NEURAL NETWORK BASED INERTIA IDENTIFICATION OF ELECTRICAL DRIVE SYSTEMS

LILIANA DAFINCA
"TRANSILVANIA" University of Braşov, 29 Eroilor Blvd., 2200 Braşov, ROMÂNIA
Tel.: 0040-68-418836, Fax: 0040-68-143116
E-mail: *dafinca@unitbv.ro*

Abstract. The paper advances the using of a neural network to estimate the total inertia moment of the motor and mechanical system. Identification of inertia moment is an important objective for designing the speed control system. The inertia is identified by the pattern recognition of the neural network trained through step command tracking speed responses, for various inertia values. To obtain the speed responses due to step command change, an acquisition system has been developed. The optimal topology of the neural network has been chosen using an original software tool for designing of neural networks based applications [1].

1. Introduction

The basic idea of the identification method is the relation between the shape of the drive speed response for the stepwise speed command and the value of the inertia moment. Therefore, having the speed response for the stepwise speed command, one can somehow obtain the inertia moment. This method belongs to the pattern recognition category of applications. The neural networks are adequately suited for this type of applications.

Actually, a neural network is a machine like the human brain with the properties of learning capability and generalization. It is used to represent functions as weighted sums of nonlinear terms. The learning capability makes it able to approximate very complicated nonlinear functions, and consequently a neural network can be a universal approximation of almost any dynamic system. The generalization property allows to train neural networks with a limited training data set. Although these networks eliminate the need for mathematical models, they require a lot of training to understand the model of a plant or a process.

2. Feedforward Neural Networks

The artificial neural networks use a dense interconnection of neurons that correspond to computing nodes. Each node performs the multiplication of its input signals by constant weights, sums up the results, and maps the sum to a nonlinear function (activation function). The result is then transferred to its output.

A feedforward neural network, which is most widely used in applications [3], is organized in layers: an input layer, one or more hidden layers, and an output layer. The input values are converted through the normalizer gains. No computation is performed in the input layer: the signals are directly supplied to the first layer. Hidden and output neurons generally have a sigmoid activation function. The mathematical model of a neuron is given by Fig. 1 and relations (1) and (2).

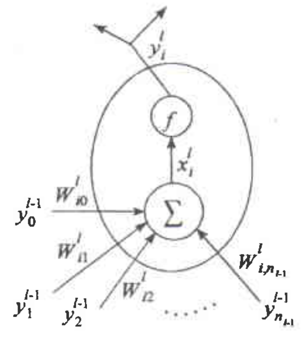

Fig. 1. The model of a neuron

$$y_i^l = f(x_i^l) = \frac{1}{1 + e^{-x_i^l \beta}} \quad i = 1,2,\dots,n_l \tag{1}$$

$$x_i^l = \sum_{j=0}^{n_{l-1}} W_{ij}^l \cdot y_j^{l-1} \tag{2}$$

The notations from (1) and (2) are:

l	the number of layers excepting the input (2 or 3);
y_i^l	the output of neuron i at layer l;
x_i^l	the sum of the inputs to node i at layer l;
f	the activation function of the neuron (here it is a sigmoid);
β	the gain of the sigmoid function;
n_l	the number of neurons at layer l.
W_{ij}^l	the connection weight from the jth node at layer l-1 to the ith node at layer l.

Fig. 2 shows the topology of the neural network used for the identification of the inertia moment. A three-layer topology has been used. The inputs of the neural network are the time basis plant speeds for the stepwise command and the output is the estimated inertia moment. After computation, the output is brought back to the actual values through denormalizer gains. The number of hidden nodes has been chosen by trials. Using normalization-denormalization techniques, the robustness of the network is enhanced.

The knowledge in a neural network is acquired through a learning algorithm, which performs the adaptation of weights of the network iteratively until the error between the target vectors and output of the network falls below a certain error goal.

