OPTIMAL SELF TUNING NEURAL NETWORK CONTROLLER DESIGN

Ladislav Körösi, Štefan Kozák

Department of Automatic Control Systems,
Faculty of Electrical Engineering and Information Technology,
SUT Ilkovičova 3, 812 19 Bratislava Slovak Republic
korosi@kasr.elf.stuba.sk, kozak@kasr.elf.stuba.sk

Abstract: The proposed paper deals with modeling and control of continuous-time processes using artificial neural network with orthogonal activation functions, applicable for real-time control. A genetic algorithm has been used to find the optimal neural structure for on-line identification with the best learning algorithm. A moving prediction horizon in the control algorithm found by genetic algorithm has been compared with a constant prediction horizon. The proposed algorithms were verified on practical control problem and have proved a good performance. *Copyright © 2005 IFAC*

Keywords: neural-network models, neural control, optimal control, on-line control, genetic algorithms

1. INTRODUCTION

Application of artificial neural networks (ANN) in modeling and control originates from an attempt to model the nervous system and its activity. Due to its universal approximation ability, a neural network can be used either as a model, or controller or another intelligent unit in the closed loop, and thus substitute conventional closed loop items. For modeling nonlinear dynamic systems, a recurrent multi-layer ANN is frequently used. In many cases, the neuron model is represented with a perceptron. Several research works and papers dealing with nonlinear process modeling using ANN show high approximation precision at the expense of computing time; therefore these techniques can't be used for real-time applications. Presented limitations (large number of iterations, long solution time, large number of training samples, etc.) can be eliminated by selecting other types of activation functions (AF), e.g. the sigmoid AF (SAF) (Oravec, et al., 1998; Šnorek, and Jiřina, 1995). Recently, orthogonal activation functions (OAF) have been used for modeling highly nonlinear processes with high precision and have been successfully applied in the design of linear and non-linear self-tuning controllers. Compared with conventional **on-line** modeling and control techniques, the SAF based techniques verified on a quantity of examples have shown a considerable modeling speed up.

The proposed paper focuses on modeling and control of continuous-time processes using artificial neural networks with OAF using genetic algorithm to optimize the ANN structure, cost function weight parameters and moving prediction horizon.

2. NEURAL NETWORK STRUCTURE USING ORTHOGONAL ACTIVATION FUNCTIONS

A typical neural structure used to approximate a non-linear function is the three-layer structure in Fig.1 (Zhu, et al., 1998). The input layer has m nodes with inputs $u=[u(1)\ u(2),...u(m)]$. The hidden layer consists of neurons with orthogonal (orthonormal) activation functions. It is assumed that the AF's for these neurons belong to the same class of orthogonal functions and no two neurons have the same order of AF in the input blocs. Each neuron has a different order activation function beginning from *0* to *n*. This ensures that no cross-correlation occurs among the

neurons in the hidden layer. The nodes on the right side of the orthogonal neurons implement the product operation. Each π node has m input signals from m different input blocks**.** The weights between the input and the hidden layers are fixed and depend on the OAF type. The input and output layers consist of linear neurons. The output of the network with OAF (estimated process output) is given by the linear combination of activation functions

$$y_m(u,w)=\sum_{n_1=0}^{N_1-1}\cdots\sum_{n_m=0}^{N_m-1}w_{n_1}\ldots n_m\phi_{n_1}\ldots n_m(u)=\Phi^T(u)W \quad (1)$$

where $u=[u_1\ u_2\ ..\ .u_m]^T$ is a *m*-dimensional input vector, N_i is the number of neurons associated with the *i-th* input, W is the vector of weights between hidden and output layers and Φ are the OAFs.

