NEURAL NETWORKS BASED PREDICTIVE CONTROL OF NON-LINEAR SYSTEM

Anna JADLOVSKÁ
Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, tel. 055/6024152, E-mail: Anna.Jadlovska@tuke.sk

SUMMARY

In this paper non-linear dynamical system is controlled using a predictive control strategy. This control technique needs a model of the system, which can predict the output of the system. The non-linear identification can be done using neural feed forward model as a predictor. The predictive control strategy is based on the on-line optimisation of a cost function. The on-line minimization is done using Levenberg-Marguardt algorithm by simulation language MATLAB.

Keywords: *Neural Networks, Non-linear Identification, Model-based Control, Predictive Control, Neural Control*

1. INTRODUCTION

Simulation is a commonly used technique to learn the behaviour of the system. When the mathematical model of a system cannot be derived with an analytical method then the only way is using the relationship between the input and output of a system. Fitting the model from the data is known as a identification of the system. For linear systems this technique is generally well known. Identification of non-linear systems is much more difficult.

Neural networks can be a very useful mathematical tool to build a non-linear model between the input and output of a real system, [1], [2] and [6]. In this paper neural model is used as a predictor in a model predictive control strategy. This control strategy uses the predictions of the neural network to forecast the future outputs of the system, [4] and [5].

Model based predictive control is an universal algorithm, because it can be used to control a whole range of difficult systems (non-minimum phase, non-linear,...). One of the most important advantages is the possibility to take into account the physical constraints of the real process, [3] and [7].

2. THE NEURAL MODEL

Identification of a linear system requires a model, which is written as equation (1). The linear model is configured by the three parameters *n, m, d* and the coefficients [$a_1,...,a_n,b_1,...,b_m$], where *n, m* denote the numbers of delayed outputs and inputs respectively and *d* is time delay. The way that these parameters are interconnected is called the structure of the model.

If $\hat{y}(k+1)$ is the output of the model, $u(k)$ and $y(k)$ are the input and output of the real process.

$$\begin{aligned}\hat{y}(k+1) &= a_1 y(k) + \ldots + a_n y(k-n+1) + \\ &+ b_1 u(k-d+1) + \ldots + b_m u(k-d-m+2)\end{aligned} \tag{1}$$

The feed forward serial-parallel neural model (2) has also the same three parameters *n, m,* and *d* but the relation between the input and output is more complex equation (3).

$$\hat{y}(k+1) = f(\boldsymbol{y_k}, \boldsymbol{u_{k-d+1}}) \tag{2}$$

where

$$\begin{aligned}\boldsymbol{u_{k-d+1}} &= [u(k-d+1),\ldots,u(k-d-m+2)]^T \\ \boldsymbol{y_k} &= [y(k),\ldots,y(k-n+1)]^T\end{aligned}$$

The neural model is totally defined by selecting the number of layers, the number of neurons in the hidden layers and the type of activation functions used in each layer. In this paper a neural net with only one hidden layer is used, f_2 is a pure linear function and f_1 is a sigmoid function, [1] and [5]:

$$\begin{aligned}\hat{y}(k+1) &= f_2(\boldsymbol{W_2} f_1(\boldsymbol{W_1 S_{in}}(k) + \boldsymbol{B_1}) + \boldsymbol{B_2}) = \\ &= f(\boldsymbol{S_{in}}(k), \boldsymbol{W_2}, \boldsymbol{B_2}, \boldsymbol{W_1}, \boldsymbol{B_1}) = \\ &= f(\boldsymbol{S_{in}}(k), \boldsymbol{\theta})\end{aligned} \tag{3}$$

Where $\boldsymbol{S_{in}}(k) = [\boldsymbol{u}_{k-d+1}^T, \boldsymbol{y}_k^T]^T$ is the input vector of the neural network and $\boldsymbol{\theta}$ is a vector of the weights $\boldsymbol{W_2}, \boldsymbol{B_2}, \boldsymbol{W_1}, \boldsymbol{B_1}$ and $\boldsymbol{W_1} = [\boldsymbol{W_1^u W_1^y}]$ so that $\boldsymbol{W_1 S_{in}}(k) = \boldsymbol{W_1^u u_{k-d+1}} + \boldsymbol{W_1^y y_k}$.