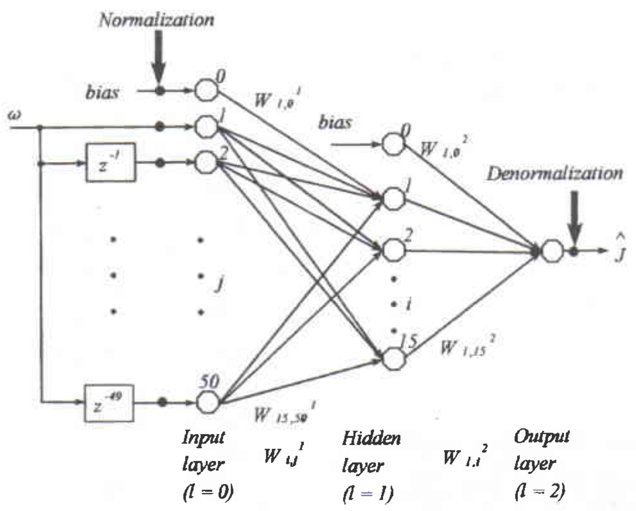

Fig. 2. The neural network used for inertia identification

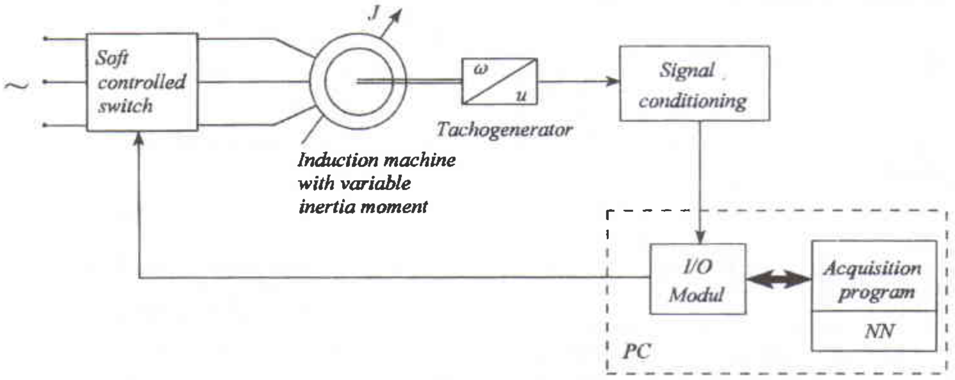

Fig. 3. The block diagram of the system used for inertia identification

3. Identification Method of Inertia

The total inertia moment of the motor and mechanical system has been obtained using the system shown in Fig. 3. A multifunction I/O board (PCI-MIO-E1) allows 12 bit A/D conversions with a sampling rate of 1.25 MS/s. The FIFO buffer of PCI-MIO-E1 has had 512 samples.

The parameters of the induction motor used for experiments are:

$P_N = 1{,}1$ kW;	$U_N = 220$ V;
$I_N = 2{,}93$ A;	$f = 50$ Hz;
$p = 4$ pole pairs;	$n_N = 1500$ rpm;
$J_M = 0.017$ kg·m^2;	$J_{tacho} = 0{,}388$ kg·cm^2.

The teachers of the neural network are the speed samples at the power on under the various system inertia.

Fig. 4 presents these teachers obtained by simulation for three different inertia moments, where J_0 denotes:

$$J_0 = J_M + J_{tacho}.$$

An experimental acquisition is shown in Fig. 5. It can be noticed that the simulation and the acquisition waveforms are similarly.

The acquisition program has been carried out using LabWindows CVI, a Rapid Application Development software.

For experiments, the inertia moment has been increased using mechanical systems with known inertia moments.

The neural network from Fig. 2 can also be used in the inertia identification of the speed control system in Fig. 6. The control model in Fig. 6 has the same PI controller as that of the plant and the teachers of the neural network are the time basis speed responses under the various system inertia shown in Fig. 7.