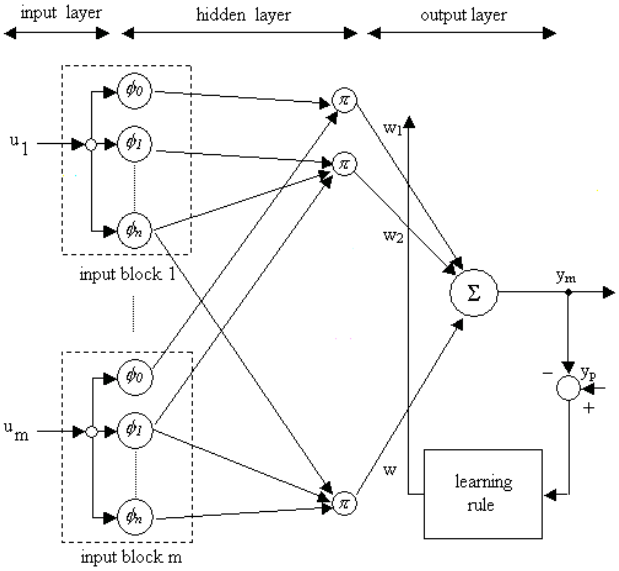

Fig. 1. Block scheme of a three-layer ANN with OAF.

The learning in ANN is performed by adapting the network weights so that the expected value of the mean squared error (MSE) between the network output and the training output is minimized. The recursive MSE learning algorithm is used to train the OAF based ANN. On-line recursive MSE algorithm can be described with equations:

$$\varepsilon\left(k,\hat{W}(k-1)\right)=y_p(k)-\phi^T(k)\hat{W}(k-1) \quad (2)$$

where $y_p(k)$ is the process output in *k-th* step, $\hat{W}(k-1)$ is the weight matrix in step *k-1*, $\phi^T(k)$ is the transformed input signal and ε is the error between process output and the ANN output.

$$K(k)=\frac{P(k-1)\phi(k)}{1+\phi^T(k)P(k-1)\phi(k)} \quad (3)$$

where P is the covariance matrix and *K* is the gain vector.

$$P(k)=P(k-1)-K(k)\phi^T(k)P(k-1) \quad (4)$$

$$\hat{W}(k)=\hat{W}(k-1)+K(k)\varepsilon\left(k,\hat{W}(k-1)\right) \quad (5)$$

There are some algorithm modification as constant trace, exponential forgetting, exponential forgetting and resetting algorithm.

The on-line methods are described in more detail in (Körösi, 2002; Linder, 2002; Kozák, 2002; Kozák 2003).

3. CONTROLLER DESIGN

The control of nonlinear processes using the ANN is extraordinarily interesting from a practical point of view, leading to many new studies and results on control problems such as robust stability and performance improvement. As ANN modeling adapts to parametric and structural variations of the system, artificial ANN with OAF provides an efficient means for robustness maximization when modeling non-linear systems.

A nonlinear self-tuning predictive controller (Fig.2) consists from two blocks (Kozák, 2002; Kozák 2003). Its first block is the feed-forward OAF based ANN; the ANN is trained on-line. The output from the network is the predicted system output. The second (optimization) block computes the system input, which ensures equality between system and network outputs.

The control variable can be determined as follows

$$u(k)=u(k-1)-\alpha\frac{\partial J(k)}{\partial u(k)} \quad (6)$$

where *J(k)* is the cost function in the form and α is the weighting factor

$$J(w,e)=\frac{1}{2}\left\{[r(k+1)-y_m(k+1)]^2+\gamma[\Delta u(k)]^2\right\} \quad (7)$$

where *r* is a reference signal, y_m is the ANN model output, Δu is the control signal and γ is the weighting factor.

Applying the Quasi-Newton method, the control variable is

$$u(k)=u(k-1)-\left(\frac{\partial^2 J}{\partial u^2(k-1)}\right)^{-1}\cdot\frac{\partial J}{\partial u(k-1)} \quad (8)$$

The robust version of the proposed control algorithm can be expressed in the form

$$u(k)=u(k-1)-\left(\max\left(abs\left(\frac{\partial^2 J}{\partial u^2(k-1)}\right),h\right)\right)^{-1}\frac{\partial J}{\partial u(k-1)} \quad (9)$$

where $h>0$ is a small constant.

