NONLINEAR SYSTEM IDENTIFICATION BASED ON EVOLUTIONARY DYNAMIC NEURAL NETWORKS WITH COMPLEX WEIGHTS

L. Ferariu

"Gh. Asachi" Technical University of Iaşi
Dept. of Automatic Control and Industrial Informatics
RO-6600 Iaşi, Bd. D. Mangeron 53A
ROMANIA
Phone, fax: +40-232-230751
E-mail: lferaru@ac.tuiasi.ro

Keywords: system identification, neural networks, multiobjective optimisation, genetic algorithms.

Abstract

The paper presents a novel dynamic neural architecture that allows a flexible and compact representation of nonlinear processes. The suggested neural topology is obtained by providing local internal recurrence for the static neural network with complex weights. An evolutionary multiobjective design procedure assists the automatic selection of appropriate neural topologies and parameters. It searches for accurate neural models, characterised by good generalisation capabilities. The experiments reveal that the presented approach is suitable for system identification.

1 Introduction

The identification of dynamic nonlinear systems can be addressed by means of neural networks. These structures have become a powerful tool [4, 5], because they are able to generate complex mapping between the input and the output space, by learning the general trend of the target values.

The multitude of computational models illustrates that none architecture can be uniformly better than the others. Although, the investigation of novel neural topologies is still necessary. These new alternatives could provide better results for a specific class of problems. Recently, the computational efficiency of the **N**eural **N**etworks with **C**omplex **W**eights (CWNNs) was demonstrated, both on mathematical examples and applications [4]. If these neural structures are compared with the nets characterised by real weights, some advantages can be revealed: the decreasing of the computational time and better approximation capabilities for certain applications.

The present paper reconfigures the static topology of CWNNs, by including local internal recurrence. The resulted neural architecture can efficiently cope with system identification. The necessary information about the past values of the plant variables is provided by the states of the internal dynamic structures. Evolutionary algorithms are used for selecting appropriate dynamic neural networks with complex weights. The design procedure automatically searches for appropriate neural topologies and parameters, via a multiobjective optimisation.

The paper is organized as follows. Section 2 describes the structure and the characteristics of the static neural networks with complex weights. The dynamic neural networks with complex weights are introduced in section 3. Then, in section 4, a description of the evolutionary design algorithm is given. The applicability of the approach is investigated in section 5, with respect to the identification of an industrial system. Finally, in section 6, several conclusions are presented.

2 Static neural networks with complex weights

The considered CWNN topology [4] has m inputs, one hidden layer with n hidden neurons and one output linear neuron (Fig. 1). Assuming that the hidden neurons are characterised by the activation function $g:\Re \to \Re$ and the input operator φ, the mathematical model of the net can be expressed as indicated in equation (1):

$$f:\Re^m \to \Re\,,\; f(\mathbf{u}) = \sum_{i=1}^{n} a_i \cdot g(\varphi(\mathbf{w}_i,\mathbf{u},\mathbf{c}_i))\,, \qquad (1)$$

where $\mathbf{u} = [u_k]_{k=1,..,m}$ denotes the neural input, $a_i \in \Re$, $i = 1,..n$ indicate the real weights of the output neuron, $\mathbf{c}_i \in \Re^m$, $\mathbf{c}_i = [c_{ik}]_{k=1,..,m}$ and $\mathbf{w}_i \in C^m$, $\mathbf{w}_i = [w_{ik}]_{k=1,.,m}$, $i = 1,..n$ represent the centres and the complex weights associated to the i^{th} hidden neuron, respectively.

The input operator is computed as follows:

$$\varphi(\mathbf{w}_i,\mathbf{u},\mathbf{c}_i) = \langle \mathbf{w}_i, \mathbf{u}-\mathbf{c}_i \rangle \cdot \langle \overline{\mathbf{w}}_i, \mathbf{u}-\mathbf{c}_i \rangle\,. \qquad (2)$$

Here, $\overline{\mathbf{w}}_i$ denotes the conjugate of $\mathbf{w}_i$, for $i = 1,...,n$, and $\langle \mathbf{w}_i, \mathbf{u}-\mathbf{c}_i \rangle$ represents the inner product of vectors $\mathbf{w}_i$ and $\mathbf{u}-\mathbf{c}_i$.

