Adaptive Process Control using Recurrent Neural Networks: a Feasibility Study

Thierry Catfolis and Kürt Meert

Expert Systems Applications Development Group
Department of Chemical Engineering
Katholieke Universiteit Leuven
de Croylaan 46, B-3001 Heverlee, Belgium
Thierry.Catfolis@CIT.KULeuven.ac.be
Kurt.Meert@CIT.KULeuven.ac.be

Abstract

In this paper we present the results of a feasibility study about the use of recurrent neural networks for adaptive process control. A network architecture based on the Real-Time Recurrent Learning (RTRL) algorithm developed by Williams and Zipser (1989) is tested on a bioreactor control problem. Two networks, the first a model of the system, the second a controller, are clustered together into one large network. Both parts are adaptive and react on changes of the real system. The results of the experiment show that a recurrent network can easily be used as a controller and that the presented adaptation techniques are adequate for use in time-evolving environments.

1. Introduction

The use of artificial neural networks for real-life problems has been studied extensively during the last five years. Applications range from character recognition (Hussain and Kabuka, 1994) over operational research (Burke, 1994) to finance (Chakraborty et al., 1992). Typical chemical engineering problems like process modelling, diagnosis, system behavior prediction or process control are difficult to solve due to the non-linearity and high complexity of the system. The basic properties of artificial neural networks, like their approximation capabilities, their ability to learn, their robustness to noise and their non-linear behavior, seem to make them good candidates for solving these chemical engineering problems. Narendra and Parthasarathy (1990), Mayosky et al. (1993) or Turner et al. (1994) proposed to use artificial neural networks for the identification and control of dynamic systems. Ungar et al. (1990), Catfolis (1993) or Farell and Roat (1994) used neural networks for fault diagnosis and process monitoring purposes. In this paper we propose a recurrent network based architecture for process control. The non-linear properties of the network make control of a complex non-linear system feasible and the temporal processing properties of the recurrent network make the implementation of such an adaptive controller easier.

In the next section we give a small introduction on neural networks and describe the RTRL algorithm that is used in the experiments. In section 3 we review the use of artificial neural networks in process control. In section 4 we build the adaptive control architecture and in the following section we explain the adaptation principles used in this paper. In section 6 we use this adaptive controller on a bioreactor problem.

2. Neural networks and dynamic systems

Artificial neural networks are a mathematical representation of some elementary brain processes. Basically, these networks are a complex structure of processing elements or nodes connected together with links. One of the most important properties of artificial neural networks is their learning capability. For an introduction in artificial neural networks see Matheus and Hohensee (1987). For an in depth description see Rumelhart et al. (1986). The different network architectures can be divided into two classes, based on the network topology.

2.1. Feedforward neural networks

The first class, called feedforward networks, have a simple topology that do not contain directed loops. Many types of feedforward neural networks exist but the best known type is certainly the back-propagation network. This is basically a gradient descending optimization algorithm. For a description of this algorithm see Freeman et al. (1991) and Rumelhart et al. (1986). These networks perform a static non-linear mapping of the input data on the output data based

on some examples learned during the training - or learning - phase. This property makes them a good candidate for solving static problems. Because of their simple topology, they were until recently the most studied networks. Many of the first real-world neural network applications were based on feedforward networks due to the well known behavior of these networks.

The strength of these networks (their simple architecture, their static behavior) is also their weakness. Dynamic problems, such as control problems, are very difficult to solve without some complex manipulation (data buffering, time delays, ...) of the data. For details on these problems see Catfolis (1994).

2.2. Recurrent neural networks

The second class of network architectures are the recurrent networks. These networks contain directed loops in their representing graph. These feedback connections enable these networks to solve dynamic problems. There exist also many types of recurrent neural networks. The most general recurrent architecture is the fully connected architecture where all nodes are connected to all nodes. In this paper we will concentrate on one specific type of fully connected networks that will be used in the following experiments: the real-time recurrent learning algorithm, developed by Williams and Zipser (1989). This algorithm is also a gradient descent method and since the learning phase is activated after each time step rather than after a certain number of time steps, this algorithm is well suited for on-line learning, and thus for adaptation problems.