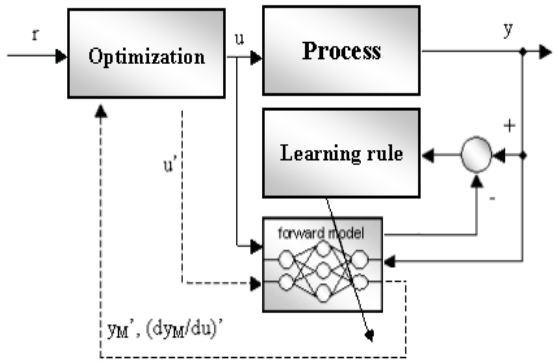

Fig. 2. Block scheme of a nonlinear self-tuning predictive controller.

The control algorithm contains these basic four steps:

1. initialization (process input-output sampling and normalization, *u(k)=u(k-1)*)
2. prediction of y_M and computation of derivations
3. computation the optimal control u (recursive algorithm using equation 9)
4. denormalization

The above algorithm is described in more detail in (Körösi, 2002; Linder, 2002; Kozák, 2002; Kozák 2003).

4. GENETIC ALGORITHMS

Using genetic algorithms is an optimization technique that relies on parallels with nature. It can tackle a variety of optimization techniques provided that they can be parameterized in such a way that a solution to the problem provides measure of how accurate the solution found by the algorithm is.
Genetic algorithms have been used in conjunction with neural networks in three major ways. First, they have been used to set the weights in fixed architectures. Second, genetic algorithms have been used to learn neural network topologies (how many hidden units a NN should have, how the nodes are connected, etc.). A third major application is the use of genetic algorithms to select training data and to interpret the output behaviour of a neural network. The second way is described in more detail this section.
Usually, there are only two main components of most genetic algorithms that are problem-dependent (Darrell, 1994): the problem encoding and the evaluation function (fitness).
Using genetic algorithms for solving optimization problems can be briefly described as follows. A population of possible solutions to an optimization problem is obtained in the form of vectors, the so-called chromosomes. Each vector consists of genes taken from the range given by its lower and upper limits. The OAF base NN structure with the learning algorithm can be coded by a chromosome, which consists of the following genes:

Gene 1 : Number of delayed process inputs
Gene 2 : Number of delayed process outputs
Gene 3 : OAF order (one dimensional OAF)[1]
Gene 4 : OAF type
Gene 5 : Learning method
Gene 6 : Learning parameter

After calculating the fitness[2] function (the fitness of an organism is measured by the success of the organism to survive) for individual vectors a new population is created.
The new population comprises:

1. the best string or strings (with the smallest fitness - minimization) to ensure the convergence
2. other selected strings (for example: roulette wheel selection, rank selection, steady-state selection, etc.)
3. crossover (to form new offspring from parents) and mutation (mutates offspring at each locus with the mutation probability) operations can be applied to the chosen strings
4. addition of new random strings

It is expected that the objective criterion values for different chromosomes will gradually improve over generations, approaching the optimal values. This ensures the convergence of the algorithm. Constructing the optimal neural network structure with genetic algorithm is an off-line method.

5. SIMULATION EXAMPLES

For testing the ANN modeling and control, the bioprocess (bioreactor) presented in (Körösi, 2002) has been considered (Fig.3).

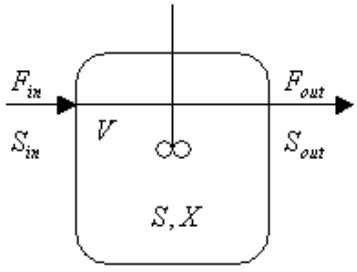

Fig. 3. Principal scheme of the bioreactor.

The process is described by the following equations:

$$\frac{dX}{dt} = \mu(S)X - DX \tag{10}$$

$$\frac{dS}{dt} = -k_1\mu(S)X - k_2 v(S)X - DS + DS_{in} \tag{11}$$

$$\frac{dP}{dt} = v(S)X - DP \tag{12}$$

---

[1]The π - neurons order is set to OAF order to ensure a faster adaptation due to a low number of interconnections between input and hidden layer.
[2] The fitness function must be relatively fast.

