1990

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

NEURAL NETWORKS AS A CONTROL METHODOLOGY

Prepared By: Claire L. McCullough

Academic Rank: Assistant Professor

University and Department: University of Alabama in Huntsville
Electrical & Computer Engineering

NASA/MSFC:
Laboratory: Structures & Dynamics
Division: Control Systems
Branch: Pointing Control Systems

MSFC Colleague: Dr. Henry Waites

Contract Number: NGT-01-002-099
The University of Alabama

While conventional computers must be programmed in a logical fashion by a person who thoroughly understands the task to be performed, the motivation behind neural networks is to develop machines which can train themselves to perform tasks, using available information about desired system behavior and learning from experience.

Goals of the project begun under the Faculty Summer Fellowship program were threefold:

1) to evaluate various neural net methods and generate computer software to implement those deemed most promising on a personal computer equipped with Matlab
2) to evaluate methods currently in the professional literature for system control using neural nets to choose those most applicable to control of flexible structures
3) to apply the control strategies chosen in 2) to a computer simulation of a test article, the Control Structures Interaction Suitcase Demonstrator, which is a portable system consisting of a small flexible beam driven by a torque motor and mounted on springs tuned to the first flexible mode of the beam.

At the present time, the first two goals have been met, and work on the third is on-going. Results of each will be discussed below.

Using many references, the currently available methods for training neural nets were examined and evaluated for ease of implementation, reliability, computer requirements, and applicability to control systems. Some methods were rejected because of the vast numbers of neurons required to work practical problems (e.g., Bidirectional Associative Memories); some, for example Boltzmann machines, because of the very large amount of computer time required to train the nets; and some, like Hopfield nets, for the extreme difficulty of implementation (in order to utilize a Hopfield net, a Lyapunov function must be generated for system "goodness" and appropriate weight adjustments based on that Lyapunov function must be determined--a procedure requiring vast "mathematical expertise and ingenuity" [1]). While there is currently no optimum method for neural nets, after careful evaluation, back-propagation was chosen as the most practical choice for implementation. This method changes network weights proportional to the partial derivative of the system error function with respect to each weight. This approximates a gradient descent procedure, and therefore assures that the system will reach an energy minimum. Difficulties with back-propagation include possible network paralysis if neurons saturate, the possibility of reaching a local rather than a global minimum, and long training times. However, the method is very easy to implement algorithmically, and is used in the majority of the controls applications appearing in the current literature. Methods have been proposed to fix difficulties with back-propagation, but each has its own associated problems (for example, Cauchy training eliminates the problem of convergence to local minima, but has a greater instance of network paralysis than systems using back-propagation, and a training time one hundred times that of the already lengthy back-propagation training). Thus back-propagation was chosen as the neural net training method to be implemented.

Using Matlab, software was generated implementing a back-propagation trained neural net on an IBM compatible personal computer. For a given problem, number of layers and number of neurons must be "empirically determined," [2] so neural nets of several sizes and configurations were compared. Some authors have hypothesized that fewer neurons may be used for a given problem if those neurons are arranged in more layers [1]. In the trials conducted, no network was found which failed to converge eventually, so no evidence was obtained to support or disprove this hypothesis. However, empirical evidence does suggest that given that both will eventually converge to a solution, a neural net with fewer layers will converge more quickly. Figure 1, showing the error measure

(total sum square error) versus number of training epochs for a two-layer neural net (with one nonlinear hidden layer and a linear output layer) and a three-layer net (with two nonlinear hidden layers and a linear output layer), is typical of the results generated.

In the second phase of the project, recent publications in the professional literature regarding applications of neural nets to control problems were examined and compared. Methods currently available can be divided into roughly three categories:

a) methods in which a neural net is trained to emulate a currently existing controller, whether human or computerized (such as [3]);
b) methods in which nets generate some state or function which is then used in a standard controller design (for example, [4] in which the neural net is used to generate estimates of unknown nonlinear system parameters, which are then used in a standard adaptive controller);
c) methods in which the neural net generates a controller for an unknown system without human intervention [2].

Of the three types, the third is by far the most sophisticated, as it assumes no mathematical knowledge of the system to be controlled, and does not require a human to be able to control the system or to generate a controller which successfully does so. This would mean that nonlinear systems which could be modeled poorly, if at all, theoretically could still be successfully controlled by a trained neural net. It was decided that such a method would be the best candidate for controlling flexible space structures.

The particular method choosen for application to the test system was that in [2]. This is a time back-propagation system. First, a neural net must be trained to emulate the behavior of the unknown system using standard back-propagation methods. This trained emulator is then used to train the controller as follows:

1) A time trajectory for system behavior is generated, with the untrained controller generating essentially random inputs to the emulator.
2) The final emulator output is compared to the desired output.
3) The error is propagated back through the emulator to generate an equivalent controller error, which is used to train the controller.
4) The process is continued, propagating back through each time step of the trajectory until the controller has been trained for all time steps.
5) Steps 1-4 are repeated for many trajectories.

Currently on-going is work applying the method in [2] to the test article. The neural net chosen for use had one hidden nonlinear layer containing 35 neurons and a linear output layer of 10 neurons to scale the outputs. One problem in implementing the method was difficulty in obtaining accurate training data for the CSI Demonstrator; the final data was generated by Mark Whorton and John Sharkee of NASA, using a Matlab simulation of the system.