We describe here the basic elements of the RTRL algorithm needed to understand the proposed control architectures in the next sections. For a complete description and derivation of the algorithm see Williams and Zipser (1989).

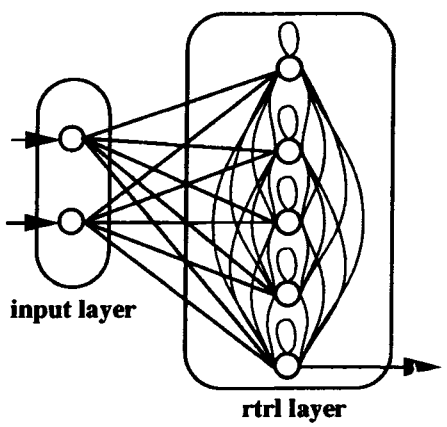

Figure 1. A basic RTRL network with 2 input nodes (m=2), 5 recurrent nodes (n=5) and 1 target node. The first layer is connected to the second RTRL layer. The RTRL layer is fully connected

A basic RTRL network consists of two layers, an input layer with m nodes (m is the number of input data from the environment used by the network) and a RTRL output layer with n nodes. The set of input nodes is called I, the set of recurrent nodes U. A subset of the RTRL nodes is called the target set (T) and consists of the target nodes. These are the nodes with a known, desired output value in the training set. Each RTRL node has a set of input links, an activation (the sum of the weighted inputs), an output value (the sigmoid transformation of the activation) and a set of output links. The input nodes have only an output value (the input data from the environment) and a set of output links. The output links from these input nodes is directed to all nodes in the RTRL layer. The nodes in the RTRL layer have their output links directed to all nodes in the RTRL layer. All links have a weight that can be changed during the training phase. Figure 1. gives an example of a RTRL network with 2 inputs and 1 output.

2.2.a. Network dynamics in the learning phase

For a RTRL node i (thus $i \in U$), we can calculate the output value $y_i(t)$ at time t by means of

$$y_i(t) = F_i\left(\sum_j y_j(t-1) * W_{ij}(t-1)\right) \quad \text{for } j \in U \cup I$$

with $W_{ij}(t)$ the weight associated with the link between node i and node j and $F_i()$ is the transfer function of node i, usually a sigmoid function. The output value of the input nodes is the data from the environment.

The parameter $\Delta W_{ij}(t)$ used by the training algorithm to update the weight W_{ij}, is calculated by means of

$$\Delta W_{ij}(t) = \alpha * \sum_k \left(e_k(t) * p_{ijk}(t)\right) \quad \text{for } j \in U \cup I \text{ and } k, i \in U$$

with α the learning rate and $e_k(t)$ the error of node k at time t. This error can be calculated with the following formulas:

$$e_i(t) = \text{target } y_i(t) - \text{calculated } y_i(t) \quad \text{for } i \in T$$

$$e_i(t) = 0 \quad \text{for } i \notin T$$

The parameter $p_{ijk}(t)$ is a dynamic parameter called impact that describes the importance of the link between node j and node i on the output value of node k. This parameter is calculated with the following formulas

$$p_{ijk}(t) = 0 \quad \text{for } t = 0, \text{for } i, k \in U \text{ and } j \in U \cup I$$

and

$$p_{ijk}(t+1) = F'_k\left(\sum_m y_m(t) W_{km}(t)\right)\left(\sum_l W_{kl}(t) p_{ijl}(t) + \delta_{ik} y_j(t)\right)$$

$$\text{for } t+1 \neq 0 \text{ and } j, m \in U \cup I \text{ and } i, l, k \in U$$

with $F'_k()$ the derivative function of $F_k()$ and δ_{ik} the Kronecker delta.