The matrix representation:

$$\frac{d}{dt}\begin{bmatrix} X \\ S \\ P \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -k_1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} SX & 0 \\ 0 & SX \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} - D\begin{bmatrix} X \\ S \\ P \end{bmatrix} + \begin{bmatrix} 0 \\ DS_{in} \\ 0 \end{bmatrix} \quad (13)$$

where X – cell mass concentration [g/l], P – product concentration [g/l], S – substrate concentration [g/l] with the following specified parameters:

$k_1 = 27.3$, $k_2 = 0$, $D = 0.3$, $S_{in} = 5$, $X(0) = 0.3$, $S(0) = 0.1$, $P(0) = 0.1$, $\alpha = 0.2$, $\beta = 0.1$.

5.1 Neural network modelling

Process identification can be defined as finding such input-output relations and parameters, which describe the process behavior with a specified precision. For nonlinear process modeling and control, the black-box approach is used, that means observing the input-output data without any knowledge about the structure, order, mathematical description, etc.

To generate the optimal ANN structure a genetic algorithm has been used because finding the optimal ANN structure (including learning method, etc.) is an optimization problem with several parameters. The population consists of 100 chromosomes. The encoding is show in the previous section with the following gene intervals:

Delayed process inputs	<1, 6>
Delayed process outputs	<1, 6>
OAF order	<0, 5>
OAF type	<0, 3>, where
0 = Hermit, 1 = Laguerre, 2 = Legendre,	
3 = Chebychev	
Learning method	<0, 2>, where
0 = constant trace, 1 = exponential forgetting,	
2 = exponential forgetting and resetting algorithm	
Learning parameter	<0.05, 1>

The algorithm uses combination of the mutation, crossover and selection operations. After 2000 simulations (20 populations) the sum squared error (SSE – fitness function) settled at 398.25 for training data and 147.36 for testing data with the following string:

1. Number of delayed inputs : 4
2. Number of delayed outputs : 4
3. OAF order : 2
4. OAF type: Hermit
5. Learning method : exponential forgetting
6. Learning parameter : 0.98 (constant exp. forgetting factor)

For simulation, 2000 training and 2000 testing samples have been used for the sampling time *T=0.1h*. The OAF based ANN has been compared with the perceptron with 8 input neurons and 24 sigmoid hidden neurons (structure similar to ANN with OAF). The SSE is smaller (341.39 for training data and 131.20 for testing data) but the learning took 2000 epochs for on-line weights adaptation. Graphical simulation results are shown in Figures 4 and 5.

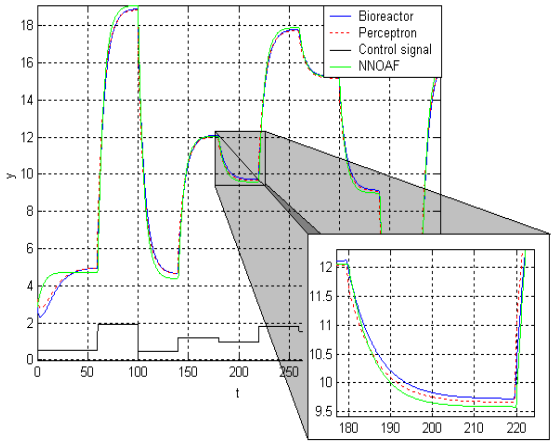

Fig. 4. Time responses of the process and the neural model outputs with Hermit OAF and SAF (training data).

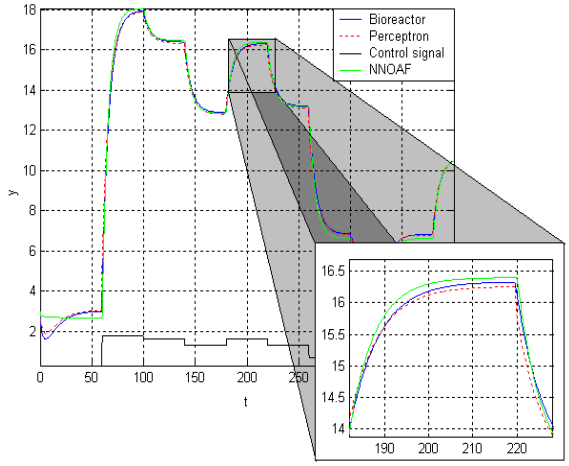

Fig.5 Time responses of the process and the neural model outputs with Hermit OAF and SAF (testing data).

