Using Neural Networks for model based predictive control

M. Haak, S.Bos, S.Panic

Department of Man Machine Interaction
Delft University of Technology
Delft, NL.

***Abstract* – In this report the use of neural networks for model based predictive control is investigated. In a simulated environment a neural network was trained to predict and control an electrical Trolley on a horizontal surface. The backpropagation of error algorithm was used to train the network. In each of the 37 simulations that were performed, the focus was on some variables that are of particular interest. These variables included the total number of neurons in the network, the number of hidden layers, the number of neurons per layer, the learning rate, the momentum, the number of input parameters that the Trolley provides, and the length of the Tapped Delay Line. Under all investigated configurations the network seemed reasonably capable of learning to predict and control the Trolley plant. There appears to be an optimal trade-off between overall performance and computational resources that is guided by the problem itself instead of an intelligently designed solution.**

***Index Terms* – Neural networks, control systems, artificial intelligence.**

I. Introduction

This report investigates the use of neural networks as predictors of the behavior of a dynamic system. The neural network is used in a model based prediction control algorithm. In a simulated environment the neural network must manipulate the acceleration of a mechanical Trolley. Different configurations of neural networks will be evaluated on their ability to make this simulated Trolley stop at a predetermined location.

To perform this investigation an experiment has been designed and executed. The design and the results will be presented and discussed in the remaining sections of this report. Before clarifying the problem statement and the experimental approach, the remainder of this section provides a brief introduction to the model based predictive control and back propagation algorithms.

A Model based predictive control

Model based predictive control is an approach to controlling a dynamical system. Based on the perceived causal relationships between the input and output or state of a system, a model can be constructed. This so called plant can be used to predict the future state of a dynamic system, while it is under a varying rate of control.

Different models for plants that can be used have been identified in the literature. A general input-output model using a discrete equation has been described by Krijgsman (Jarmulak, 1994). The difference between the models can be whether they make use of prior knowledge about the system or not, or if they take external factors into account. Declerq and Keyser described the prediction model as being the neural model plus a noise model, adding some white noise to the equation (Declerq and Keyser, 1996).

The neural model can be used as a base for a predictor for the dynamic behavior of the system. This neural net based predictor is used to control the system state. The model is used to make predictions about the future state of the system, in order to control the dynamic processes. As with the models, there are different types of neural network based predictors. Declerq and Keyser compared the performance of feedforward, radial based and Elman neural networks (Declerq and Keyser, 1996). They conclude that the main problem with the performance is the validity of the neural predictor itself. They distinguish 'emulators' from 'predictors', with the former being more faithful general approximations while the latter only approximates a system with regards to a limited prediction horizon and control signal. Although the feedforward neural net did not have the shortest training time, it has a better extrapolation property and requires less neurons than the other two.

B The backpropagation learning algorithm

Backpropagation is a form of supervised learning to train MultiLayer Neural Networks (MLNNs). It requires a feed-forward perceptron neural network architecture. It was first explored in 1969 by Bryson and Ho in their paper “Applied optimal control” (Bryson and Ho, 1969). Initially it did not gain any recognition and it was not until the mid 80s that it was rediscovered by Rumelhart et. al. before this technique was valued.

The backpropagation algorithm requires a training set of input and output pairs. This makes that the algorithm is not suitable for all problems (McCollum, 2003), especially those that can’t easily be translated into an input/output pair. Applications of the technique are mostly found in the field of pattern recognition.

The problem of training MLNNs compared to single layer NNs is in the hidden layers. Since the desired output per node in these layers cannot be determined, algorithms like the delta rule (the training rule for single layer NNs) doesn’t work for the hidden layers. The error per node is calculated via the formula

$$Error_{node} = Output_{desired} - Input.$$

This formula requires the output to be measured, which for hidden layers cannot be done. The backpropagation algorithm solves this problem by assigning desired outputs to nodes in the hidden layer and finally train the NN (change the weights) using some training rule. The full algorithm consists of three steps.

- First calculate the error for the output nodes in the output layer in a feed forward fashion (this is possible because the desired output are fixed in the training input/output pairs)
- Then propagate the error in the output layer over the nodes in the hidden layer, in a backward fashion (hence the name backwards propagation). This error is a weighted sum of the connected nodes. The actual output for the hidden nodes, together with the propagated error, result in a desired output for the hidden nodes.
- Finally the weights for all of the links are updated. These weights represent the 'knowledge' of the NN, but the back propagation algorithm also uses the weights to describe the "blame" or contribution to the error. The weights are updated by some training rule in a way to minimize the propagated error.

