Performance analysis of neural network based controller for nonlinear systems

Mirela TRUSCA
Automation Department
Technical University of Cluj-Napoca,
St. Daicoviciu, 15, Cluj-Napoca,
ROMANIA
Mirela.Trusca@aut.utcluj.ro

Gheorghe LAZEA
Automation Department
Technical University of Cluj-Napoca,
St. Daicoviciu, 15, Cluj-Napoca,
ROMANIA
Gheorghe.Lazea@aut.utcluj.ro

Abstract – **The actual trend is to combine traditional control methods with neural networks in parallel. This paper places the neural network inside the closed loop, in series with the existing controller. With the neural network inside the closed-loop, randomly initialized weights, unknown performance levels, and multiple reinitializations are more difficult. A problem not so readily seen is that the weights update rules for neural networks were not designed to work in a feedback setting but in a feed-forward setting. The derivation of update rules, particularly for back propagation, were based on the independence of the weights and the input to the neural network. For a neural network in the closed-loop, the assumption is no more valid; therefore, a new update rule had to be derived.**

I. I. INTRODUCTION

The feed-through neural network is an excellent choice when using neural networks in situations where the nonlinear plant cannot be tested in the lab and the neural network can only be converged once.

The adaptation of the weights of the neural network is usually done with the standard back propagation algorithm. The improved convergent algorithm is developed to take advantage of a priori knowledge and the closed-loop configuration, will be derived to replace back propagation. Special consideration has to be given to the fact that one of the nodes in the hidden layer is linear. Initially, it is desired that the performance be acceptable, with no degradation in performance during training. It is not desirable, nor necessary, to step too far on each iteration. The learning rate is to be set very small. When the learning rate is small, there is a greater chance that the output error will converge to local minima of the weight space. This is the engineering trade-off that there first exists a willing to make in order to guarantee initial performance.

The training of the neural network uses with white noise at the reference input because white noise has a broad band frequency content. A uniform random number generator creates a random number at each iteration. This eliminates the possibility that the neural network will memorize the solution because there is no single training set on which to perform multiple iterations. The neural network is never seeing the same training data twice. If the neural network is trained with a finite training set that is repeated, there is a risk that the neural network will memorize the training set. Checking to see if the neural network is converging, it is compared in fact the desired step response and the actual step response.

The key to the method that will be implemented is the weight initialization. The 'feed through' neural network is initialized such that, on the first iteration, the current input, u(k), to the neural network is the output. The feed-through network is a fully-connected, feed-forward neural network with a single path that is linear and that initially has weights of 1.0. All weights from the inputs to the nodes of the first hidden layer are initially zero, except the weight going from the current input, u(k), to the linear node on the hidden layer, which is initialized to 1.0. The other weights past the first layer are initially random, except for the weight from the linear node on the hidden layer to the output, which is also initialized to 1.0, as seen in Fig. 1. If the neural network structure has more than one hidden layer, then each hidden layer has a linear node and an initial weight of 1.0, connecting it with the other adjacent linear nodes.

The amount of time needed for convergence is determined by several factors. The back propagation algorithm, the learning rate, and the squashing function are some of the factors that influence the rate of convergence.

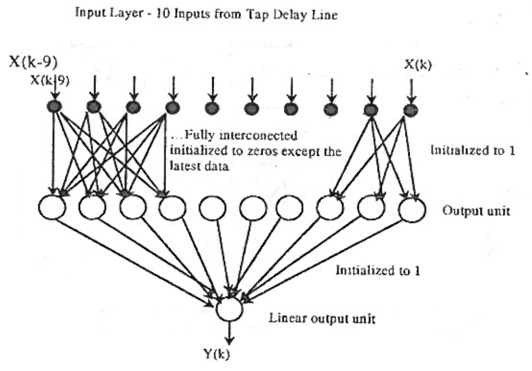
Input Layer - 10 Inputs from Tap Delay Line
X(k-9)
X(k-9)
X(k)
...Fully interconnected initialized to zeros except the latest data
Initialized to 1
Output unit
Initialized to 1
Linear output unit
Y(k)

Fig. 1: Adaptive neural network filter structure.