5.2 Neural network control

For process control using the adaptive ANN model, the previous optimal structure has been used. Because the control quality depends on the prediction horizon and penalization factors the maximal prediction horizon, 32, has been found by increasing the prediction horizon from 0 and observing the control quality. Increasing the prediction horizon over 32 gives worse results per consequens of computation error.

The next step is choosing the penalization factors α and γ. A genetic algorithm has been used to find these parameters for the prediction horizon 15 (average). The population consists of 100 strings. Each string contains 2 genes from intervals:

1. α <0.05, 2>
2. γ <0.05, 2>

In the algorithm, the mutation, crossover and selection operations have been combined. After 1000 simulations (10 populations) the sum squared error (SSE – fitness function) settled with the penalization factors $\alpha = 0.95$ and $\gamma = 2$. The simulation results for different prediction horizons are in Fig. 6.

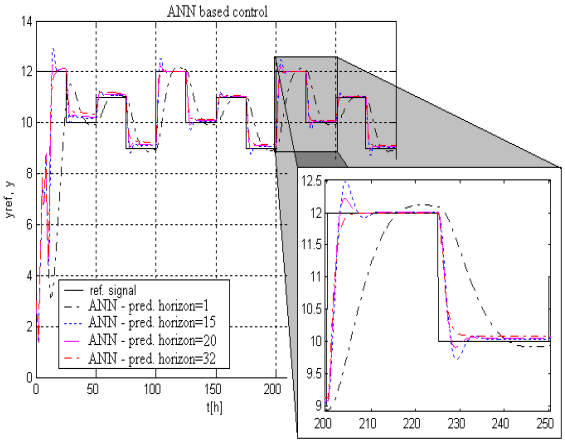

Fig. 6. ANN based control using Hermit OAF for prediction horizons 1, 15, 20 and 32.

A rising prediction horizon speeds up the control process at the beginning (with an overshoot) and then slows it down. With a smaller prediction horizon the control is very slow. There are several possibilities how to improve the control, e.g. using an off-line trained ANN, leaving more time to adapting the ANN to the reference variable, using a moving prediction horizon, etc. In our case a moving prediction horizon has been applied.

To construct the optimal prediction horizon path, a genetic algorithm is used. The population consists of 70 chromosomes. Each chromosome contains 10 genes from the interval <1, 32> which represents the following points: $<ts_i; gene_i>$, where $ts=0,1,2,\ldots$ is the time elapsed after the reference change; $gene_i$ is the prediction horizon and $i=1,..,10$. The neighboring points are interpolated with lines (Fig. 7.)

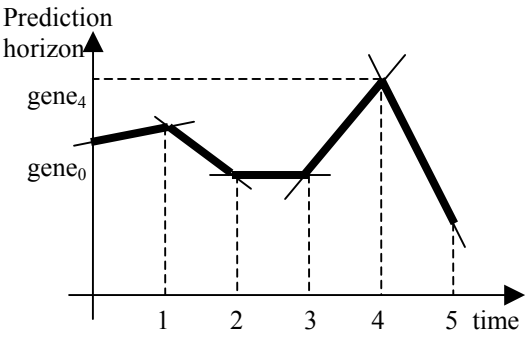

Fig. 7. Example of construction of the optimal prediction horizon path using genetic algorithm.

In the proposed algorithm, the mutation, special mutation, crossover and selection operations have been combined. Special mutation means mutation at every gene, one mutated gene per chromosome; therefore the offspring for this operation will contain 10 new mutated chromosomes.