The following formulas are used to update the weights.

$$W_{ij}(t)=W_{ij}(t-1)+\Delta W_{ij}(t) \text{ for each } t$$

Before starting the learning phase all $p_{ijk}(0)$ are set to zero. At each cycle, $\Delta W_{ij}(t)$ is calculated and the new W_{ij} are calculated by adding $\Delta W_{ij}(t)$ to W_{ij}.

2.2.b. Network dynamics in the running phase

The output value of the RTRL nodes is given by

$$y_i(t) = F_i\left(\sum_j y_j(t-1) * W_{ij}(t-1)\right) \quad \text{for } j \in U \cup I$$

with W_{ij} the weight associated with the link between node i and node j. This weight is constant during the running phase. The output value of the input nodes is the data from the environment.

2.2.c. Properties of the RTRL algorithm

The RTRL algorithm has a simple computation scheme (see equations). But it has also some major drawbacks. The main problem is that the calculations are not local. This means that all weights and all activities have to be known for the calculations of a new activity. Another, practical, problem is that the algorithm has a very slow convergence. A method for speeding up the RTRL algorithm has been developed by Catfolis (1993). But the main advantage of this algorithm is the possibility to train fully connected networks without any a-priori knowledge of the temporal nature of the problem.

3. Neural network based control architectures

Control schemes based on neural networks are generally made by replacing the linear models in the classical control schemes by non-linear neural networks. This non-linear character of the networks poses severe problems to the analysis of the stability. In most cases, the neural network used in control schemes is a feedforward network. The input of the network is formed using a process input history. The assignment of inputs and outputs is determined according to the requirements of the actual problem. The schemes can easily be adapted to MIMO cases by applying the time window to all relevant process inputs and outputs. However, the careful design of history windows of process inputs and process outputs is quite difficult and depends on the system dynamics. For unknown systems, this task can become unfeasible. The use of recurrent neural networks offer a solution for this difficulty since they develop through learning an internal representation of the temporal relations between input and output. The control schemes derived for feedforward networks (Figure 2.) are significantly more complex than the one derived for recurrent neural networks (Figure 3.) .

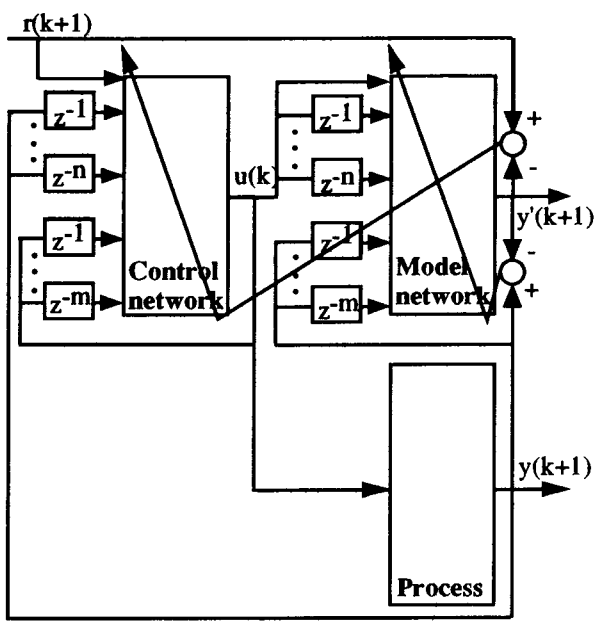

Figure 2. Indirect adaptive control using feedforward networks. Remark the complex feedback structures with the delay lines.