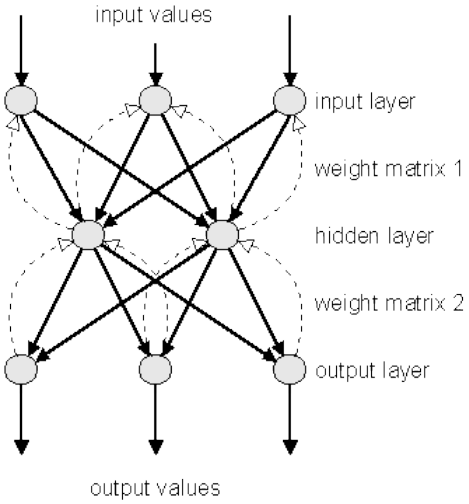

Fig. 1: Backpropagation of errors is represented by the dashed arrows that propagate the signal error from the output neurons back to the input neuron, while adjusting the weight matrices so that the propagated error is minimized

The most interesting part of the algorithm happens in the third phase. Choosing a suitable training rule that modifies these weights depends on the problem. We discuss here the generalized delta rule, which, unlike the original delta rule, is it able to deal with other activation functions besides the basic linear one. It is the most commonly used training rule for backpropagation and suitable for many classification problems. This rule is also from a mathematical point of view very sound. The generalized delta rule is a gradient descent learning rule, effectively meaning it iteratively finds weights in such a way that the propagated error per node (local error) is minimized (Negnevitsky, 2001).

Backpropogation is an iterative process, continuingly decreasing the mean squared error (MSE). The MSE will eventually converge below some pre-defined point. At this point the weights have been configured in such a way that the network has "learned" a suitable solution. It will never learn the exact solution, it only approximates it. In some cases it will never converge below the pre-defined point, usually indicating that with the current configuration a solution to the described problem can not be learned. Sometimes instead of MSE the root of the MSE is used, called the RMS error.

To speed up the learning process the learning rate parameter can be modified. The value of this variable is between 0 and 1 and is usually very small (a common value is 0.05). The backpropagation algorithm is based on small changes being made to the weights of the input at each step. If these changes are too large the performance may 'bounce around' in a counter productive fashion. In such cases the learning rate can be reduced, but at the cost of requiring more steps to reach the stopping criterion (Wilson, 2008). It is common to change the learning rate adaptively, based on the MSE. As such, when the MSE is decreasing for several epochs the learning rate is increased, and when the MSE is increasing and decreasing (oscillating behaviour) for several epochs the rate is decreased. In our study we fix the learning rate and evaluate the results for three values. To smoothen the MSE descent the generalized delta rule can be enhanced with a momentum term. Now every iteration a small part of the old weight change is added to the normal weight change. According to Watrous and Jacobs this has a stabilising effect, and the total error converges more steady (Watrous 1987, Jacobs, 1988).

II. Problem statement and evaluation

This experiment aims to investigate if neural networks can be used as an identifier and controller in model based predictive control. The training and performance results of a number of preconfigured networks will be collected during an experiment, which can be compared and contrasted in order to analyze the performance of neural networks as identifiers and controllers of second order dynamic systems.

III. Experimental approach

This experiment trains a neural network as an identifier and controller of a simulated electrical Trolley. The experiments are set up so that a number of variables of interest can be investigated. The recorded results of the

experiment will be further analyzed in the next chapter of this report.

A The Trolley as a simulated dynamic system

An electrically driven Trolley on a horizontal surface is selected as the dynamic system to use for the experiment. NeuroControl Workbench, the software package used to model and simulate the Trolley, offers two variations of this system. One version has the position of the Trolley as a single output, while the other version also outputs the speed of the Trolley. This simulation has two parameters, gain *k* and response time T. The gain is a proportional value that shows the relationship between the magnitude of the input and output signals of the system. The response time determines how fast the Trolley reacts to changes in the input signal (Jarmulak, 1994).