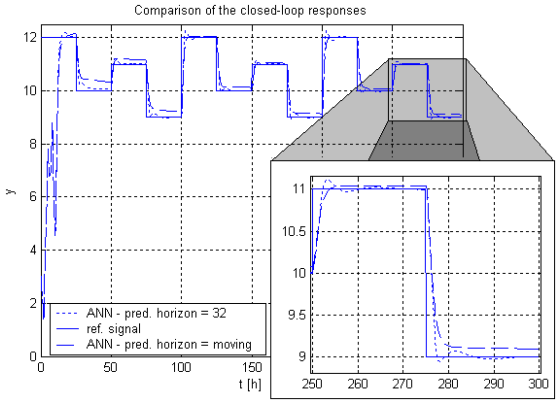

Fig. 8. Comparison of the closed-loop responses using the ANN with Hermit OAF for the prediction horizon 32, and the moving prediction horizon.

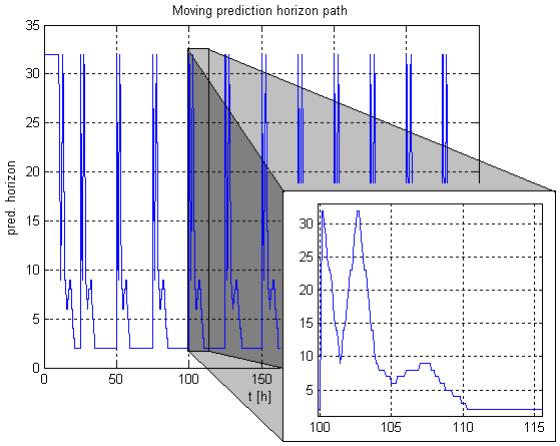

Fig. 9. Specification of the time path of the moving prediction horizon.

The optimal path after genetic algorithm evaluation is coded with the following chromosome: [31.7540, 8.6002, 31.7836, 8.3830, 5.6929, 7.7739, 8.2488, 4.4140, 2.5917, 6.9444]. Using the moving prediction horizon (Fig.9) brought about performance improvement in terms of SSE and settling time (Fig.8). This control algorithm is less computationally demanding because of a smaller prediction horizon.

6. CONCLUSION

The presented paper deals with design and improving the conventional ANN modeling and control methods using orthogonal activation functions (OAF). The proposed approach to modeling and control has been demonstrated for a real plant model of a bioreactor. Simulation results confirm the advantages of the designed solution in terms of precision, speed and quality. Theoretical, numerical and graphical results prove the effectiveness of this approach applied in modeling and control of highly nonlinear practical problems.

ACKNOWLEDGMENT

This paper was partially supported by the Slovak Scientific Grant Agency VEGA, Grant No. 1/0115/03.

REFERENCES

Darrell W. (1994), A genetic algorithm tutorial. *Statistics and Computing*, 4:65-85

Kozák, Š. and L. Körösi (2002), Využitie umelých neurónových sietí v praxi a v priemysle, Conf. SSKI, Trebišov

Kozák, Š. and L. Körösi (2003), A novel type of self-tuning neural controller, IFAC Conference CAO, Visegrád, Hungary

Körösi, L. (2002), Metódy identifikácie a riadenia nelineárnych procesov pomocou ortogonálnych a wavelet funkcií, FEI STU Bratislava, KASR p. 1-70

Linder, P. (2002), Modelovanie a riadenie dynamických systémov pomocou neurónových sietí s ortogonálnymi aktivačnými funkciami. FEI STU Bratislava, KASR, p. 1-55

Liu, G. P. (2002), Neural-learning control of nonlinear systems using variable neural networks, IFAC, 15th Triennial World Congress, Barcelona, Spain,

Oravec, M., J. Polec and S. Marchevský (1998), Neurónové siete pre číslicové spracovanie signálov. Vydavateľstvo FABER, Bratislava, 196p.

Šnorek, M. and M. Jiřina (1995), Neuronové sítě a neuropočítače. Vydavateľstvo ČVUT, Praha, 124P.

Zhu, Ch., D. Shukla and F.W. Paul (1998), Orthogonal Function for System Identification and Control, Control and Dynamic System, Edited by Cornelius T. Leondes , Academic Press, p.1-72