3. TRAINING NEURAL MODEL

After selecting the model a proper training set $\{u(k), y(k), k = 1,\ldots,N\}$ has to be selected. In this paper the training signal looks like a stair with different step weights. The neural model will be more accurate in the range of the training data; outside this training range the neural model will be less accurate. It is of great importance to know the working range of the real system. In the case of a linear system linear model is valid everywhere [1].

In this section the training of the weights $\boldsymbol{\theta}$ is discused. The calculation of the parameters $\boldsymbol{W_2}, \boldsymbol{B_2}, \boldsymbol{W_1}, \boldsymbol{B_1}$ is done by minimizing the function

$$V(\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^{N}\left(y(k+1) - f(\boldsymbol{S_{in}}(k), \boldsymbol{\theta})\right)^2 \quad (4)$$

Because the function $V(\boldsymbol{\theta})$ is complex an iterative minimization method is normally used. The well-known Back-propagation algorithm is a method, which moves the parameters $\boldsymbol{\theta}$ in the direction of the steepest gradient of $V(\boldsymbol{\theta})$. The convergence speed of this method can be improved by using an adaptive learning rate and momentum, [5].
A faster and more interesting minimization technique is the Levenberg-Marquardt algorithm [3]. When minimizing eqn. (4) using an iterative procedure, Levenberg proposes extending of eqn. (4) by adding a term (5), which is zero in the neighbourhood of a minimum.

$$\frac{1}{2}\mu\left(a_1\delta\theta_1^2 + a_2\delta\theta_2^2 + \ldots\right) \quad (5)$$

$\delta\theta_i = \theta_i(k+1) - \theta_i(k)$ is the displacement of the parameter θ_i during one iteration of the optimisation procedure. This method makes the error surface more convex in the neighbourhood of the minimum. The Levenberg-Marguardt method is based on the first order Taylor expansions of the equation (3) around an initial $\boldsymbol{\theta}$.

$$f(\boldsymbol{S_{in}}(k), \boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = f(\boldsymbol{S_{in}}(k), \boldsymbol{\theta}) + \frac{\partial f(\boldsymbol{S_{in}}(k), \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Delta\boldsymbol{\theta}\ldots \quad (6)$$

For a small $\Delta\boldsymbol{\theta}$, the second and higher order terms of the Taylor expansion can be neglected. After the substitution of the first order terms of (6) in (4), equation (7) is obtained.

$$V(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx V(\boldsymbol{\theta}) + \boldsymbol{g}^T\Delta\boldsymbol{\theta} + \frac{1}{2}\Delta\boldsymbol{\theta}^T\boldsymbol{G}\Delta\boldsymbol{\theta} + \\ + \frac{1}{2}\mu\left(a_1\delta\theta_1^2 + a_2\delta\theta_2^2 + \ldots\right) \quad (7)$$

where $\boldsymbol{g}$ is the gradient of the function $V(\boldsymbol{\theta})$ and $\boldsymbol{G} = \boldsymbol{J}^T\boldsymbol{J}$ with $\boldsymbol{J}$ the Jacobian matrix of partial derivatives $\frac{\partial f}{\partial \boldsymbol{\theta}}$. Remark that $\boldsymbol{G}$ is not Hessian matrix of $V(\boldsymbol{\theta})$ because the second derivatives of the equation (6) are neglected.
The parameters a_i are a kind of scaling where Marquardt recommended to take $a_i=1$. A set $\Delta\boldsymbol{\theta}$ is found by setting derivative $\frac{\partial V(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})}{\partial \Delta\boldsymbol{\theta}}$ equal to zero. Using eqn. (7) this derivative is approximated by eqn. (8):

$$\frac{\partial V(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})}{\partial \Delta\boldsymbol{\theta}} \approx \boldsymbol{g} + (\boldsymbol{G} + \mu\boldsymbol{S_{in}})\Delta\boldsymbol{\theta} \quad (8)$$