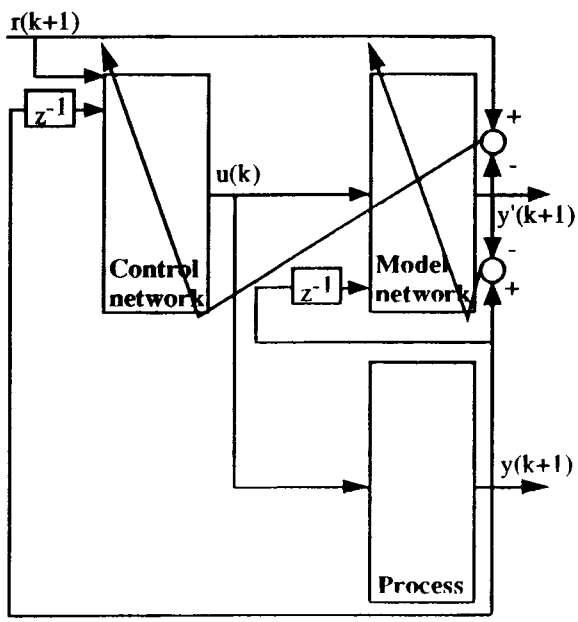

Figure 3. Indirect adaptive control using recurrent networks. Remark the difference in the feedback lines compared to the scheme with feedforward networks.

4. Application of recurrent networks in control

The training of a neural network with the RTRL algorithm requires the knowledge of input-output data. The input data for the controller network is nothing else than the classic input for a controller: the setpoint and the previous system output. The output data , on the contrary, is the error of the implemented controller i.e. the difference between the control action of the ideal controller we want to build and the control action of the implemented controller. In other words, we do not know what the output is because we don't have an ideal controller as benchmark. In the indirect scheme using backpropagation networks (Figure 2.) we are able to back-propagate the error from the network model into the controller. This solution can not be used for recurrent networks, due to the fully connected structure of these networks. Chovan et al. (1994) proposed to combine the two separate networks (the model and the controller) into one large recurrent network and to train this entire network in one step with the RTRL algorithm - see Figure 4. For this network the input-output data are known: inputs are setpoint and previous system output, output is actual system output.

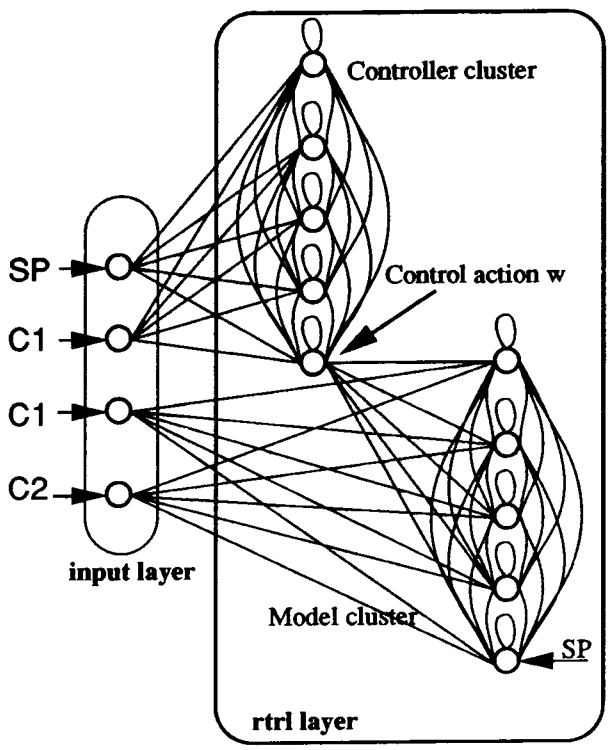

Figure 4. The clustered network consists of a model cluster and a control cluster. This network was used in the bioreactor tests.

The building of the controller consists of three steps:

1. Training of a recurrent network model of the system.
2. Add an untrained recurrent (control) network to the trained model.
3. Train the entire recurrent network without altering the model part of the network - step 3 is thus training the controller network.

5. Adaptation techniques for recurrent networks

A requirement for many control systems is adaptability. This means that the controller should be able to evolve if the real system is evolving and to lower the controller error (the difference between setpoint and system output) caused by a change of the system. These changes can be induced by many elements such as temperature evolution, fouling in heat exchangers, catalyst deterioration in reactors, etc. In the control scheme we use in this study, two neural networks are present. The first network is a model of the system, the second network is the controller. Both parts of the control architecture have to change when the environment is evolving. The adaptation of the model network is done with the error-injection principle as described by Catfolis (1994b).