II. THEORETICAL CONSIDERATION

The state space controller and the plant are treated as a single unit, and the neural network is wrapped around the outside of the closed loop. This method has the advantage that the weights for the neural network can be randomly initialized because the neural network is not in the directly-closed loop; it does not affect the initial stability of the system. The disadvantage of this method is that the system for which the neural network is trying to compensate is the closed-loop dynamics of the open-loop plant due to the dynamics of the estimator when the loop is closed through the controller.

The initialization scheme was given the name "feed-through neural network" because the initial input values to the neural network become the output value until the neural network starts to converge. The closed-loop system will initially have the exact same performance with the neural network as without the neural network. The system can then be put on-line, and the neural network can be converged in the field without any initial loss of performance due to the randomly initialized weights.

III. FEED-THROUGH NEURAL NETWORK ALGORITHM

The developed algorithm is derived for minimizing the squared error between the output of the neural network and the ideal output of the neural network. The ideal output of the neural network is calculated from the reference model and the inverse model to the plant model, as seen in Fig. 2.

Terms used in Fig. 2 and throughout this work are listed below.

x is the input into the system;
u is the output of the fixed gain controller;
u_f is the output of the neural network;
W_1 is the weights of the hidden layer;
W_2 is the weights of the output layer;
P is the discrete plant model;
C is the fixed gain controller;
P_m is the discrete reference model;
μ is the learning rate;
y is the output of the plant;
z is the output of the hidden layer;
f is the squashing function;
f' is the derivative of the squashing function;
y_m is the model output;

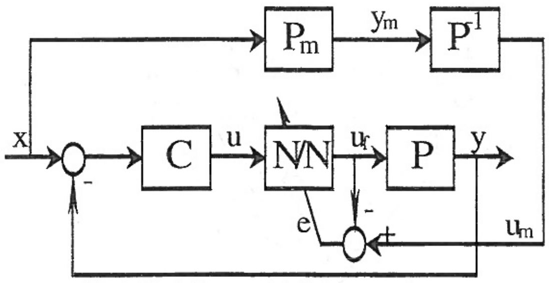

Fig. 2: Block diagram of clode-loop control system

u_m is the input to the plant that would result in y matching y_m;
X is a vector of past values of x; and
Y is a vector of past values of y.

The output layer and one of the nodes on the hidden layer have a linear squashing function. This allows the feed-through neural network to be applied. The update algorithm is broken into two parts. There is a different algorithm for each layer, but the same error is minimized for each layer. Equation 1 defines the error to be minimized by this algorithm. The error is the difference between the output of the neural network and the ideal output of the neural network. The ideal output of the neural network is calculated by using the output of the reference model and the inverse of the plant model. The plant is not modeled exactly because the plant has nonlinearities that are not modeled. However, this is the plant model used to develop the fixed gain controller. Although the ideal output of the neural network is not exact, the result is that a large portion of the plant's dynamics will be accounted for. If the plant were modeled exactly, the neural network would not be required. Several equations that define key relationships can be seen in Equations 1 through 7.

$$e = u_m - u_f \tag{1}$$

$$uf = zW_2 \tag{2}$$

$$z = f(W_1 U) \tag{3}$$

$$u = C(x - y) \tag{4}$$

$$y_m = P_m x \tag{5}$$

$$u_m = P^{-1} y_m \tag{6}$$

$$y = Pu_f \tag{7}$$

The update algorithm for the output layer can be seen in Equations 8 through 14. The partial derivative of the error squared is taken with respect to the weights in the output layer, as seen in Equation 8.

$$\frac{\partial e^2}{\partial W_2} = -2e\frac{\partial u_f}{\partial W_2} \tag{8}$$

Equation 9 is the partial derivative of the output of the neural network to the weights of the output layer. Because the output layer has a linear squashing function, the derivative of the squashing function is unity.