A set of weight changes $\Delta\boldsymbol{\theta}$ is founded by solving the equation (9)

$$(\boldsymbol{G} + \mu\boldsymbol{S_{in}})\Delta\boldsymbol{\theta} = -\boldsymbol{g} \quad (9)$$

This algorithm is implemented in the Neural Toolbox of simulation language MATLAB. A $\Delta\boldsymbol{\theta}$ is found by increasing μ by multiplying with a constant until $V(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \le V(\boldsymbol{\theta})$. It can be proved that there exist a μ so that $V(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \le V(\boldsymbol{\theta})$. The choice of the initial μ and the other parameters have to be set by the users. A good estimation of the initial value of μ is given by [3]:

$$\mu = \frac{\boldsymbol{g}^T\boldsymbol{g}}{V(\boldsymbol{\theta})} \quad (10)$$

The convergence speed depends not only on the training set but also on the initial weights.

The input weights. Because f_1 is sigmoid function, the method in this paper is rescaling $\boldsymbol{W_1}, \boldsymbol{B_1}$ by the same scale factors used to map the data into region $\langle -1,1\rangle$. We suppose that the collected data starts from a working point close to regime so $u_{reg} \approx u(0)$ and $y_{reg} \approx y(0)$. Let $\Delta u_{max} = max(abs(u - u_{reg}))$ and $\Delta y_{max} = max(abs(y - y_{reg}))$. So using eqn. (11) will map the input and output data in to the region $\langle -1,1\rangle$

$$u(k)_{scaled} = \frac{u(k) - u_{reg}}{\Delta u_{max}} \\ y(k)_{scaled} = \frac{y(k) - y_{reg}}{\Delta y_{max}} \quad (11)$$

Using eqn. (11) new initial weights $\boldsymbol{W}_1^*, \boldsymbol{B}_1^*$ can be calculated by (12) and (13):

$$\boldsymbol{W}_1^* = \left[\frac{\boldsymbol{W}_1^u}{\Delta u_{\max}} \frac{\boldsymbol{W}_1^y}{\Delta y_{\max}}\right] \quad (12)$$

$$\boldsymbol{B}_1^* = \boldsymbol{B}_1 - \frac{\boldsymbol{W}_1^u}{\Delta u_{\max}}u_{reg} - \frac{\boldsymbol{W}_1^y}{\Delta y_{\max}}y_{reg} \quad (13)$$

The output weights. Because the activation function f_2 in the output layer is a linear function the same technique as for the input weights $\boldsymbol{W}_1$, $\boldsymbol{B}_1$ can be used. Let $\boldsymbol{W}_2$, $\boldsymbol{B}_2$ are matrices with random values in the range $\langle -1,1\rangle$. The initial values $\boldsymbol{W}_2^*, \boldsymbol{B}_2^*$ can be derived by using (14)

$$\begin{aligned} \boldsymbol{W}_2^* &= \boldsymbol{W}_2 \Delta y_{max} \\ \boldsymbol{B}_2^* &= \boldsymbol{B}_2 \Delta y_{max} + y_{reg} \end{aligned} \tag{14}$$

4. VALIDATION OF THE NEURAL MODEL

After training the neural model has to be evaluated. We used two types of validation tests by [1], [5].
A correlation test is based on known correlation criterias. If the neural net succeeds in all these tests it is accepted as a good model of the non-linear system.
A what-if test is time-validation test in which the output of the neural net is compared to the output of the system for an input signal, which is different than the input signal used for training, [5].

5. MODEL BASED PREDICTIVE CONTROL

When a neural predictor is trained and validated the system can be controlled used different control algorithms. In this paper we used the Model based predictive control strategy (MBPC). MBPC is a very powerful control algorithm, which can control a whole range of difficult controllable systems (non-minimum faze, large dead-time, non-linear…). The neural parallel model (15) can be used to predict the output of the system

$$\hat{y}(k+1) = f(\hat{\boldsymbol{y}}_{\boldsymbol{k}}, \boldsymbol{u}_{\boldsymbol{k-d+1}}) \tag{15}$$

