$$f(W^{x''u}u_k + d^{x''} + \theta^{x''}) = \begin{bmatrix} f(W_1^{x''u}u_k + d_1^{x''} + \theta_1^{x''}) \\ f(W_2^{x''u}u_k + d_2^{x''} + \theta_2^{x''}) \\ . \\ . \\ f(W_{h-r}^{x''u}u_k + d_{h-r}^{x''} + \theta_{h-r}^{x''}) \end{bmatrix}$$

$$f(W^{x'u}u_k + d^{x'} + \theta^{x'}) = \begin{bmatrix} f(W_1^{x'u}u_k + d_1^{x'} + \theta_1^{x'}) \\ f(W_2^{x'u}u_k + d_2^{x'} + \theta_2^{x'}) \\ . \\ . \\ f(W_r^{x'u}u_k + d_r^{x'} + \theta_r^{x'}) \end{bmatrix}$$

$$f'(W^{x''u}u_k + d^{x''} + \theta^{x''}) = \mathrm{Diag}\begin{bmatrix} f'(W_1^{x''u}u_k + d_1^{x''} + \theta_1^{x''}) \\ f'(W_2^{x''u}u_k + d_2^{x''} + \theta_2^{x''}) \\ . \\ . \\ f'(W_{h-r}^{x''u}u_k + d_{h-r}^{x''} + \theta_{h-r}^{x''}) \end{bmatrix}$$

$$f'(W^{x'u}u_k + d^{x'} + \theta^{x'}) = \mathrm{Diag}\begin{bmatrix} f'(W_1^{x'u}u_k + d_1^{x'} + \theta_1^{x'}) \\ f'(W_2^{x'u}u_k + d_2^{x'} + \theta_2^{x'}) \\ . \\ . \\ f'(W_r^{x'u}u_k + d_r^{x'} + \theta_r^{x'}) \end{bmatrix}$$

The notation DiagV, where V is a vector, specifies a diagonal matrix with the components of vector V on the main diagonal and all other matrix elements zero. Solving the equation given above, we obtain:

$$y^* - Cf(W^{x''u}u_k + d^{x''} + \theta^{x''}) - f(W^{x'u}u_k + d^{x'} + \theta^{x'}) =$$
$$(f'(W^{x'u}u_k + d^{x'} + \theta^{x'})W^{x'u} + Cf'(W^{x''u}u_k + d^{x''} + \theta^{x''})$$
$$W^{x''u})(u - u_k)$$

and the following iterative relation:

$$\begin{aligned} u = u_k + [&f'(W^{x'u}u_k + d^{x'} + \theta^{x'})W^{x'u} + \\ &Cf'(W^{x''u}u_k + d^{x''} + \theta^{x''})W^{x''u}]^{-1} \\ &[y^* - Cf(W^{x''u}u_k + d^{x''} + \theta^{x''}) - f(W^{x'u}u_k + d^{x'} + \theta^{x'})] \end{aligned} \tag{3}$$

The matrix which must be inverted in the relation given above is a r x p matrix. If the number of unfixed network inputs is different than the number of network outputs, the inverse from the equation (3) must be replaced with the suitable pseudo-inverse.

Changing the notations as follows:

$$f'(W^{x''u}u_k + d^{x''} + \theta^{x''}) = \begin{bmatrix} f'(W_1{}^{x''u}u_k + d_1{}^{x''} + \theta_1{}^{x''}) \\ f'(W_2{}^{x''u}u_k + d_2{}^{x''} + \theta_2{}^{x''}) \\ . \\ . \\ f'(W_{h-r}{}^{x''u}u_k + d_{h-r}{}^{x''} + \theta_{h-r}{}^{x''}) \end{bmatrix}$$

$$f'(W^{x'u}u_k + d^{x'} + \theta^{x'}) = \begin{bmatrix} f'(W_1{}^{x'u}u_k + d_1{}^{x'} + \theta_1{}^{x'}) \\ f'(W_2{}^{x'u}u_k + d_2{}^{x'} + \theta_2{}^{x'}) \\ . \\ . \\ f'(W_r{}^{x'u}u_k + d_r{}^{x'} + \theta_r{}^{x'}) \end{bmatrix}$$

the iterative relation (3) becomes:

$$\begin{aligned} u = u_k + [&f'(W^{x'u}u_k + d^{x'} + \theta^{x'})oW^{x'u} + \\ &C(f'(W^{x''u}u_k + d^{x''} + \theta^{x''})oW^{x''u})]^{-1} \\ &[y^* - Cf(W^{x''u}u_k + d^{x''} + \theta^{x''}) - f(W^{x'u}u_k + d^{x'} + \theta^{x'})] \end{aligned} \tag{4}$$

The operator o from the relation given above multiplies a row of a matrix with the corresponding element of the vector. In this way the complexity of the calculus is significantly reduced, when the iterative relation (4) is implemented, instead of relation (3). This aspect is useful because the inversion calculus is made on-line, and will help to not compromise the control performances when the control actions must be taken in critical time.