$$\frac{\partial u_f}{\partial W_2} = z + W_2\frac{\partial z}{\partial W_2} \tag{9}$$

Equation 10 is the partial derivative of the output of the hidden layer with respect to the weights of the output layer. For the hidden layer, the squashing function is not assumed to be linear. The derivative of the squashing function is not unity but f'.

$$\frac{\partial z}{\partial W_2} = \frac{\partial f(W_1 U)}{\partial W_2} = f' W_1 \frac{\partial U}{\partial W_2} \tag{10}$$

Equation 11 is the partial derivative of the input to the neural network, which is the output of the fixed gain controller with respect to the weights of the output layer. This is where the new update algorithm noticeably separates from the derivation of back propagation. In the derivation of the back propagation algorithm, the partial derivative of the input to the neural network with respect to the weights is zero. The reason in the new update algorithm has value for this partial derivative is because the neural network is inside the closed-loop.

$$\frac{\partial U}{\partial W_2} = \frac{\partial C(x-y)}{\partial W_2} = -\frac{\partial Cy}{\partial W_2} = -C\frac{\partial y}{\partial W_2} \tag{11}$$

Equation 12 is the assumed form for the plant.

$$\begin{aligned} y(k) &= -b_1 y(k-1) - b_2 y(k-2) - \ldots \\ &\ldots - b_l y(k-l) + a_0 u(k) + a_1 u(k-1) + \ldots \end{aligned} \tag{12}$$

Equation 13 is the partial derivative carried through the plant. A key assumption is made about the plant: the input to the plant does not directly feed through; the term a_o is zero. With this assumption, the partial derivative of the output of the plant with respect to the weights can be calculated from values of the past iterations.

$$\frac{\partial y(k)}{\partial W_2} = -b_1 \frac{\partial y(k-1)}{\partial W_2} - b_2 \frac{\partial y(k-2)}{\partial W_2} - \ldots + \\ + a_0 \frac{\partial u_f(k)}{\partial W_2} + a_1 \frac{\partial u_f(k-1)}{\partial W_2} \tag{13}$$

The update algorithm can be seen in Equation 14. The weights for the output layer can be adjusted during each iteration.

$$W_2 = W_2 + 2\mu e \frac{\partial u_f}{\partial W_2} \tag{14}$$

This is the derivation for the update algorithm of weights on the hidden layer.

The partial derivative of error squared with respect to the weights of the hidden layer can be seen in Equation 15.

$$\frac{\partial e^2}{\partial W_1} = -2e \frac{\partial u_f}{\partial W_1} \tag{15}$$

Equation 16 is the partial derivative of the output of the neural network with respect to the weights on the hidden layer. Because the squashing function is linear for the output layer, the derivative of the squashing function is unity.

$$\frac{\partial u_f}{\partial W_1} = W_2 \frac{\partial z}{\partial W_1} \tag{16}$$

Equation 17 is the partial derivative of the output of the hidden layer with respect to the weights on the hidden layer. Because the squashing function is not linear for the hidden layer, the derivative of the squashing function, f', is not unity.

$$\frac{\partial z}{\partial W_1} = \frac{\partial f(W_1 U)}{\partial W_1} = f\left(U + W_1 \frac{\partial U}{\partial W_1} \right) \tag{17}$$

Equation 18 is the partial derivative of the input to the neural network, which is the output to the fixed gain controller with respect to the weights of the hidden layer.

$$\frac{\partial U}{\partial W_1} = \frac{\partial C(x-y)}{\partial W_1} = -\frac{\partial Cy}{\partial W_1} = -C \frac{\partial y}{\partial W_1} \tag{18}$$

Equation 19 is the assumed form of the plant.

$$y(k) = -b_1 y(k-1) - b_2 y(k-2) - \ldots \\ -b_l y(k-l) + a_0 u(k) + a_1 u(k-1) + \ldots \tag{19}$$