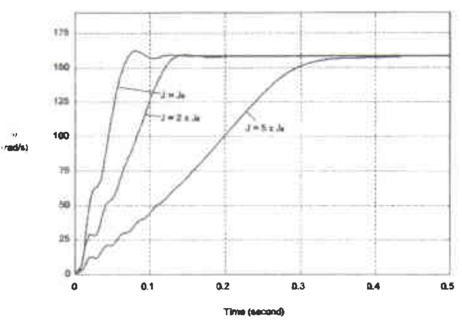

Fig. 4. Drive speed for different inertia moment

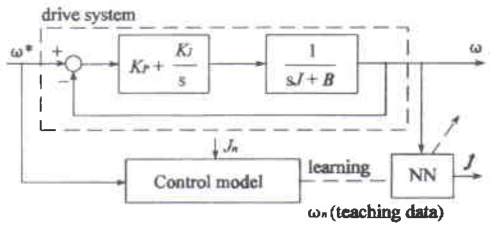

Fig. 6. Inertia identification for a closed loop system

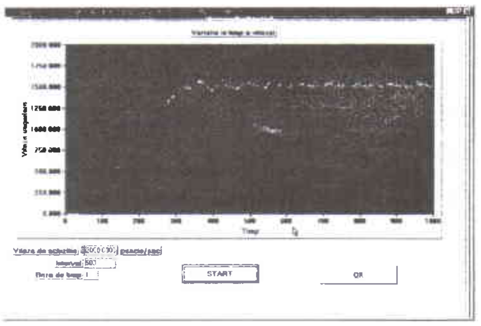

Fig. 5. An experimental acquisition used as teacher for the neural network

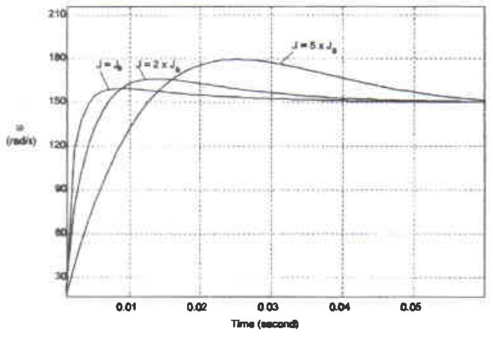

Fig. 7. Step speed responses waveformes

4. The Backpropagation Training Algorithm

The most popular learning algorithm for feedforward neural networks is the backpropagation algorithm, which consists of a forward and backward action. In the first, the signals are propagated through the network layer by layer. An output vector is thus generated and subtracted from the desired output vector. The resultant error vector is propagated backward in the network and serves to adjust the weights in order to minimize the output error.

The time required to train a neural network depends on the size of the training data set, the size of the neural network and the training algorithm. The standard version of the backpropagation algorithm is very slow and requires a large number of iterations. Improved versions of this algorithm permit the reduction of the number of iterations. These versions are the backpropagation with momentum and the adaptive learning rate. The neural-network development system, used here [1], combines both accelerating methods of the training process and allows the variation of the momentum factor while the training performs.

The connection weights are obtained based on a training data set. Training data consists of input-output pairs generated by the process which the network is to emulate. For the proposed application, the set of training patterns is:

$$\begin{aligned} &\{(X_i^0[50], O_i^2[1]) \mid i = 1,2,\dots,M\} \\ &X_i^0[50] = [x_i^0[1],\dots,x_i^0[50]] \\ &O_i^2[1] = [o_i^2[1]] \end{aligned} \tag{3}$$

where:

$X_i^0[50]$	is the input vector at layer 0 according to the pattern i (50 samples of step speed response);
$O_i^2[1]$	is the output vector at layer 2 provided by the pattern i (the inertia moment for $X_i^0[50]$ step speed response);
M	is the number of training patterns (M = 3 for this application).

The backpropagation training algorithm [3] can be summarized as follows:

Step 1.

At the beginning the connection weights have small values obtained randomly.

Step 2.

For a training pattern ($X_i^0[50]$, $O_i^2[1]$) chosen randomly, the weights of the net are recursively calculated.