Previously, it was presented a method of inverting a three layered neural network. Any real-world process can be modelled by a three layered network. Be h the number of needed (or desired) hidden nodes to model a process. If the user still wish to use a multilayer neural network (with more than one hidden layer), and in order to reduce the inversion calculus, we propose to chose m+1 hidden layers with r nodes on each layer, where:

$$h = m * r + t \quad r \le t \le 2r,\ m \in N$$

Given the output y, it is possible to calculate in one iteration the outputs of all hidden layers, until the second hidden layer, solving m determined linear equation systems. Now, once the output of the second hidden layer is calculated, the input u, which produces the desired output y, can be determined with the inversion method presented previously in this section for a three layered neural network.

4. SIMULATION RESULTS

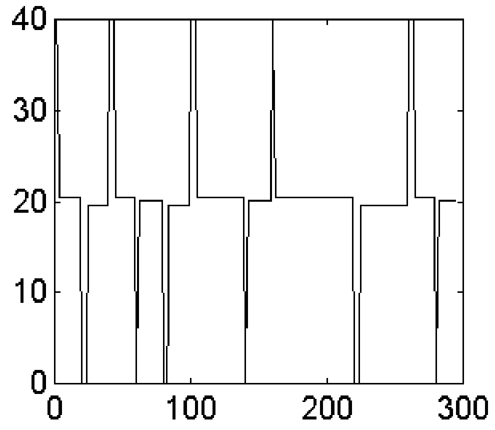
In order to test the inversion method described in section 3, the first case study is made on a common benchmark process, the liquid tank, described by the following equation:

$$\frac{dy}{dt} = \frac{1}{A}[u + d - K\sqrt{y}]$$

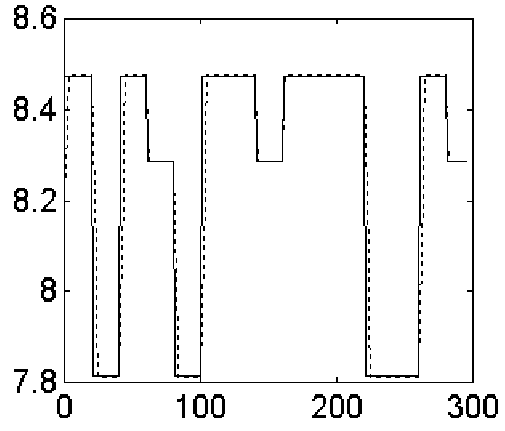
where K=7, A=30, $u \in [0;40]$, $y \in [0;10]$, $d \in [0;10]$

The training data set contains 300 examples, also the testing data set. The hidden layer contains three nodes. The forward neural model of the process was developed, and this forward model is used in the inverse neural control structure shown in figure 7, where P is the process and C is the controller. The forward neural model contains, as network inputs, the current input of the process u(k-1) and the previous output of the process y_s(k-1), and as network output, the output of the system y_s(k).

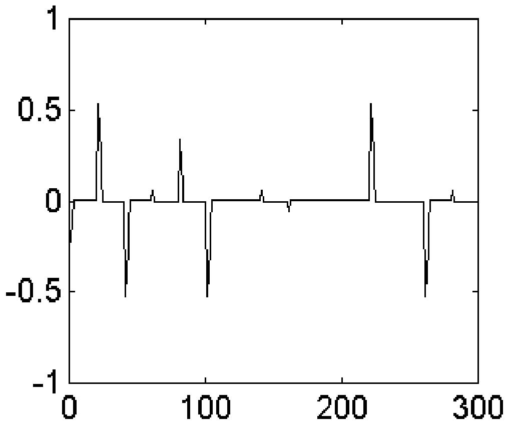
The command u (controller output) is calculated by inverting the forward model with the method described in section 3, where one input, y_s(k-1), is known. Figure 5 presents the behaviour of the inverse controller to setpoint changes, and figure 6 presents the controller performances regarding the disturbance rejection.



a) the controller output - u

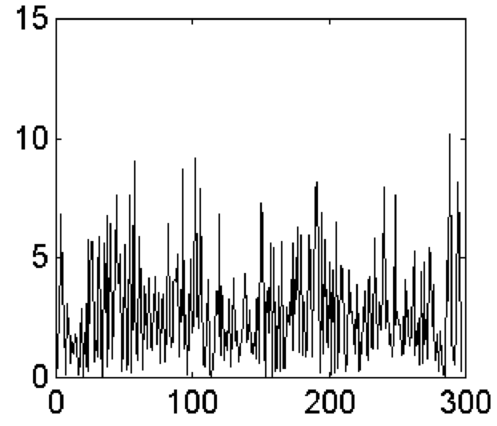


b) the process output y_s (dotted) and setpoint y_d.