Equation 20 is the partial derivative of the plant with respect to the weights on the hidden layer. The feed-through term on the plant's input is assumed to be zero. The value of the partial derivative of the plant can be calculated using values of previous iterations.

$$\frac{\partial y(k)}{\partial W_1} = -b_1 \frac{\partial y(k-1)}{\partial W_1} - b_2 \frac{\partial y(k-2)}{\partial W_1} - \ldots \\ + a_0 \frac{\partial u_f(k)}{\partial W_1} + a_1 \frac{\partial u_f(k-1)}{\partial W_1} \tag{20}$$

The equation to update the weights of the hidden layer can be seen in Equation 21.

$$W_1 = W_1 + 2e \frac{\partial u_f}{\partial W_1} \tag{21}$$

IV. SIMULATION RESULTS

In the simulation experiments, we use a fourth order nonlinear plant model. The stable plant has two second-order modes each lightly damped, with a damping coefficient of approximately .05. The natural frequency of the first filter in the series is 1 rad/sec; the natural frequency of the second filter is twice the first, 2 rad/sec. The saturation function inserted in between is a hyperbolic tangent function. For the unstable plant, the damping on the second filter is made negative. The estimator/regulator control system was designed based on the linear model, which was exactly nonlinear plant without the nonlinearity, to give the closed-loop system a response with 0.707 damping. Due to the nonlinearity, the closed-loop system does not perform as well as expected.

A nonlinear plant with saturation in the middle of the dynamics will be considered (open-loop stable and open-loop unstable). The structure of the open-loop process is mentioned in Fig. 3. Generally for unstable plants with neural network control, the plant is first stabilized with feedback and the neural network is added outside of the closed-loop.

There is no guarantee that the resulting closed-loop system will be stable. This is one of the reasons adaptive inverse control is seldom used on an unstable plant. The stable plant would converge when the learning rate is decreased because the neural network adds a unknown gain to the closed loop. The unstable plant is first stabilized by the estimator/regulator so the neural network can freely run at a higher learning rate. After approximately 300,000 iterations, the neural network converges to where there is marginal improvement in performance. It is evident that given several reinitializations, the neural network could converge to a spot that will give better performance than the case without a neural network, but that situation is not found during this set of experiments.

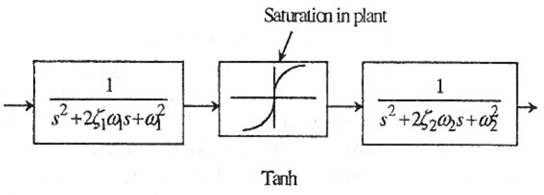
Saturation in plant
$\frac{1}{s^2+2\zeta_1\omega_1 s+\omega_1^2}$
$\frac{1}{s^2+2\zeta_2\omega_2 s+\omega_2^2}$
Tanh

Fig. 3: Structure of the open-loop process

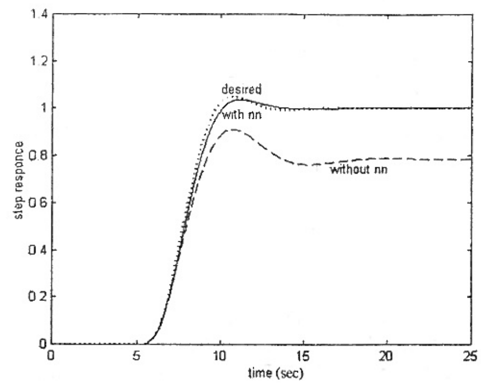

Fig. 4: Simulation of the step response of the stable plant (with (-)and without neural network(--), desired (:))

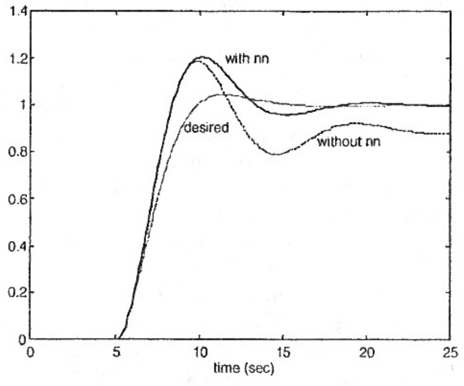

Fig. 5 Simulation of the step response of the unstable plant (with (-)and without neural network(--), desired(:))