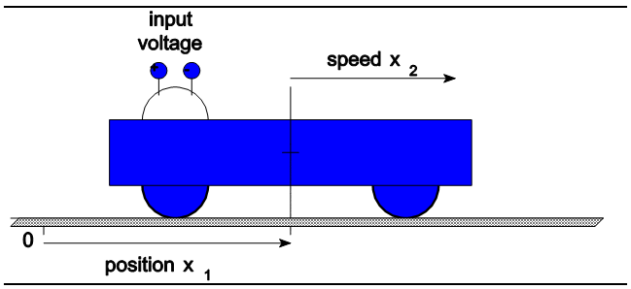

Fig. 2: An electrically driven Trolley on a horizontal surface (Jarmulak, 1994)

The Trolley is an interesting system for predictive control, as its response to a change in the input is not acute. Therefore stopping a trolley in time requires "knowledge" of how the trolley will respond with regards to change in the input. Telling the trolley to stop just as it reaches the desired end point, the trolley will overshoot, and corrective action is necessary. These repeated corrections cause an oscillation around the end point.

B Experimental environment

To model the Trolley for our research, we used the NeuroControl Workbench (NCWB) software package, developed by Jacek Jarmulak (Jarmulak, 1994). It uses plants to simulate different systems which can be trained using neural control. The notion of a Plants is a familiar one within the field of neural control. A Plant is a system where input and output are causally related. On top of that, a plant keeps an internal state, usually affecting this relation.

NCWB provides several pre-programmed plants, alongside two user definable plants. Among these plants are only two relevant to this paper, namely both Trolley plants. They differ in that one only has the current location as an input, while the other has both location and speed as inputs. How this affects the performance is discussed later in this paper.

After selecting the Trolley and configuring its basic set-up (gain *k* and response time T) the plant is ready for use. This still leaves the configuration of the neural network. NCWB provides a limited configurability of the network and its inputs. Networks can be configured to use past positions instead of just the latest location. The network itself uses a back propagation algorithm, has a maximum of three hidden layers each consisting of a user defined number of neurons, and can only further be tweaked through learn rate and momentum. The effect of these variables on the performance is discussed in a later chapter.

C Experimental procedure

Within the NCWB program a new simulation is initialized. This simulation is based on the Trolley plant, with a gain $k=0.45$ and response time $\tau=0.05$. A neural network is configured with different combinations of the variables of interest that are mentioned in the next section. A trainings set of 50 elements is generated, and the learning algorithm is set to run for a maximum of 30.000 epochs. With these settings a simulation is executed and a performance graph is recorded for further analysis.

D Variables of interest

The neural network that is used in the simulation has a number of variables that are of particular interest. These variables are: the total number of neurons in the network, the number of hidden layers, the number of neurons per layer, the learning rate, the momentum, the number of input parameters that the Trolley provides, and the length of the Tapped Delay Line (TDL)[1].

Table I and II list the different combinations of variables with which the experiment has been repeated. The resulting graphs of 37 different simulations have been recorded for further analysis.

TABLE I
THE AMOUNT OF SIMULATED NEURONS AND THEIR DISTRIBUTION ACROSS THE HIDDEN LAYERS OF THE NEURAL NETWORK. CONFIGURATION 1 IS THE BASELINE CONFIGURATION

Config.	Neurons on layer #1	#2	#3	Config.	Neurons on layer #1	#2	#3
1	5	0	0	**5**	5	5	0
2	2	0	0	**6**	5	5	5
3	1	0	0	**7**	5	10	5
4	10	0	0	**8**	2	2	0

[1]A tapped delay line (TDL) is a delay line with at least one ``tap". A delay-line tap extracts a signal output from somewhere within the delay line, optionally scales it, and usually is summed with other taps to construct the delay line signal.

TABLE II

SOME NEURAL NETWORK CONFIGURATIONS WERE REPEATEDLY SIMULATED WITH VARIATIONS TO ONE OF THESE VARIABLES. VALUES IN THE COLUMN LABELED #1 ARE PART OF THE BASELINE CONFIGURATION

	#1	#2	#3
Learning rate	0.1*	0.05	0.01
Momentum	0.9*	0.5	0.01
Tapped Delay Line	1*	2	3
Trolley Plant inputs	1*	2	

E Comparison with a baseline configuration

As a baseline for further comparison, Configuration 1 from Table I has been chosen. The learning rate of 0.1, momentum of 0.9 and TDL length of 1 were chosen as default values. This baseline configuration uses the variation of the Trolley system that provides a single input parameter, the position of the Trolley. The performance graph shown in Fig. 3 indicates that most of the networks adaptation takes place in the first 100 epochs. During the remaining 29.900 epochs the error gradient continues to converge to, but never actually reaches a minimum value. This behavior appears to continue when the network is trained with more epochs.

In the next section of this report some of the more interesting results from the different test runs will be presented.