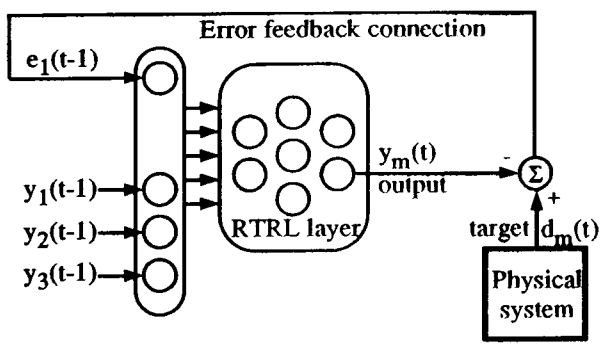

Figure 5. The error-injection principle as adaptation method for the recurrent network model.

The idea is to give as input to the model network the error of the model - the difference between model output and system output. See Figure 5 for an example. The advantages of that method are a higher stability and a better and faster model. We achieve this lower computational cost by not using the RTRL algorithm as adaptation rule.

The adaptation of the controller part is done with the RTRL algorithm. The difference between the setpoint and the model output is used by the RTRL algorithm to change the weights of the - controller part of the - network.

6. Simulation studies of recurrent neural controllers

We applied these ideas to control a bioreactor. This problem was suggested by Ungar (1990) to be used as a benchmark for neural controllers. This system consists of a continuous flow stirred tank reactor (CFSTR) containing water and cells (e.g. bacteria). These cells are consuming nutrients (substrate) and producing more cells and some other products (e.g. alcohol). Such bioreactor system can become very complex due to the self-regulatory mechanism of cells, and their changing behavior in a changing environment. For example, a small lowering of the temperature of the cells' milieu can cause a complete interruption of the cell production. However, Ungar selected a simple problem without complex structures like multiple reactors or unsteady operations. We call this system thus relatively simple, because it has only a few variables, but the control problem is very difficult due to strong non-linearities. These non-linearities make the bioreactor system also extremely complex to model. Figure 6. shows the system.

The basic equations for this bioreactor are:

$$\frac{dC_1}{dt} = -C_1 w + C_1(1-C_2)e^{C_2/\gamma}$$

$$\frac{dC_2}{dt} = -C_2 w + C_1(1-C_2)e^{C_2/\gamma}\frac{1+\beta}{1+\beta-C_2}$$

C_1 is the dimensionless cell mass conversion and is the controlled parameter, C_2 is dimensionless substrate conversion, w is the flow rate into the reactor (and thus also out the reactor because we assume a constant volume reactor)

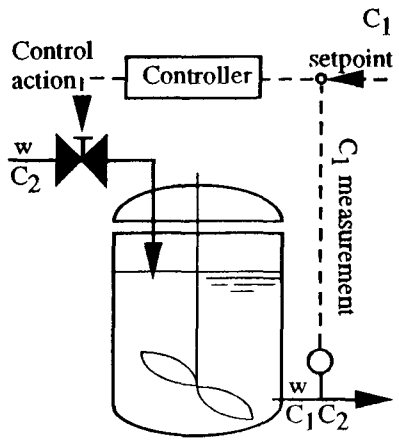

Figure 6. Basic elements in the bioreactor system. Only the relevant elements are shown (e.g. the level control system is left out for convenience).

and is the controlling parameter (control action). The constants β and γ are temperature dependent parameters controlling the cell growth and nutrient consumption. In our experiments we kept γ constant at 0.02 and we changed β between 0.48 and 0.30. The first equation explains that the change of cell mass in the reactor depends on the amount of cells leaving the reactor - $C_1 w$ - and the amount of cells created in the reactor - $C_1(1-C_2)e^{C_2/\gamma}$ - which is proportional to the amount of cells and non-linearly depending on the amount of substrate. The same reasoning can be done for the change of substrate in the reactor. As Ungar explained, this scheme is not a completely realistic model of a bioreactor, but it provides a challenging system for process