The prediction model (16) is the neural model plus a noise model

$$y^*(k+i) = f(S_{in}(k+i-1), \theta) + \frac{1}{\Delta}\xi(k+i) \tag{16}$$

The control action $u(k)$ on the moment k is derived by minimizing the eqn.(17) w.r.t. $\boldsymbol{u}$:

$$\begin{aligned} J(u) = &\sum_{i=N_1}^{N_2}\left[y^*(k+i) - r(k+i)\right]^2 + \\ &+ \sum_{i=1}^{N_u} \rho\left[\Delta u(k+i-1)\right]^2 \end{aligned} \tag{17}$$

with

$$\Delta u(k+i-1) = 0 \qquad i \geq N_u \leq N_2 \tag{18}$$

and

$$r(k+i) = \alpha r(k+i-1) + (1-\alpha) w(k) \tag{19}$$

N_2 and N_1 determine the coincidence horizon, N_u is a control horizon, ρ is the weight for control sequence and $u = [u(k), u(k+1), \ldots, u(k+N_u-1)]$. The equation (19) generates the reference trajectory and $w(k)$ is the setpoint of the control loop at time k. Minimization of $J(\boldsymbol{u})$ is done using learning the Levenberg-Marquardt algorithm. Calculation of the Jacobian $\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{u}}$ is described in [7].

5.1 Example of Neural MBPC

In this section we used SISO process to illustrate the possibilities using Neural Model Based Predictive Control (NMBPC).

$$y(k+1) = \frac{0.95y(k) + 0.25u(k) + 0.58u(k)y(k)}{1 + y(k)^2} \tag{20}$$

The structure of the neural net was set as a feed forward neural net with 10 neurons in the hidden layer, a lagged output vector *n=3* and a lagged input vector *m=2.* The activation function in the hidden layer is *logsig* function and in the output layer a *linear* function was selected.
During the training the value of V_{min} was set to 1e-5 and the maximum training epochs were set to 500. There are in Fig.1 plotted the input and output signals, which were used to train the neural net by simulation scheme in Fig.2.

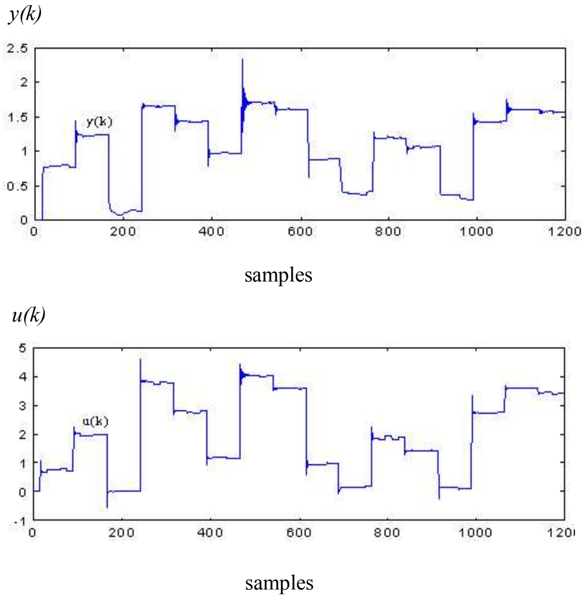

Fig. 1 The input and output training signals

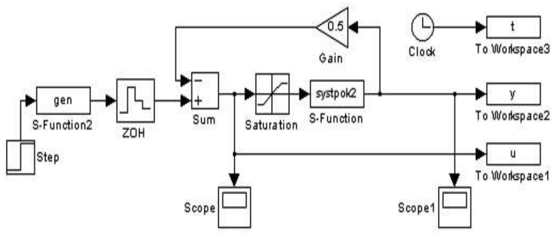

Fig. 2 The simulation scheme for collection of the training data

The neural net was trained with the Levenberg-Marquardt training algorithm. The training was stopped after 500 epochs and the value of the sum-squared error was $e_s = 0.008224$. Model is validated using time-validation test on Fig.3. It is clear that neural model is accepted by this test.