If $|w_{i1}| = ... = |w_{im}| = w_i$, meaning that $w_{ik} = w_i e^{j\theta_{ik}}$, with $\theta_{ik} \in [0, \pi - \Delta] \bigcup [\pi + \Delta, 2\pi]$, $\Delta > 0$, $i = 1,.,n$, $k = 1,.,m$, a simpler form of the input operator can be obtained, using Euler's formula:

$$\varphi(w_i, \mathbf{u}, \mathbf{c}_i) = w_i \cdot \sqrt{\breve{\varphi}(\mathbf{u}, \mathbf{c}_i)}\ , \tag{3}$$

$$\breve{\varphi}(\mathbf{u}, \mathbf{c}_i) = \left[\sum_{k=1}^{m} \cos\theta_{ik} \cdot (u_k - c_{ik}) \right]^2 + \left[\sum_{k=1}^{m} \sin\theta_{ik} \cdot (u_k - c_{ik}) \right]^2 . \tag{4}$$

The present approach considers the activation function:

$$g : \Re \to \Re, g(x) = \exp(-x^2)\ . \tag{5}$$

Usually $\Delta = 0.01$.

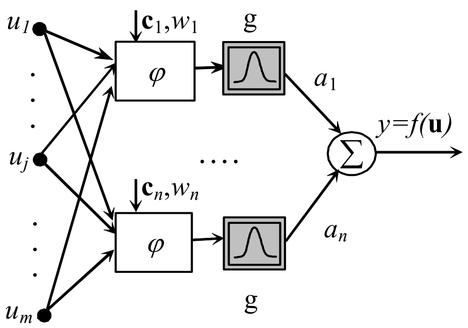

Fig. 1. The static topology of the neural networks with complex weights (*m* inputs, *n* hidden neurons, one output)

It has been proven that the neural networks with complex weights represent universal approximators of static continuous nonlinear functions [4].

In order to model dynamic nonlinearities, identification schemes based on time delay networks with complex weights can be considered. The most suitable structure for the net is the input-output format. The case of **M**ulti **I**nputs **M**ulti **O**utputs (MIMO) systems is illustrated. For each output of the process, a different neural approximator is designed. A set of external blocks implements the lagged values of the process inputs and outputs, such that the static neural network could approximate one output of the process:

$$\hat{y}_i(k) = f(\mathbf{u}_p(k-d), \mathbf{u}_p(k-d-1), .. \mathbf{u}_p(k-n_u), \mathbf{y}_p(k-1), ... \mathbf{y}_p(k-n_y)). \tag{6}$$

Here, k denotes the current sample time, n_u and n_y indicate the maximum time delays considered for the process inputs ($\mathbf{u}_p$) and the process outputs ($\mathbf{y}_p$), respectively, y_i denotes the i^{th} output of the process and $\hat{y}_i$ represents the approximation given by the neural network for y_i.

Usually, the design procedures consider that the number of time delays and the dead time of the process are tuned according to trial and error method. The dimension of the neural input increases, depending on the number of requested lagged measurements [5]. The topology indicated in Fig. 1 is characterized by $Ps = n \cdot (m+2) + 1$ parameters.

3 Dynamic neural networks with complex weights

The architecture of static CWNN is extended by including **A**uto-**R**egressive **M**oving **A**verage (ARMA) filters. The novel neural architecture is named **D**ynamic **N**eural **N**etwork with **C**omplex **W**eights (DCWNN).

The ARMA filters placed on the input connections of a neuron implement the local synaptic feedback [5], the ARMA filters placed before the activation functions of the neurons implement the local activation feedback and the ARMA filters placed on the recurrent connections provided from the output of a neuron to the input of its activation function implement the output feedback. The internal dynamic blocks are denoted as follows (Fig. 2): FH_{ik}^s, i=1,…,n, k=1,…,r represents the synaptic hidden filter corresponding to the connection considered from the k^{th} input to the i^{th} hidden neuron; FH_i^a and FH_i^o denote the activation filter and the output filter of the i^{th} hidden neuron, respectively; FO_i^s specifies the synaptic filter corresponding to the connection considered from the i^{th} hidden neuron to the output neuron; FO^a and FO^o denote the activation filter and the output filter of the output neuron, respectively.