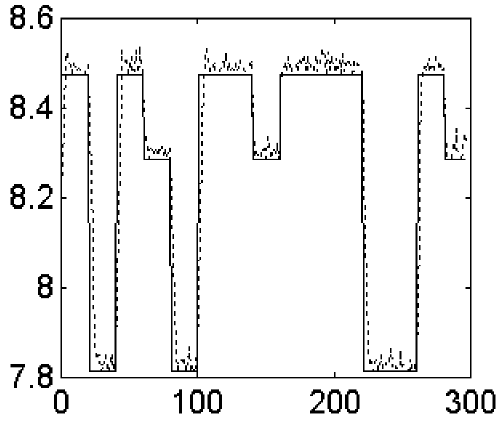


c) the error $y_s - y_d$

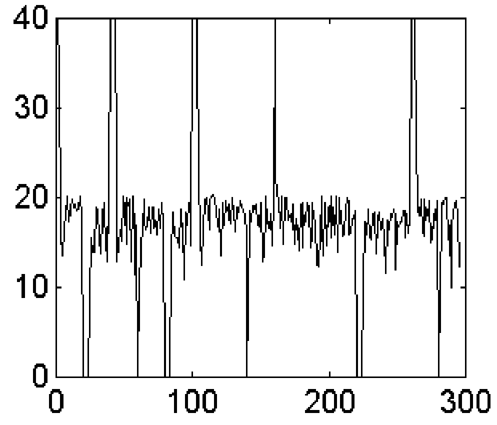
Figure 5 The behaviour of the inverse controler to setpoint changes



a) the perturbation d



b) the process output y_s (dotted) and setpoint y_d



c) the command u with perturbation

Figure 6 The disturbance rejection performances

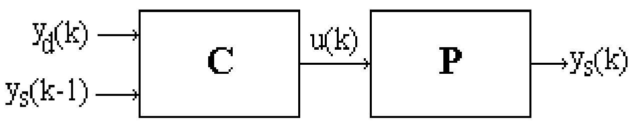


Figure 7. Inverse neural control structure

As shown in figures 5 and 6, the inverse neural controller performs very well on setpoint changes, but not so well on disturbance rejection. In order to reject the perturbation, we trained a neural network with three inputs: u(k-1), y_s(k-1), d(k-1), and one output: y_s(k). The controller output is determined by inverting the previously developed forward model, considering two fixed inputs (the inputs corresponding to y_s(k-1) and d(k-1)). The perturbation is completely rejected and the results are the same with those from figure 5. So, if the perturbation can be measured, it is possible to reject it, if we consider the perturbation as input to the neural network which represents the forward model of the process.

The second case study chosen to test the inversion method presented in section 3 is to control the speed of a DC motor, using an internal model-based control structure shown in figure 3.

The DC motor is described by the following equation:

$$\frac{di}{dt} = \frac{ur}{L} - \frac{60}{2 \cdot \Pi} \cdot \frac{K_e \cdot K_\Phi}{L} \cdot \omega \cdot i_e - \frac{r}{L} \cdot i$$

$$\frac{di_e}{dt} = \frac{u_e}{L_e} - \frac{re}{Le} \cdot i_e$$

$$\frac{d\omega}{dt} = \frac{K_m \cdot K_\Phi}{J} \cdot i \cdot i_e - \frac{M_r}{J}$$

where:

ω -	the angular speed
J -	the inertial torque
u_r -	the induced command voltage
u_e -	the inducer command voltage
i -	the induced current
i_e -	the inducer current

The network, which implements the forward model, was trained having as network input the vector $[\omega(k-1)\ \omega(k-2)\ \omega(k-3)\ u_r(k)\ u_r(k-1)\ u_r(k-2)\ u_r(k-3)]^T$ and as network output the current speed $\omega(k)$. The hidden layer contains 50 neurons. The successful identification of the DC motor is proved by the good performances of the internal model control structure, shown in the following figures.

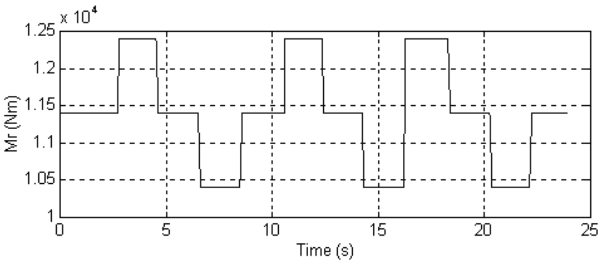


Figure 8. a) Resistant torque

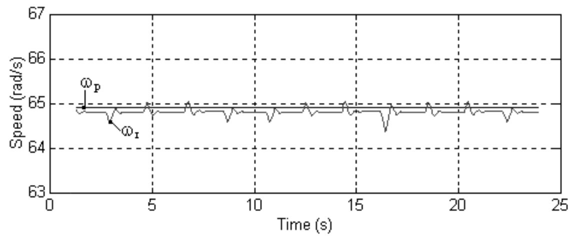


Figure 8. b) Prescribed speed (ω_p) and real speed (ω_r).