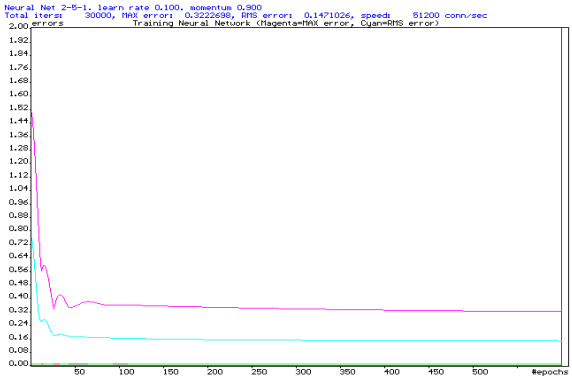

Fig. 3: Evaluating a single layer neural network with 5 neurons. The magenta line shows the maximum error, while the cyan line shows the Root Mean Square (RMS) error. The curves are the result of 30000 epochs.

IV RESULTS

In total 37 simulations have been performed and recorded with the Trolley plant in NCWB. To investigate each of the mentioned variables of interest, the recorded error graphs of selected network configurations are compared and contrasted. This is done by discussing a single graph that shows multiple RMS error curves of interest.

A Varying the total number of neurons

The graphs in Fig. 4 show the training results for a single layer network with n=1,2,5 and 10 neurons. This corresponds to Configurations 1 to 4 in Table I. Lowering the number of neurons from the baseline of 5 increases the RMS error, with n=2 giving the worst performance. The increase of neurons from 5 to 10 does not seem to have such a significant impact on the RMS error. This suggests that there exists an optimal amount of neurons for which the network is still capable of performing the computations while minimizing the RMS.

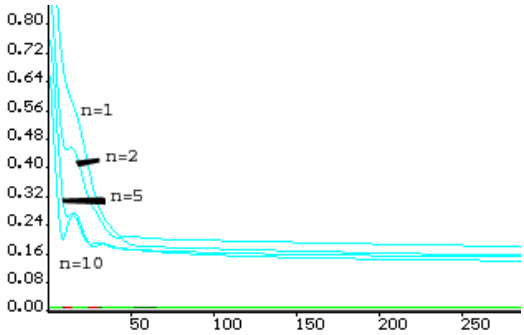

Fig. 4: Varying the number of neurons n of a single layer network. The graphs show the RMS errors in cyan.

B Varying the distribution of neurons across hidden layers

The graphs in Fig. 5 show the training results for a network with 1, 2 and 3 layers. When increasing the number of layers while keeping the total number of neurons equal, the resulting network performs about equally well after 250 epochs (l=1 and l=2). The curve for the RMS error is also more monotonous than with a single layer. However when yet another layer is added, the resulting graph (l=3) indicates that the network has more trouble with adjusting its performance.

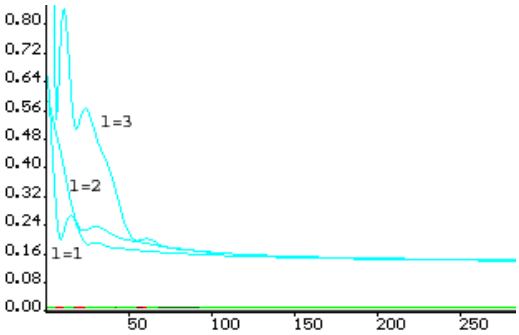

Fig. 5: Varying the number of layers l. For l=1 and 2, the total number of neurons is 10. For l=3 the total number of neurons is 15.

C *Varying the learning rate*

The graphs in Fig. 6 show the training results of the training of a network with a baseline configuration, with varying learning rates. A learning rate of 0.1 appears to lead to a minimal RMS error.

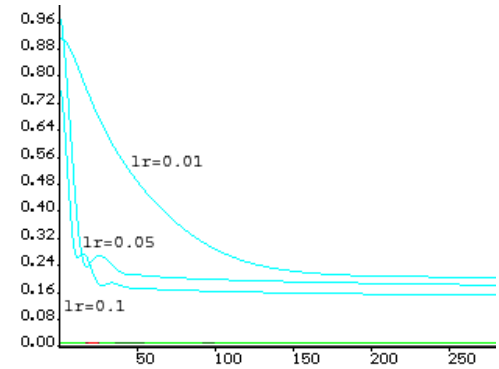

Fig. 6: Baseline configuration of the neural network, with learning rates 0.1, 0.05 and 0.01

D *Varying the momentum*

The graphs in Fig. 7 show the training results of the training of a network with a baseline configuration, with varying momentum. A momentum of 0.9 appears to lead to a minimal RMS error.