These filters are characterised by the discrete transfer function:

$$G(z^{-1}) = w \frac{1 + b_0 z^{-1}}{1 + a_0 z^{-1} + a_1 z^{-2}} = \frac{P(z^{-1})}{Q(z^{-1})}\ . \tag{7}$$

Here, $P(z^{-1})$, $Q(z^{-1})$, respectively, denote the numerator and the denominator of the discrete transfer function and w, b_0, a_1, a_0 indicate their coefficients. Using equation (7), two particular cases are simple to be illustrated: connection characterised by a simple weight ($a_0 = a_1 = b_0 = 0$, $G(z^{-1}) = w$) and connection eliminated from the neural architecture ($w = 0$). This allows a convenient encoding of the dynamic neural topologies.

The DCWNNs can provide a better approximation of the dynamic non-linearities, if the design procedure allows a flexible configuration of its internal dynamic blocks. The ARMA filters can reduce the level of noise that affect the neural inputs and memorize the necessary past states of the neural network. For example, FO^o acts as a memory of the network output, FH_{ik}^s, i=1,…,n, k=1,…,m memorize the past values of the neural inputs, etc. As consequence, an important reduction of input space dimensionality can be obtained, because the identification schemes based on DCWNNs do not need supplementary external dynamic elements. The neural input vector, denoted with $\mathbf{u}(k)$, is obtained as follows:

$$\mathbf{u} \in \Re^r,\ \mathbf{u}(k) = \begin{bmatrix} \mathbf{u}_p(k) \\ \mathbf{y}_p(k-1) \end{bmatrix}, \tag{8}$$

where $\mathbf{u}_p(k)$ and $\mathbf{y}_p(k)$ represent the inputs and the outputs of the process at k^{th} time sample, respectively.

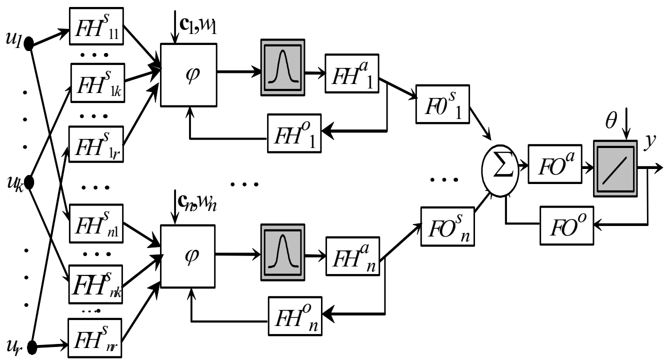

Fig. 2. The topology of DCWNNs (r inputs, n hidden neurons, one output). Here, θ specifies the bias of the linear output neuron and $\mathbf{c}_i, w_i$ indicate the centres and the complex weight of the i^{th} hidden neurons, i=1,…,n.

The structure of the DCWNN is defined by the number of hidden neurons, the type and the complexity of the ARMA filters that were included into the topology and the configuration of the connections existing between the input layer and the hidden layer. The neural parameters are represented by the coefficients of the ARMA filters, the bias of the linear output neuron, the centres and the complex weights of the hidden neurons. The DCWNN can have maximum $Pd = (5r+13)n+9$ parameters.

The method described in the following uses evolutionary techniques in order to search for the optimal DCWNN topologies and parameters. The training procedure does not require the derivatives of the objective functions. Also, a flexible configuration of the neural architecture is performed. One considers neural topologies for which the input layer is not fully connected with the hidden neurons and not all permitted dynamic structures are compulsory. As consequence, no a priori knowledge about process dynamic orders and process dead time is required.

Taking into account the maximum time-delays implemented according to the dynamic architecture indicated in Fig. 2, the performances of DCWNNs can be compared with the static topologies having $m = 8(r-1)+12$ inputs. For $n \geq 2$, it results that $Ps > Pd$, if $r \geq 4$. Moreover, the proposed design procedure guarantees a significant reduction of the total number of parameters, encouraging the selection of simple topologies.

4 Evolutionary design of dynamic neural networks with complex weights

At each iteration, the evolutionary algorithm acts on a population of *Nind* possible solutions, named individuals or chromosomes. The initial population is randomly generated from the space of permitted neural models. The centres of the hidden neurons are initialized according to the fuzzy C – means clustering algorithm. During the evolutionary loop, the values of these parameters are modified by means of genetic techniques.