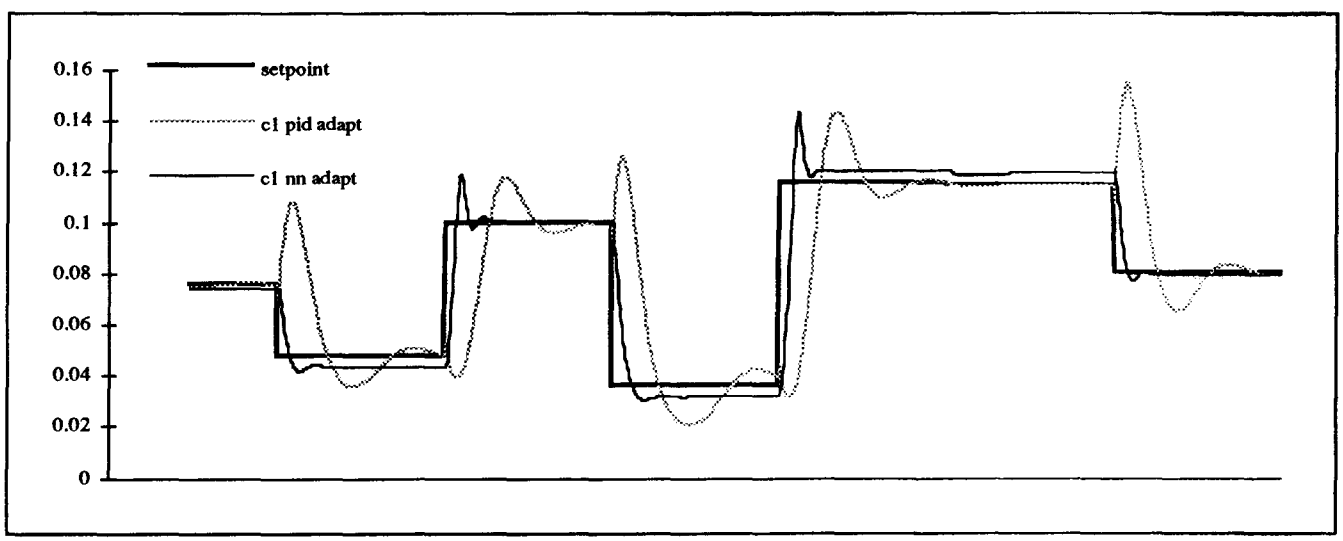

Figure 7. Example of the controller behavior in an unknown region. The adapted parameter β is here around 0.30. Shown are the setpoint of C_1, C_1 controlled by the neural controller and C_1 controlled by a PID controller.

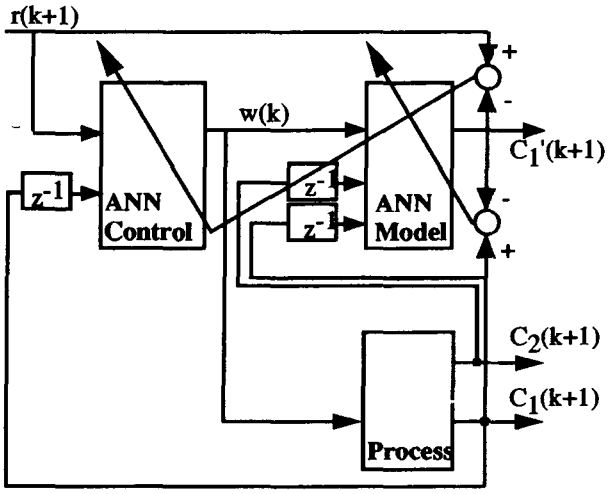

Figure 8. Indirect adaptive control using recurrent neural networks applied on the bioreactor problem.

control. There are many reasons for that: the equations are highly non-linear and exhibits limit cycles, optimal behavior occurs near unstable region and a desired setpoint can be found with different control actions.