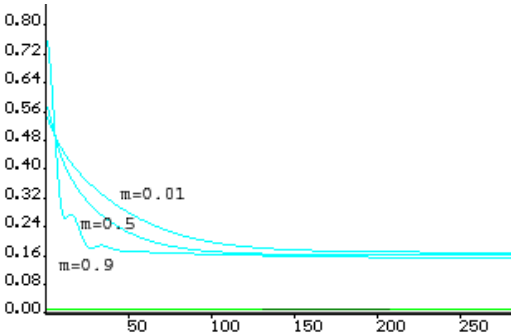

Fig. 7: Baseline configuration of the neural network, with momentum m=0.9, 0.5 and 0.01

E *Varying the length of the Tapped Delay Line*

The graphs in Fig. 8 show the training results of the training of a network with Configuration 4 from Table I, with varying length of the Tapped Delay Line. When the length increases from the default 1 to 2, the networks performance increases as the RMS error drops. However when the length is increased to 3, the increasing RMS error suggest that the network is less able to adapt. Apparently the high delay data corrupts the prediction, rather than improves it, because the information indicates less about the Trolley's current situation. A TDL length of 2 appears to lead to a minimal RMS error.

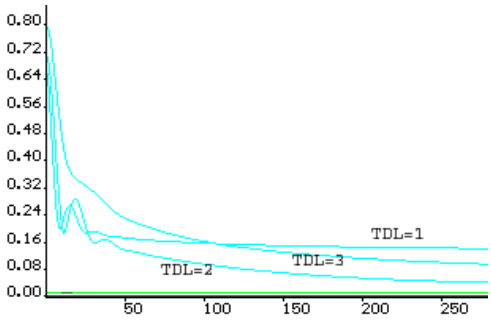

Fig. 8: Neural network Configuration 4, with varying TDL lengths

F *Varying the number of parameters provided by the plant*

NCWB's Trolley plant can provide the neural network with one or two input parameters, position and speed. To evaluate the influence of this variable we compared the RMS error curves for a number of network configurations. The graphs from configurations 2 (2 0 0), 8 (2 2 0), 5(5 5 0) and 6 (5 5 5) may offer some insights into the behavior of basic neural networks. The second input parameter 'speed' can be helpful with training a predictor for the dynamic system. In Fig. 9 this shows in the lower RMS error when both input parameters are used. Furthermore in each case the RMS curve was more smooth when both input parameters were used.

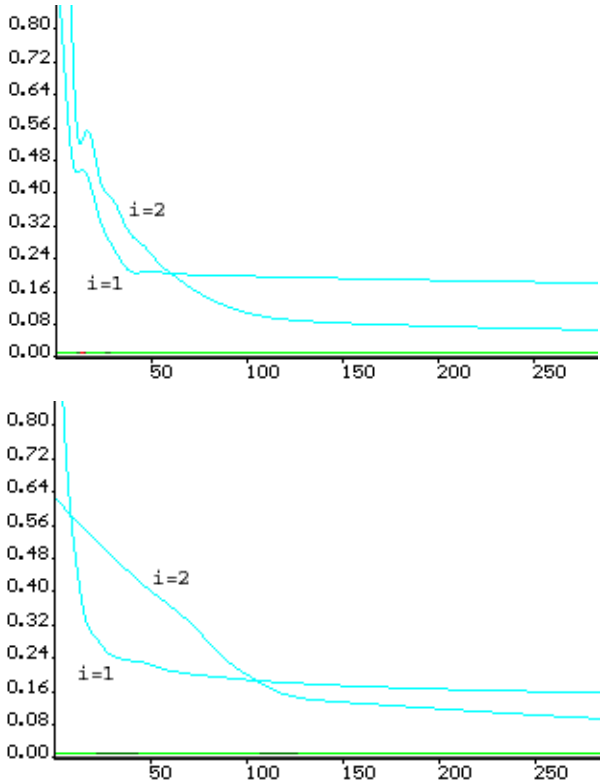

Fig. 9: From top to bottom: network configuration 2(2-0-0) and 8(2-2-0) with i=1 or 2 input parameters provided by the Trolley plant.

V. Discussion and conclusion

The results that were collected during the experiment can be used to draw some elementary conclusions about the use of neural networks for model based predictive control.

