

# An approach of DC driving system neural control by inverting forward model

Vasile Palade, Gheorghe Puscasu, Daniel-Ciprian Neagu

**Abstract**-- This paper proposes 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 is tested to a D.C. motor control problem. The forward neural model of the D.C. motor is developed, and the controller output is calculated by on-line inverting of the forward model.

**Index Terms**-- neural networks, neural control, inverse model.

## I. 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. Neural networks can approximate any linear or nonlinear mapping between the input and the output of the system and they are able to learn in order to perform this approximation. 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 [1][2][3][4][5][9] that neural networks can solve complex and difficult control tasks, where traditional control methods fails. 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 approaches [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 and supervisory control.

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  $\varepsilon$ , which represents the input in the network. Before acting as a controller, the network must be trained with data. In [7] the authors trained a network to mimic a traditional PID controller. After the training phase, the neural network

will replace the traditional controller, acting as an on-line controller. 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. 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][6], 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. In supervisory control, the neural network [8] is trained to control the parameters of a traditional controller, such a PD, PI, or PID controller.

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 traditional model-based control schemes. In [4] the authors introduced neural network models into an IMC (Internal Model Control) scheme, as shown in figure 1. 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]. The inverse model (the controller) can be learned by training, or can be obtained by inverting the forward model of the process. The forward model can be updated on - line to improve the modeling 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 paper is structured as follows. Section II 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, optimizing 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 authors are with the Faculty of Electrical Engineering, "Dunarea de Jos" University of Galati, Domneasca Str. 47, 6200, Galati - Romania.

the inversion method, section III presents a case study based on a DC motor control problem. The paper is ending with some conclusions.

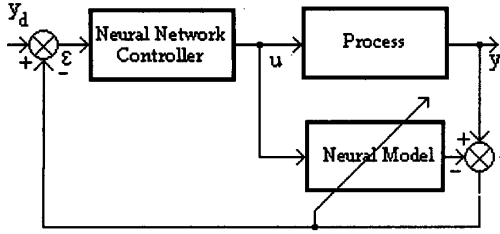


Figure 1. Internal model control scheme with NN

## II. INVERTING NEURAL MODEL

As described in section I, 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:

$$\begin{aligned} y' &= f^{-1}(y) - \theta^y \\ W^{yx} x &= y' \end{aligned} \quad (1)$$

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,  $\theta^y$  is the bias vector of the output layer.

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

$$\begin{aligned} x &= (W^{yx})^{-1} y' \\ x' &= f^{-1}(x) - \theta^x \\ W^{xu} u &= x' - W_f^{xu} u_f \end{aligned} \quad (2)$$

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,  $\theta^x$  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:

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

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

$$\begin{aligned} x &= \begin{bmatrix} x' \\ x'' \end{bmatrix} \text{ where} \\ x' &= [x_1, x_2, \dots, x_r]^t \text{ and} \\ x'' &= [x_{r+1}, \dots, x_h]^t \end{aligned}$$

$$\begin{aligned} W^{xu} &= \begin{bmatrix} W^{x'u} \\ W^{x''u} \end{bmatrix} \quad \theta^x = \begin{bmatrix} \theta^{x'} \\ \theta^{x''} \end{bmatrix} \quad d = \begin{bmatrix} d^{x'} \\ d^{x''} \end{bmatrix} \text{ where} \\ W^{x'u} &\text{ is a } r \times p \text{ matrix, and} \\ W^{x''u} &\text{ is a } (h-r) \times p \text{ matrix} \end{aligned}$$

we have the relations:

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

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

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

equivalent with:

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

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

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

where:

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

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

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

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

The notation  $\text{Diag}V$ , 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:

$$y^* - Cf(W^{x''}u_k + d^{x''} + \theta^{x''}) - f(W^{x'}u_k + d^{x'} + \theta^{x'}) = (f'(W^{x'}u_k + d^{x'} + \theta^{x'})W^{x''}u_k + Cf'(W^{x''}u_k + d^{x''} + \theta^{x''}) - f'(W^{x'}u_k + d^{x'} + \theta^{x'}))^{-1} (y^* - Cf(W^{x''}u_k + d^{x''} + \theta^{x''}) - f(W^{x'}u_k + d^{x'} + \theta^{x'}))$$

and the following iterative relation:

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

The matrix which must be inverted in the relation given above is a  $r \times p$  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:

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

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

the iterative relation (3) becomes:

$$u = u_k + [f'(W^{x'}u_k + d^{x'} + \theta^{x'}) \circ W^{x''}u_k + Cf'(W^{x''}u_k + d^{x''} + \theta^{x''}) - f'(W^{x'}u_k + d^{x'} + \theta^{x'})]^{-1} (y^* - Cf(W^{x''}u_k + d^{x''} + \theta^{x''}) - f(W^{x'}u_k + d^{x'} + \theta^{x'})) \quad (4)$$

The operator  $\circ$  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:

$$h = m * r + t \quad r \leq t \leq 2r, \quad m \in \mathbb{N}$$

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.

### III. SIMULATION RESULTS

The case study chosen to test the inversion method presented in section II is to control the speed of a DC motor, using an internal model-based control structure shown in figure 1. The DC motor is described by the following equation:

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

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

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

where:  $\omega$  is 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.

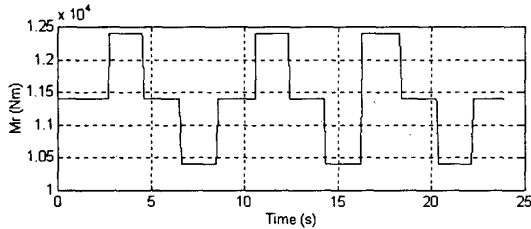


Figure 2. Resistant torque

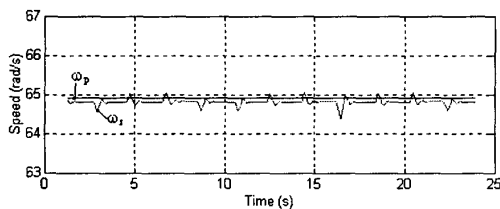


Figure 3. Prescribed speed ( $\omega_p$ ) and real speed ( $\omega_r$ ).

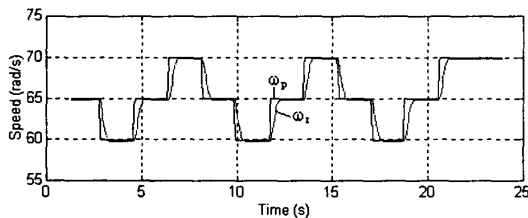


Figure 4. Prescribed speed ( $\omega_p$ ) and real speed ( $\omega_r$ ).

The network, which implements the forward model, was trained having as network input the vector  $[\omega(k-1) \omega(k-2) \omega(k-3) u_r(k-1) u_r(k-2) u_r(k-3)]^T$  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 figures 3 and 4. The neural network controller from figure 1 is replaced with a program procedure, which calculates the controller output by inverting the forward neural model previously developed. Two aspects were taken into consideration in the simulation:

- to keep the speed constant in the presence of some variations of the resistant torque (see figure 2), shown in figure 3;
- to change the D.C. motor speed according with an imposed variation law (figure 4).

Figure 5 illustrates the behaviour of an inverse controller in the presence of some torque variation. As shown, the D.C. motor speed is maintain almost constant around the nominal value. Also the output of the controller is obtained by inverting the forward model previously developed.

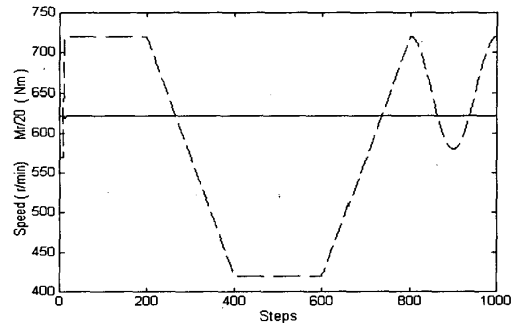


Figure 5. The resistant torque and obtained speed for inverse controller.

#### IV. 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. Krijnsman, *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, G. Puscasu and D. Neagu, "Neural network-based control by inverting neural models", *Control Engineering and Applied Informatics*, vol.1, No. 1, pp. 25-31, dec. 1999.
- [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] 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.
- [9] G. M. Scott, *Knowledge - Based Artificial Neural Networks for Process Modelling and Control*, PhD. thesis, University of Wisconsin, 1993.