The control architecture derived from the ideas in previous sections is shown in Figure 8.

The output of the neural controller (the control action w) is used by the model and by the bioreactor. The difference between the output of the bioreactor and the output of the model is due to the model. This error is used to adapt the network model by using the error-injection principle. The difference between the model output and the setpoint is caused by the controller. This error is used to adapt the controller network by using the RTRL algorithm. In Table I some of the results of our experiments are shown. For all experiments β evolved between 0.48 and 0.3. In experiment A we tested a neurocontroller without any adaptation possibility. In experiment B we used the same network but with the adaptation techniques. Experiment C is the same problem, but controlled with a PID controller which is tuned as good as possible. This means that, when the PID controller is tuned with a classical method (e.g. Ziegler-Nichols) for $\beta = 0.48$, it will become very unstable for all values of β lower than 0.40. The error in Table I is the average error over a range $\beta = 0.35$ to $\beta = 0.30$. The extreme value for the error of experiment C is mainly due to the change of the system.

	Experiment A neurocontrol non adaptive	**Experiment B neurocontrol adaptive**	**Experiment C PID control (non-adaptive)**
error	0.0059	0.0051	0.0131

Table I. Results of the experiments.

Figure 7. gives an example of the system behavior for $\beta \approx 0.30$ using a neurocontroller or a PID controller. Figure 9. gives an example of the model behavior for $\beta \approx 0.30$. The advantage of the adapted model is a smaller error, what will result in a better controller performance.

Conclusion and further research

In this paper we have demonstrated that recurrent neural networks are good candidates for solving complex control problems. The bioreactor benchmark proposed by Ungar is very difficult to control with a simple PID controller, due

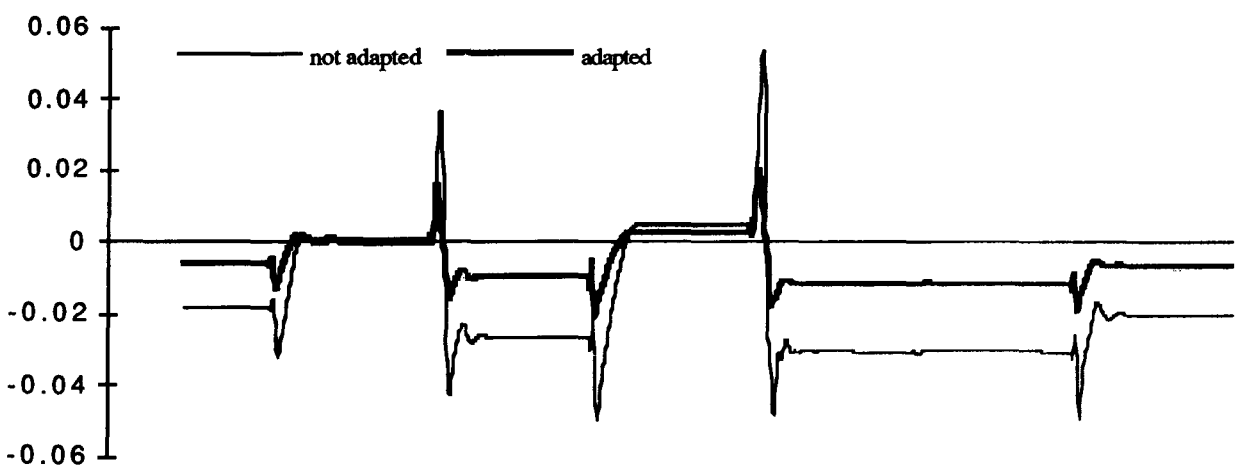

Figure 9. Example of the model behavior in an unknown region. The adapted parameter β is here around 0.30 . The bold line is the error of the adapted model (model output - system output), the normal line the error of the not adapted model.

to the non-linearity, the unstable regions and the multiplicity of the problem. For the adaptive neurocontroller, the results were promising. The use of recurrent networks makes the implementation of the control architecture easier than when using feedforward neural networks. The only problem encountered is the error of the system model. Further research will focus on two domains. First, how to solve the model error ? Possible solutions are to use a neural controller together with some classic controllers or to use models with a more than one-step ahead prediction. The second research domain is a stability analysis of recurrent neurocontrollers.