First, the output of the network:

$$Y_i^2[1] = [y_i^2[1]]$$

is calculated according to (1), (2) by forward computation using the corresponding inputs:

$$X_i^0[50] = [x_i^0[1], x_i^0[2], ..., x_i^0[50]]$$

Then, the error signals for the third layer neurons are obtained using the relation:

$$\delta_i^2[1] = \beta y_i^2[1](1 - y_i^2[1])(o_i^2[1] - y_i^2[1]) \qquad (4)$$

where:

$\delta_i^l[j]$ is the error signal for the neuron j at layer l provided by the pattern i;

$o_i^l[j]$ is the desired output for the neuron j at layer l provided by the pattern i;

$y_i^l[j]$ is the real output for the neuron j at layer l calculated with current weights.

The error signals for the second layer are obtained in a similar way:

$$\delta_i^1[j] = \beta y_i^1[j](1 - y_i^1[j]) \sum_i \delta_i^2[j] \cdot W_{qi}^2[j] \qquad (5)$$

Error signals are used to adjust the weights of the neural network. So the weights are updated according to the relations:

$$\begin{aligned} W_{qj}^2(i+1) &= W_{qj}^2(i) + \eta\, \delta_j^2(i) y_q^1(i) + \alpha \Delta W_{qj}^2(i) \\ W_{mq}^1(i+1) &= W_{mq}^1(i) + \eta\, \delta_q^1(i) x_m^0(i) + \alpha \Delta W_{mq}^1(i) \end{aligned} \qquad (6)$$

where the notations are:

α	is the momentum factor chosen by trials as 0.9;
η	is the learning rate chosen by trials between 0.05 - 0.25;
$W_{qj}^l(i)$	is the connection weight at the step i from the jth node at layer l to the qth node at layer l-1;
$\Delta W_{qj}^l(i)$	is the momentum, i.e.: $\Delta W_{qj}^l(i) = W_{qj}^l(i) - W_{qj}^l(i-1)$

Step 3.

Error is calculated as in relation (7).

$$E[i] = \frac{1}{2}(o_i^2[j] - y_i^2[j])^2 \quad i = 1,...,M \qquad (7)$$

If the error is acceptable for every pattern, the training is completed. Otherwise another training cycle is necessary (*Step* 2).

Combining several accelerating methods, the efficiency of the training program is improved.

The training algorithm has been implemented in C++ programming language for Windows '95. The user-friendly graphic interface of the program allows initializing the neural network for a specific application. So that the number of neurons at every layer, the momentum factor, the learning rate can be chosen as program options. The training patterns have been kept in ASCII files produced by simulation or acquisition programs. Also, the weights obtained during the training are saved in an ASCII file. This file will be used after training, for identification.

The neural network software can be used to train any multilayer neural network.

5. Simulation and Experimental Results

The simulation is very important to choose the appropriate topology of the neural network. Since the number of neurons in the input and output layers is determined by the external information (in this special case, 50 inputs and one output), the only possible selection is the number of the hidden neurons to reduce the required amount of calculation.

Considering that the neural network is completely trained for three patterns (according to Fig. 4), the simulation results obtained for 8, 12, 15, 18 or 24 hidden neurons are given in Table I and Fig. 8.

Table I. Simulation results.

Real J (kg·m^2)	Error (%) n_1= 8	Error (%) n_1=12	Error (%) n_1=15	Error (%) n_1=18	Error (%) n_1=24
J_0 =0.017	0	0	0	0	0
1.5J_0=0.0255	- 4.9	-5.1	- 3	- 3.73	- 2.6
2J_0=0.034	0	0	0	0	0
3J_0=0.051	3.83	8.7	- 0.8	3.79	- 7.98
4J_0=0.068	- 0.49	10.58	- 0.94	7.86	-11.74
5J_0=0.085	0	0	0	0	0