4. 1 Encoding

Each individual included in the population encodes the architecture and the parameters of a DCWNN. The chromosome is organized in a three–level structure, containing control and parametric genes, as described in Fig. 3. The highest priority level, named level 1, specifies which hidden neurons are included in the neural topology. The second priority level indicates the active dynamic structures and the orders of the corresponding numerators and denominators. All neural parameters are encoded, as float numbers, in the lowest priority level, denoted with level 3.

Level 1

the i^{th} hidden neuron, i = 1,…,n		

0 – hidden neuron deactivated;
1 – hidden neuron activated;

Level 2

	FH^s_{i1}	…	FH^s_{im}	FH^a_i	FH^o_i	
	the dynamic blocks for the i^{th} hidden neuron					

FO^s_1	…	FO^s_n	FO^a	FO^o
the dynamics blocks for the output neuron				

According to eq. (7):
0 – connection does not exist ($w = 0$)
(only for filters FH^s, FH^o and FO^o);
1 – simple weight: $G(z^{-1}) = w$;
2 – filter described by $G(z^{-1}) = w/(1+a_0 z^{-1})$;
3 – filter described by $G(z^{-1}) = w/(1+a_0 z^{-1}+a_1 z^{-2})$;
4 – filter described by $G(z^{-1}) = w(1+b_0 z^{-1})/(1+a_0 z^{-1}+a_1 z^{-2})$.

Level 3

..	$\theta_{i1}\ldots\theta_{im},\ c_{i1}\ldots c_{im},\ w_i$	…	w	b_0	a_1	a_0	..	θ
the parameters of the neural model								

Fig. 3. The hierarchical encoding of the DCWNNs

The control genes included in a higher level can activate (when control gene's value is nonzero) or deactivate (when control gene's value is „0") the corresponding control or parametric genes contained in a lower level. The inactive genes are preserved in the chromosome structure [6], being considered as initial values in the next activations. The hierarchical encoding assures an efficient exploration of the search space, because even small variations of the control genes can produce great changes in the structure of the encoded DCWNN. The approach allows the integer encoding of control levels, in order to improve the exploration capabilities of the algorithm, while maintaining a reduced length of the chromosome.

4. 2 Genetic operators

Offspring are generated using discrete recombination (for the control levels 1 and 2), intermediary crossover (for the parametric level 3) and uniform mutation [1]. These operators are able to reduce the negative effect of "competing conventions" and allow the efficient exploration of the search space**.**

A correct architecture satisfies the following requirements: the hidden layer includes at least one hidden neuron; each input is connected to at least one hidden neuron; each hidden neuron is connected to the output neuron and with at least one input of the network. If an offspring encodes an incorrect topology, remedy actions are applied, meaning that supplementary connections and/or neurons are activated. The new structural elements are randomly selected from the set of available alternatives.

4. 3 Multiobjective optimisation

The selection of appropriate DCWNNs is done by means of a multiobjective optimisation. Six objective functions are considered. They are organized, according to the assigned priority, on a two-level hierarchy [2]. The objective function f_1, namely the sum of the output squared errors computed for the normalised training data set, represents a measure of the neural model accuracy and is assigned with the highest priority. The values of this objective function are computed after applying the supervised training procedure presented below. The other objective functions describe the complexity order of the encoded neural architecture and have the same low-level priority. Taking into account the influence of each structural element on the functionality of the neural network and the resulted total number of neural parameters, five different directions are separated: f_2 - the number of active hidden neurons; f_3 - the number of active connections existing between the network's inputs and the active hidden neurons; f_4 - the number of active output filters; f_5 - the sum of numerators and denominators' orders, corresponding to all active output filters; f_6 - the sum of numerators and denominators' orders, corresponding to all active synaptic and activation filters. These objectives force the selection of simple neural models, with expected good generalisation capabilities.

For each DCWNN encoded into the population, a supervised training procedure adapts the set of neural parameters, with respect to the minimisation of the highest priority objective function. The training algorithm is implemented as follows. At the first stage, a standard genetic search is performed, working, for a predefined number of generations (*Ngen_Tr*), on a population of *Nind_Tr_1* individuals. The best set of parameters achieved at the end of this evolutionary loop, denoted as $\mathcal{PR}$, is passed to the second stage of the training procedure. Here, a fine search is considered. The set $\mathcal{PR}$ competes with *Nind_Tr_2* offspring (*Nind_Tr_2* < *Nind_Tr_1*). The offspring are generated considering small variations in the genetic material of $\mathcal{PR}$. The training algorithm does not require the derivatives of the objective function f_1.