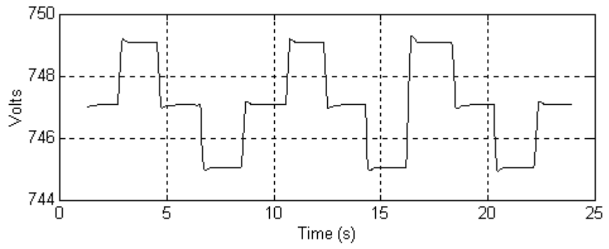


Figure 8. c) Control voltage

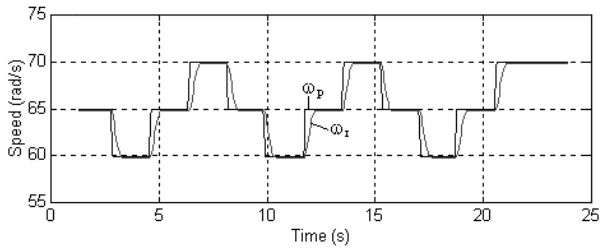


Figure 9. a) Prescribed speed (ω_p) and real speed(ω_r).

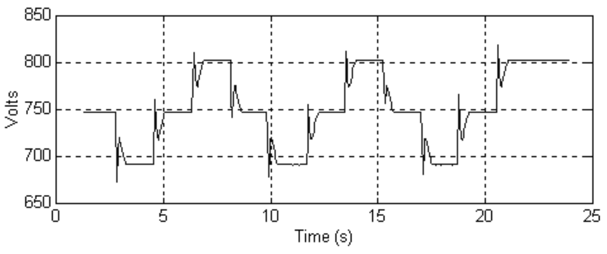


Figure 9. b) Control voltage

The neural network controller from figure 3 is replaced with a program procedure, which calculates the controller output by inverting the forward neural model previously developed.

We considered two aspects in the simulation:

- to keep the speed constant in the presence of some variations of the resistant torque, shown in figures 8a,b,c;
- to change the motor speed according with an imposed variation law (figures 9a,b).

5. CONCLUSION

A method of inverting a neural network was proposed in this paper. This method is useful in the inverse neural control structure and neural model based control structure, when the inversion calculus is made on-line, and the network input, which produces the desired network output, must be obtained in critical time.

REFERENCES

[1] B. Kosko, *Neural Networks and Fuzzy Systems,* Prentice - Hall International Inc., 1992.

[2] A. J. Krijsman, *Artificial Intelligence in Real-Time Control*, PhD. thesis, Universiteit Delft, 1993.

[3] E. Levin, R. Gewirtzman, and G. F. Inbar, “Neural Network Architecture for Adaptive System Modelling and Control”, *Neural Networks,* No 4(2), pp. 185-191, 1991.

[4] S. Manchanda, M.J. Willis, M.T. Tham, C. DiMassimo, and G.A. Montague, “An Appraisal of Nonlinear Control Philosophies for Application to a Biochemical Process”, *Proceedings of the 1991 American Control Conference*, San Diego USA, 1991, pp. 1317-1322.

[5] V. Palade, *Hybrid Expert Systems for Process Control*, PhD. thesis, “Dunarea de Jos” University of Galati – Romania, 1999.

[6] V. Palade, V. Mazilescu and S. Bumbaru, “Special issues on the use of artificial neural networks in the intelligent control systems”, *The Annals of “Dunarea de Jos” University of Galati*, Vol. 3, pp. 34-38, 1992.

[7] D. Psaltis, A. Sideris, and A. A. Yamamura, “A Multilayered Neural Network Controller”, *IEEE Control Systems Magazine,* No. 8, pp. 17-21, 1988.

[8] S. K. Sanjay and A. Guez, “Adaptive Pole Placement for Neurocontrol”, *Proc. of the International Joint Conference on Neural Networks*, Washington D.C. USA, 1990, vol. 2, pp. 563-569.

[9] G. M. Scott, *Knowledge - Based Artificial Neural Networks for Process Modelling and Control*, PhD. thesis, University of Wisconsin, 1993.

[10] R. W. Swiniarski, “Novel Neural Network Based Self - Tuning PID Controller which Uses Pattern Recognition Technique”, *Proc. of the American Control Conference*, San Diego USA, 1990, vol. 3, pp. 3023-3024.