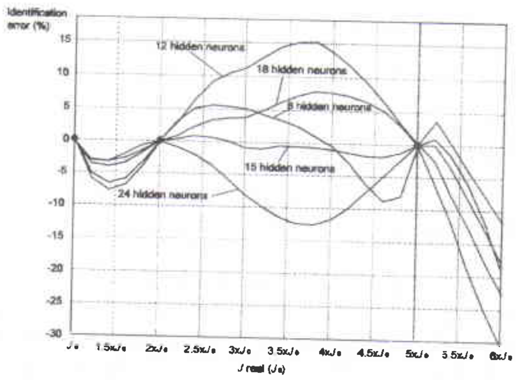

Fig. 8. Identification error for different values of total inertia moment

According to [2], [3] etc., the selection of a greater number of hidden neurons results in a better accuracy of the net with the price of the amount of calculations. But the investigations shown in Fig. 8 have revealed that selecting more or less 15 nodes in the hidden layer, the accuracy becomes worse. The reason for such behaviour is the presence of local minima in the training algorithm. They appear for 12, 18 or 24 hidden nodes and they act like atractors. Obviously, the amount of calculation increases with the number of neurons.

From Fig. 8, it can be noticed that the error becomes unsatisfactory if the inertia moment is greater than the inertia used as patterns.

Having tested the validity of the proposed identification method by simulation, the implementation is carried out. The experimental results (Table II) confirm the effectiveness of this method obtained for the topology 50-15-1.

Table II. Experimental results.

Real J (kg·m^2)	Error (%) for n_1=15
J_0 =0.017	0
1.5J_0=0.0255	- 2.81
2J_0=0.034	0
3J_0=0.051	- 0.9
4J_0=0.068	- 1.1
5J_0=0.085	0

It can be noticed that the simulation and experimental results are nearly the same because of the noise tolerance of the neural networks.

For the closed loop system (Fig. 6) and the teachers given in Fig. 7, the accuracy is worse (Table III) because the controller "hides" the information contained in the speed step responses. In spite of the closeness of the patterns, the results are satisfactory.

Table III. Identification results for closed loop system.

Real J (kg·m^2)	Error (%) for n_1=15
J_0 =0.017	0
1.5J_0=0.0255	- 1.78
2J_0=0.034	0
3J_0=0.051	6.83
4J_0=0.068	4.23
5J_0=0.085	0

6. Conclusions

The simulation results and the experiments have confirmed the validity of the proposed identification algorithm. Obviously, to obtain maximum of information for the training patterns, the speed samples should result from the transient conditions.

The implementation of the neural network based applications can be made by using PC software simulation, by dedicated analog or digital neural network chips and by DSP boards.

The parallel architecture of the two latter categories is based on multiple processing units that are interconnected to achieve high-speed computation at low-cost.

The recent rapid and revolutionary progress in microelectronics has made it possible to use neural networks in common applications. Nevertheless, PC software simulations are necessary as a first step to develop any neural network based application.

Low-cost implementation possibilities and the effectiveness of neural networks in many industrial applications ([4], [5], [7], etc.) motivate to pursue further research in this area.

References

1. L. Dafinca; R. Crăciun: "A software tool for designing of neural network based applications", *Bulletin of the Transilvania University of Braşov*, Vol. 6 (41) - 1999;

2. D. Dumitrescu, H. Costin: "Reţele neuronale. Teorie şi aplicaţii" (Neural networks. Theory and Applications), Ed. Teora, Bucureşti, 1996;

3. S. T. Welstead: "Neural Network and Fuzzy Logic Applications in C/C++", Wiley Professional Computing, New York, 1995;

4. M. T. Wishart, R. G. Harley: "Identification and Control of Induction Machines Using Neural Networks" *IEEE Trans. Industry Applications*, Vol. 31, No. 3, pp. 612-619, May/June 1995;

5. L. Dafinca: "Adaptive Control Systems Based on Neural Networks", *Lectures Notes in Computer Science -Computational Intelligence*, Vol. 1625, pp. 615 - 625, Springer-Verlag, 1999;

6. B. Kosko: "Neural Networks and Fuzzy Systems", Prentice Hall International, 1993;

7. N. Matsui: "Applications of Soft Computing to Motion Control", *Proc. of AMC '98 COIMBRA*, pp. 272-281, 1998.