Pareto-optimisation. The search procedure is combined with a decision mechanism, according to a progressive articulation of preferences [3, 9]. A goal is associated to each objective function. The goals define the desired area for the objective values. The individuals placed beyond the specified area are not encouraged to produce offspring and to survive. During the evolutionary loop, the goals are adapted according to the mean performances of the current population [7]. The Pareto-optimisation method is based on a ranking selection. The ranks of the individuals are computed according to the following rules. If an individual satisfies all imposed goals, its rank is assigned based on the values of the highest priority objective function, f_1. Otherwise, the rank is specified taking into account the degree of goals' violations and the priority of the unsatisfied goals. These rules encourage the algorithm to produce accurate models, characterised by a simple architectures [2].

4. 4 Insertion and migration

The insertion is solved by means of the *Pareto reservation strategy* [10]. S_{off} offspring and $Nind - S_{off}$ individuals contained in the current population are selected to survive in the next generation. The method encourages the survival of non–dominated chromosomes.

Migration strategy. The population is separated into two partially isolated subpopulations [2]. The genetic material of the main subpopulation is improved according to the multiobjective optimisation previously formulated. It guarantees the achievement of accurate models, characterised by simple topologies. For the auxiliary subpopulation, a mono-objective optimisation is considered, demanding only the minimisation of the highest priority objective function, f_1. The auxiliary subpopulation improves the accuracy of the neural models. Once at *No_migr* generations, an exchange of information is permitted between the two subpopulations. During migration, the genetic material of the main subpopulation is enriched with accurate models and, as consequence, the highest priority goal is considerable reduced. Also, chromosomes that encode simple neural structures are introduced into the auxiliary subpopulation; their approximation capability will be improved till the next migration stage. At the end of the evolutionary loop, the best neural model included in the main subpopulation is used. The algorithm encourages the survival and the duplication of accurate models, while maintaining an adequate complexity order of the encoded topology. The selection pressure imposed by the highest priority objective is adapted according to the performances of the current population.

4. 5 The stages of the evolutionary algorithm

A schematic description is presented in the following:

1. Create an initial hierarchical population containing *Nind* individuals.
2. Check the correctness of the encoded topologies (with remedy actions if necessary) and compute actual values of goals.
3. Train the neural networks encoded into the population.
4. Evaluate the chromosomes according to all considered objectives and compute the fitness values.
5. Loop over a number of *Max_gen* generations:
   5.1 For each subpopulation:
      5.1.1. Select parents for the reproduction pool.
      5.1.2. Apply crossover and mutation operators.
      5.1.3. Check the consistency of offspring (with remedy actions if necessary).
      5.1.4. Train the neural networks encoded by the offspring.
      5.1.5. Evaluate offspring and compute their fitness values.
      5.1.6. Insert the offspring into the population, according to the Pareto reservation strategy.
      5.1.7. Once at *No_migr* generations, exchange individuals with the other subpopulation (migration stage).
      5.1.8. Adapt goals and compute fitness values.
   5.2 Determine the best individual(s) of the main subpopulation.
6. Determine best individual(s) over all performed generations.
7. Train the selected neural model (considering bigger values for *Ngen_Tr, Nind Tr_1, Nind_Tr_2*).
8. End of the algorithm.

5 APPLICATION

The performances of DCWNNs are investigated with respect to the identification of an industrial evaporator system [8,11]. Details are given in the following.

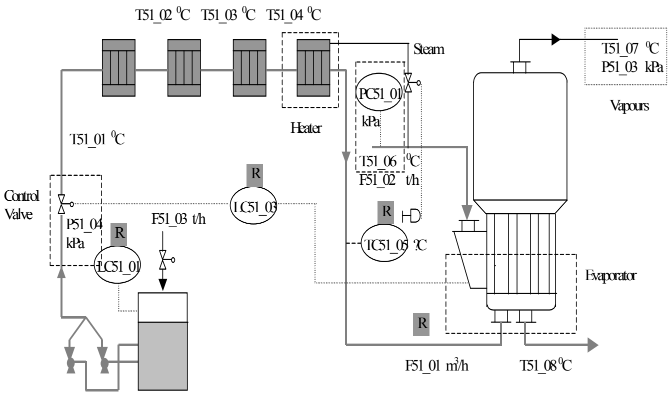

Fig. 4. First section of the evaporation station (Sugar factory, Lublin, Poland)