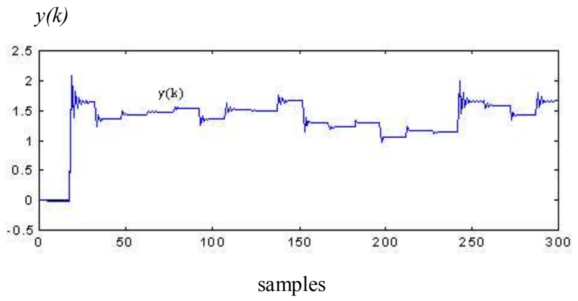

Fig. 3 Validation of the neural model

The parameters of NMBPC are set $N_2 = 5$, $\rho = 0.4$, $N_u = 2$. The discrete pole α of the reference system was set to 0.6. This example shows real power of the neural modelling and predictive control strategy (Fig.4).

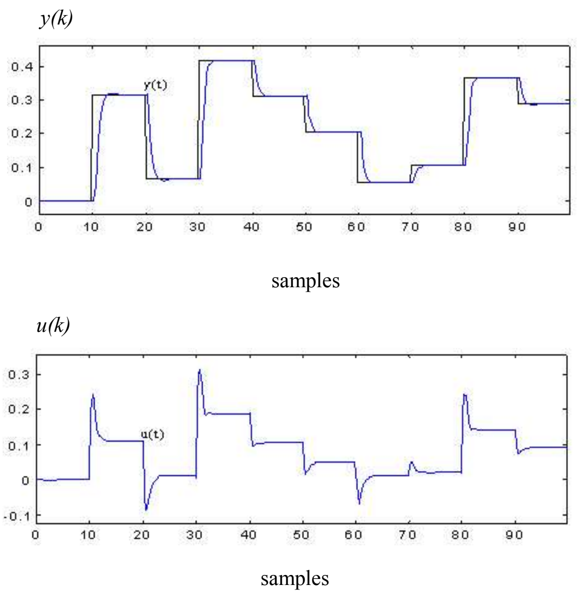

Fig. 4 The predictive control of the non-linear system

6. CONCLUSION

In this paper is given a description of a neural model based predictive control strategy for non-linear system. Simulations in language MatLab-Simulink and Neural Toolbox were illustrated by SISO version of dynamical system. The cost function of the MBPC was minimized using the Levenberg-Marquardt algorithm. This reduces the optimisation time in comparison with a gradient descent method.

REFERENCES

[1] Chen, S., Billings S., Grant P.: System Identification using Neural Networks. In International Journal of Control, Vol.51, No.6, pp.1191-1214, 1990.

[2] Hunt, K.J., Sbarbaro D., Zbikowski R.: NeuralNetworks for Control Systems–a survey, In Automatica, Vol.28, No.6, pp. 1083-1112, 1992.

[3] Hagan, M.T., Menhaj B.M.: Training Feed forwardNetworks with Marquardts Algorithm. In IEEE Transaction Neural Networks, Vol.5, No.6, pp.989 – 993, 1994.

[4] Jadlovská, A.: Using Linear Time-varying Prediction Models for Nonlinear Systems. In Proceedings the 4th International Scientific Technical Conference PROCESS CONTROL 2000, Říp, CD-ROM, 7 pages, 2000.

[5] Jadlovská, A.: Modelling and control nonlinear processes using neural networks. PhD. thesis, Department of Cybernetics and Artificial Inteligence, FEI TU, Košice, 158 pages, 2001.

[6] Ljung, L.: System Identification: Theory for the User, (T. Kailath, Ed.). Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[7] Tan, Y.: Neural Networks Based Adaptive Predictive Control. In Advances in Model Based Predictive Control (D.W. Clarke, Ed.). Oxford University Press, 1993.

BIOGRAPHY

Anna Jadlovská received MSc. degree in Technical Cybernetics from Faculty of Electrical Engineering of Technical University Košice in 1984 and Ph.D. degree in Automation and Control in 2001 at the same university. Her PhD thesis dealt with modeling and control of non-linear processes using neural networks. Since 1994 she works at the Department of Cybernetics and Artificial Inteligence, Faculty of Electrical Engineering and Informatics at TU in Košice as assistant professor. Her main research and teaching activities include problems adaptive and optimal control – especially predictive control with constrains for non-linear processes with time-variant parameters using neural networks (Inteligent control design). She is the author of articles and contributions published in journals and international conference proceedings. She is also co-author of the two monographs.