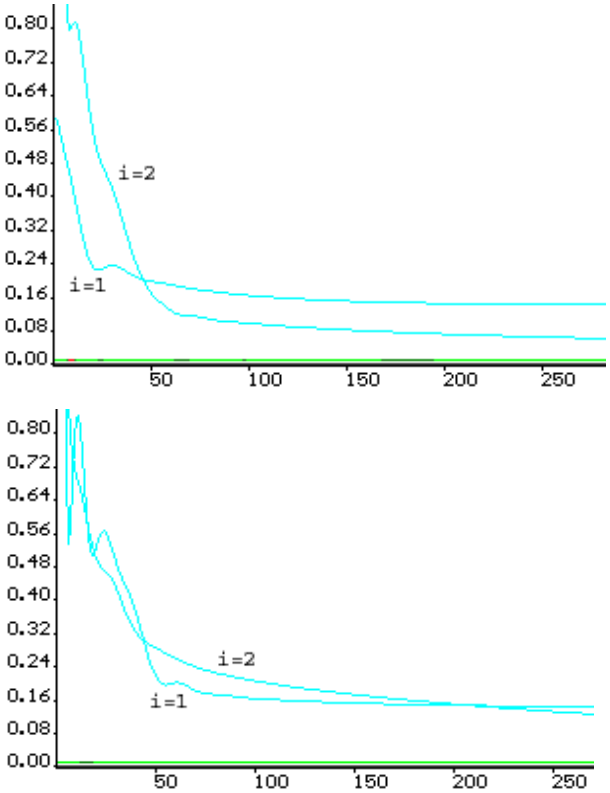

Fig. 10: From top to bottom: network configuration 2(2-0-0), 8(2-2-0), 5(5-5-0) and 6 (5-5-5) with i=1 or 2 input parameters provided by the Trolley plant.

A general observation is that the RMS error curves that resulted from changing the variables of interest, were smoother for some values more than for others. A smoother RMS error curve during the learning phase suggests that the network learns at a more predictable rate. In the case of dynamical systems more complex than the Trolley plant, it may be better to configure a network for a smoother RMS curve than for instance for minimal amount of neurons. If for instance the NN controlling the Trolley plant is integrated into a larger simulated environment, the presence of jitter on the RMS curve may make the adaptation process of dependent neural networks more difficult.

Another observation is that providing the network with more computational resources did not always seem to lead to a performance improvement. For some variables such as the number of neurons and their distribution across layers, the optimal configuration is not determined by just adding more neurons or layers, but determined by evaluating the problem that the neural network has to solve. Fig. 4 shows that the optimal solution (low RMS error and smooth curve) is for n=5 instead of 10. Fig. 5 shows that the optimal solution is for 2 layers and 10 neurons, instead of 3 layers and 15 neurons.

The graphic interface for the NCWB DOS application appears very archaic, but on the brighter side NCWB appears very stable and well documented. It provides all features required for this research[2]. Using a different simulation environment, such as MATLAB's SimuLink may provide greater flexibility and easier integration with other applications for post processing of the data.

Neural networks appear to be suitable for use as predictors of the behavior of a dynamic system. Given a few basic parameters, their capability of learning appears to quickly converge towards minimum, with a small number of epochs (250). Optimal values for the parameters can be determined by monitoring the actual network performance, rather than by using a fixed and predesigned configuration.

References

[1] Jarmulak, J. (1994), "Neurocontrol Workbench Manual", Delft University of Technology, Delft.

[2] Declerq, F. & De Keyser, R. (1996) "Comparative study of neural predictors in model based predictive control", IEEE Computer Society, Washington DC.

[3] Wilson, B. (2008). "The Machine Learning Dictionary". Retrieved July 5, 2009, Web site: http://www.cse.unsw.edu.au/~billw/mldict.html

[4] Bryson, A.E & Ho, Y.C. (1969) "Applied optimal control". Blaisdell, New York.

[5] McCollum, P. (2003) "An introduction to backpropagation neural networks". Retrieved July 5, 2009, Web site: http://www.seattlerobotics.org/encoder/nov98/neural.html

[6] Negnevitsky, M. (2001) "Artificial Intelligence – a guide to intelligent systems", Addison Wesley.

[7] Watrous, R. L. (1987). Learning algorithms for connectionist networks: Applied gradient of nonlinear optimization. In *Proceedings of the IEEE International Conference on Neural Networks, 2*, 619-627.

[8] Jacobs, R.A. (1988). Increased rate of convergence through learning rate adaptation. In: Neural Networks, 1, 295-307.

[2] Some tweaking of the DOS box window (don't run it full screen for the textual parts) will allow complete and intended usage of the program.