The evaporator is a component sub-process of the first section of the **E**vaporation **S**tation (ES) from the Sugar Factory of Lublin, Poland (Fig. 4). The ES has to increase the concentration of the sucrose juice. The thin juice passes, in sequence, through all five sections of the ES, each one reducing the water content. The steam recovered from a stage is used as heating source in the next section. The evaporator is a multi input single output process. The inputs of the systems are the steam flow to the input of ES, the steam temperature at the input of ES, the juice temperature after the heater. The juice temperature after section 1 of ES represents the output of the evaporator. No analytical model is known for this process.

The model of the system is designed using real data from the sugar factory. A large collection of measurements was available. They were acquired with the sample period Ts =10 sec, during one month of plant exploitation. The selected learning data set contains 3000 rows and corresponds to a period of time of about 8 hours, i.e. a production shift. It illustrates the maximum possible excitation of the process and it includes a reduced number of missing or uncertain values. The isolated missing and uncertain values have been replaced by means of polynomial interpolation. In order to reduce the noise, a low - pass filtering, based on 4^{th} order Butterworth filters, has been performed. This also allows the reduction of the amount of data used during the learning stage. The data have been decimated using each 10-th sampled value. The validation of the neural model is done with respect to another testing data set, which includes measurements collected from the previous month of plant exploitation.

For stability reason, series-parallel schemes are utilized, meaning that the output of the plant is fed back into the neural model during the training stage.

The DGNN had 4 inputs, representing the current values of the process inputs and the plant output values obtained at the previous sampling moment, as indicated in equation (8). The ARMA filters' parameters were selected between –5 and 5. No a priori information about the process dead time and process order is required. In all experiments, a reduced number of hidden neurons was sufficient, i.e. $n = 4$. This allowed for a fast evaluation of the DCWNNs and for a significant reduction of the search space.

A set of preliminary experiments was carried out, in order to find appropriate values for all parameters of the design procedure.

It is advantageous to set low values for *Ngen_Tr*, even in combination with a high population size, because the local optimisation procedure must be applied sequentially, iteration by iteration, to each chromosome, but the genetic search can support a parallel implementation. If the *Ngen_Tr* value is too low, the topology of the best neural model can result very different from a generation to another one. The procedure can offer, as final solution, sometimes very simple architectures and sometimes very complex and bad adapted architectures, due the fact that the evaluation is made on insufficient trained networks. Low values for *Nind* cannot support an efficient exploration of the search space and the obtained results are unsatisfactory.

Also, if the number of generations considered between two successive migrations is too small, a premature exchange of information between the main and the auxiliary subpopulation can be allowed, with negative effect on the exploration capabilities of the algorithm.

The results commented in the following were obtained considering a population of $Nind = 160$ individuals, $Max\_gen = 200$ evolutionary generations and $No\_migr = 25$ generations between two successive migrations. The training procedure was applied for $Ngen\_Tr = 80$ iterations, considering $Nind\_Tr\_1 = 100$ test individuals per iteration at the first stage and $Nind\_Tr\_2 = 40$ test individuals at the second stage. The best neural model obtained at the end of the evolutionary loop was supplementary trained for $Ngen\_Tr = 800$, $Nind\_Tr\_1 = 3000$, $Nind\_Tr\_2 = 300$.

The selected DCWNN provides a good approximation of the normalised training data set. Its generalisation capabilities are illustrated in Fig. 5, with respect to the testing data set.

Also, this neural model has a reduce order of complexity, as suggested by the corresponding low-priority objective values: $f_2 = 1; f_3 = 4; f_4 = 0; f_5 = 0; f_6 = 5$. The topology includes one hidden neuron, connected with all neural inputs: two connections are characterised by simple weights and the other two connections have active ARMA synaptic filters. No output filter is included in the selected neural architecture. The structure also contains an activation ARMA filter for the hidden neuron and a synaptic ARMA hidden filter for the output neuron.