Acknowledgement

This research was done at the Expert Systems Applications Development Group, headed by Prof. M. Rijckaert.

References

Burke, L.I. (1994) "Neural methods for the Travelling Salesman Problem: Insights From Operation Research.", *Neural Networks*, vol. 7, p. 681-690

Catfolis, T. (1993) "A method for Improving the Real-Time Recurrent Learning Algorithm.", *Neural Networks* vol. 6 p. 807-821

Catfolis, T. (1993) "Monitoring a Control System with a Hybrid Neural Network Architecture.", *Proceedings of the International Conference on Artificial Neural Networks '93*, p. 854, Amsterdam, Nederland

Catfolis, T. (1994) "Mapping a Complex Temporal Problem into a Combination of Static and Dynamic Neural Networks." , *Sigart Bulletin* , vol. 5, no. 3, p. 23-28

Catfolis, T. (1994b) "Generating Adaptive Models of Dynamic Systems with Recurrent Neural Networks.", *proceedings of the IEEE International Conference on Neural Networks '94*, vol. 5, p. 3238-3243, Orlando, Florida

Chakraborty, K. , Mehrotra, K. , Mohan, C.K. and Ranka, S. (1992) "Forecasting the Behavior of Multivariate Time Series Using Neural Networks", *Neural Networks*, vol. 5, p. 961-970.

Chovan, T., Catfolis T., and Meert K. (1994) "Process Control using Recurrent Neural Networks." *preprints of the 2nd IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control*, p. 135-140, Lund, Sweden

Farell, A.E. and Roat, S.D. (1994) "Framework for enhancing fault diagnosis capabilities of artificial neural networks.", *Computers and Chemical Engineering*, vol. 18, no. 7, p. 613-635

Freeman, J.A. and Skapura D.M. (1991) Neural Networks. Algorithms, Applications and Programming Techniques. Addison -Wesley, New York.

Hussain, B. and Kabuka, M.R. (1994) "A novel Feature Recognition Neural Network and its Application to Character Recognition.", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, p. 98-106

Matheus, C.J. and Hohensee, W.E. (1987) "Learning an artificial neural systems.", *Computing Intelligence*, vol. 3, p. 283-294

Mayosky, M.A., Catalfo, J.M. and Acosta, G.G. (1993) "Neural-Net-Based Control of Dynamical Systems: A Case Study", *Applied Intelligence*, vol. 3, no. 4, p. 267-274

Narenda, K. , Parthasarathy, K. (1990) "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Transactions on Neural Networks*, vol. 1, no. 1, p. 4-27.

Rumelhart, D.E, Hinton, G.E. & Williams, R.J. (1986) "Learning Internal Representation by Error Propagation.", in D.E. Rumelhart, J.L. McClelland and the PDP Research group (eds.) Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Volume 1: Foundations. MIT Press, Cambridge, MA.

Turner, P., Montague G.A., and Morris, A.J. (1994) "Neural networks in process plant modelling and control", *Computing and Control Engineering Journal*, vol. 5, no. 3, p. 131-134

Ungar, L. H. (1990) "A Bioreactor Benchmark for Adaptive Network-based Process Control.", in W. T. Miller, R. S. Sutton and P. J. Werbos (eds.) Neural Networks for Control. MIT Press, Cambridge, MA.

Ungar, L.H., Powell, B.A. and Kamens, S.N. (1990) "Adaptive networks for fault diagnosis and process control. ", *Computers and Chemical Engineering*, vol. 14, no. 4/5, p. 561-572

Williams, R.J. and Zipser, D. (1989) "Experimental Analysis of the Real-Time Recurrent Learning Algorithm", *Connection Science*, vol. 1, p. 87-111.