The developed algorithm reduces the amount of effects due to the plant's dynamics that can inhibit the convergence of the neural network. By using a-priori knowledge of the system, the algorithm converges the weights of the neural network very quickly. For this example, the weights converge within 4000 iterations. This result is comparable to the back propagation results, which took 300,000 iterations to converge. The improved performance was achieved with a single convergence of the neural network. The algorithm also reduces the mean squared error of the closed-loop system from 3.27 x 10 2 to 3.27 x 10 4 , a 99% reduction of mean squared error for the step input. The results of the step input with and without the neural network can be seen in Fig. 4 and Fig. 5.

V. CONCLUSIONS

The performance of a closed-loop control system can be degraded by nonlinearities and an unexpected dynamics in the plant. The technique developed in this research can increase the performance of ill-modeled plants when a fixed-gain, closed-loop control system already exists. The feed-through neural network was devised to initially maintain the performance of the closed-loop control system. As the neural network converges, the performance of the closed-loop system will be improved. Back propagation was used to update the weights of the neural network. This technique worked well but converged very slowly. Back propagation was originally derived to work on an open-loop system, and it does not use any a-priori knowledge of the system.

The feed-through neural network is a neural network with its weights initialized to have a unity gain. The linear version of the feed-forward neural network is an FIR filter; back propagation is analogous to the LMS update algorithm for the FIR filter, which is also derived for the open loop. There did not exist an update algorithm for the FIR filter inside the closed-loop.

The amount of time required to converge is much less than that of the back propagation algorithm. This result is not surprising because the back propagation algorithm is not designed to operate inside the closed-loop. Back propagation is also not a model-based algorithm. If a priori knowledge of the system is available, the two new algorithms capitalize on this information and reduce the convergence time. The algorithms can be applied to a practical example with a greater unknown nonlinearities such as of a boiler plant that actually models the transport delays of the system.

VI. REFERENCES

[1] Bass, E. and K. Y. Lee, 1994, "System linearization with guaranteed stability using norm-bounded neural networks," *IEEE International Conference on Neural Networks*, pp. 2355-2360.

[2] Chen, F. C. and C. H. Chang, 1994, "Practical Stability Issues in CMAC Neural Network Control Systems," *Proceedings of the American Control Conference*, vol. 3, pp. 2945-2946.

[3] Choi, J. Y. and H. F. Van Landingham, 1995, "Empirical Data Modeling and State Estimation for a Steam Boiler System," *Proceedings of the 1995 IEEE International Conference on Systems*, Man and Cybernetics, pp. 3943 – 3948.

[4] Joerding, W. H. and J. L. Meador, 1991, "Encoding A Priori Information in Feed-Forward Networks," *Neural Networks*, Vol. 4, pp. 847-856.

[5] Narendra, K. S. and K. Parthasarathy, 1990, "Identification and Control of Dynamical Systems using Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 4-27.

[6] Nguyen, D. and B. Widrow, 1990, "Improving the Learning Speed of 2-layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vol. 3, pp. 21-26.

[7] Renders, J. M., M. Saerens, and H. Bersini, 1994, "Adaptive neurocontrol of MIMO systems based on stability theory," *IEEE International Conference on Neural Networks*, pp. 2476-2481.

[8] Smith, B. R., 1997, "Neural Network Enhancement of Closed-Loop Controllers for Ill-Modeled Systems with Unknown Nonlinearities", *IEEE International Conference on Neural Networks*, Vol. 4, pp. 247-256.

[9] Tripathi, N., M. Tran, and H. Van Landingham, 1995, "Knowledge-Based Adaptive Neural Control of Drum Level in a Boiler System," *Proceedings of SPIE - The International Society for Optical Engineering*, pp 160 – 171.