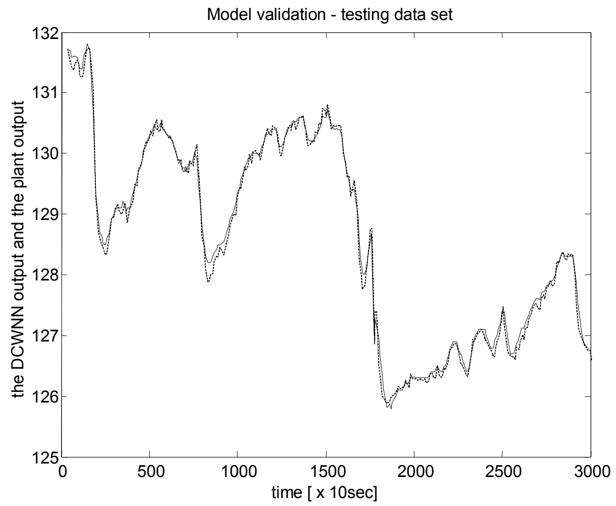

Fig. 5: Evaporator subsystem – model validation on the testing data set (one step ahead prediction). Here the output of the process is indicated with solid line and the DCWNN output with dotted line.

6 CONCLUSIONS

The paper investigates the identification capabilities of new dynamic neural networks. The presented approach improves the computational efficiency of the neural topologies with complex parameters, by providing the static architectures with local internal dynamics. The resulted structures can perform accurate approximations of dynamic nonlinearities and are characterised by good generalisation capabilities.

The design procedure does not require any information about the gradient of the objective functions. It supports a flexible configuration of the neural topology, via a multiobjective optimisation. The approach can be used only for off-line identification, because it needs large computational resources.

REFERENCES

[1] T. Bäck, D. Fogel and Z. Michalewicz. *Handbook of Evolutionary Computation,* Oxford University Press, UK, (1997).

[2] L. Ferariu, and T. Marcu. "Evolutionary Design of Dynamic Neural Networks applied to System Identification", Proceedings of *IFAC Congress b'02*, Barcelona, Spain, CD-ROM-2110, (2002).

[3] C. M. Fonseca and P. J. Fleming. "Multiobjective Optimisation and Multiple Constraint Handling with Evolutionary Algorithms – Part I: A Unified Formulation", *IEEE Transactions on Systems, Man, and Cybernetics – Part A*, **28** (1), 26-37, (1998).

[4] B. Igelnik, M. Tabib-Azar and S. R. LeClair. "A Net with Complex Weights", *IEEE Transactions on Neural Networks*, **12** (2), pp. 236-249, (2001).

[5] R. Isermann, S. Ernst and O. Nelles. „Identification with Dynamic Neural Networks: Architectures, Comparisons, Applications", Preprints of *IFAC Symposium on System Identification,* Fukuoka, Japan, **3**, pp. 997-1022, (1997).

[6] K. F. Mann, K. S. Tang, S. Kwong and W. A. Halang. *Genetic Algorithms for Control and Signal Processing*, Springer, London, UK, (1997).

[7] T. Marcu, L. Ferariu and P. M. Frank. "Genetic Evolving of Dynamic Neural Networks with Application to Process Fault Diagnosis", Proceedings of *European Control Conference*, Karlsruhe, Germany, CD-ROM, F1046-1, (1999).

[8] T. Marcu, L. Mirea, L. Ferariu, P. M. Frank. "Miscellaneous neural networks applied to fault detection and isolation of an evaporation station", Proceedings of *4th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS'2000*, Budapest, Hungary, **1**, pp. 352 – 356, (2000).

[9] K. Rodriguez-Vazquez and P. J. Fleming. "A Genetic Programming NARMAX Approach to Nonlinear System Identification", Proceedings of *GALESIA'97*, Sheffield, UK, pp.409-413, (1997).

[10] H. Tamaki, H. Kita and S. Kobayashi. "Multiobjective Optimisation by Genetic Algorithms: A Review", Proceedings of *Conference on Evolutionary Computation*, Nagoya, Japan, pp. 517-522, (1996).

[11] P. Wasiewicz. "Description of sugar technology process", Prep. of the *EC INCO-Copernicus Workshop on Integration of Quantitative and Qualitative Fault Diagnosis Methods within the Framework of Industrial Application*, Kazimierz, Poland, pp.17-23, (1998).