Another difficulty encountered was ill-conditioning of the data. Although it was mentioned nowhere in the literature, it was discovered that if inputs to the neural net vary by several orders of magnitude, as is the case of the Demonstrator, the nonlinear neuron layer soon saturates, so that training of that layer comes to a virtual standstill. This causes the nonlinear layer to send the same input to the linear layer regardless of the system input, causing the linear weights to grow without

bound as they try to adjust to give varying outputs a constant input. This causes the error measure to grow without bound. This problem was solved by scaling the trajectories of very large system states to bring them down to the level of the others and prevent layer saturation.

It was also discovered that the 8088 PC being used for software development was too slow to be practical in training a neural net to emulate the test article; currently the software is being run on an 80386 machine with 10,000 training patterns comprising a single epoch (a single epoch takes approximately 45 minutes on the 80386 and over 24 hours on the 8088). As yet, the emulator has not converged to zero error, but to a TSS of approximately 4. Figure 2, of TSS versus epoch number, shows this convergence. When state trajectories for both the system and the emulator are compared, results for different states range from that in state X_1 where the emulator does not adequately follow the system response (Figure 3) to state X_3 (Figure 4) in which the two are practically identical. Possible reasons for this include an inadequate training set (i.e., one which does not fully span the state space) and a neural net with an inadequate number of neurons and/or layers. Work to perfect the emulator is continuing. At such time as the emulator adequately predicts all system states, the controller will be trained as part of an on-going effort during the coming academic year. Once the controller is trained, its performance may be compared to the currently existing controllers for the system in terms of computation requirements, robustness, etc. Other neural net strategies, such as using the neural net as an estimator for system parameters needed by standard adaptive controllers, could also be addressed at some future time.

While neural nets have yet to be fully evaluated as a tool for control of nonlinear or poorly modeled systems, they show great potential in this area, and deserve further consideration and study.

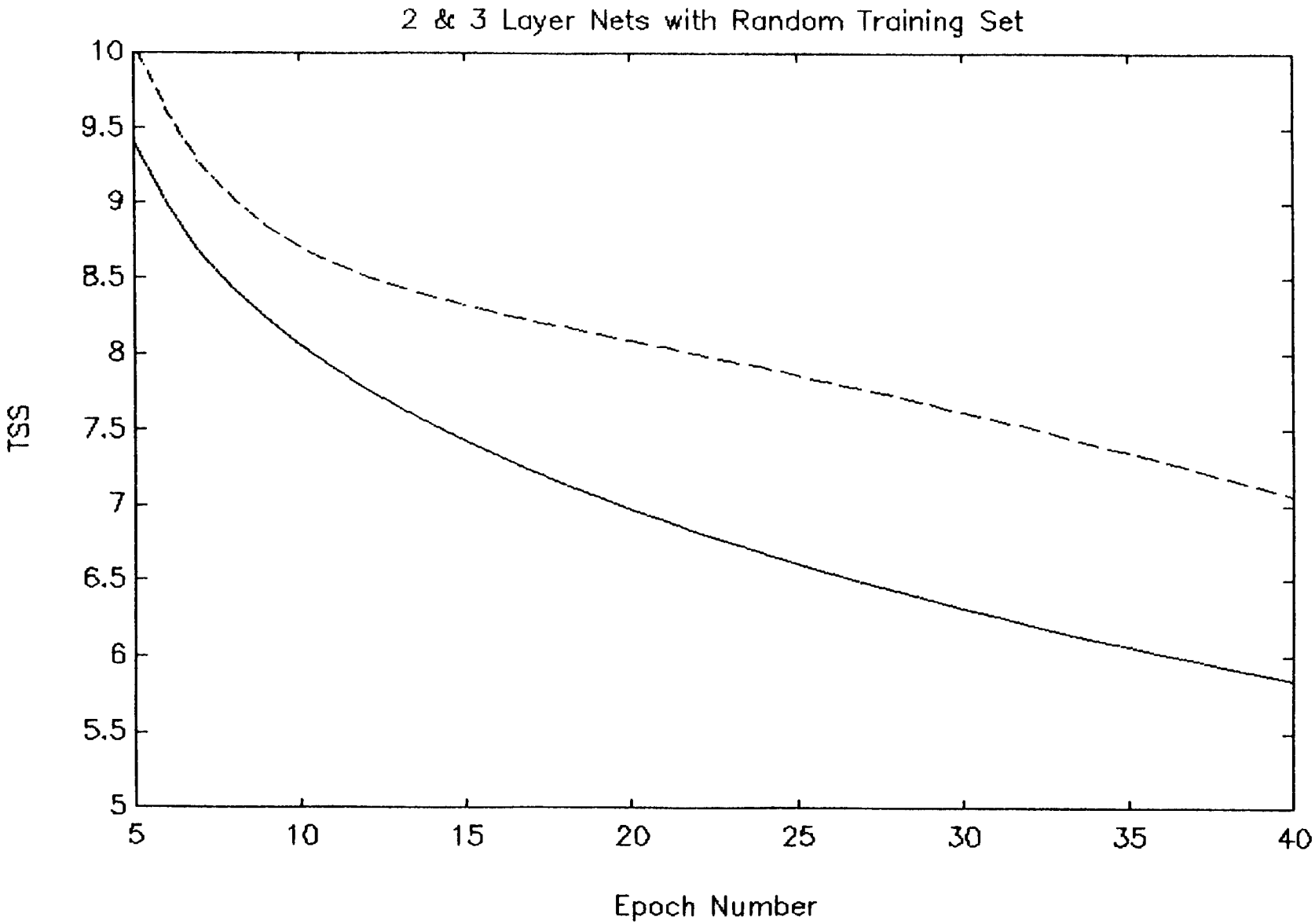
REFERENCES

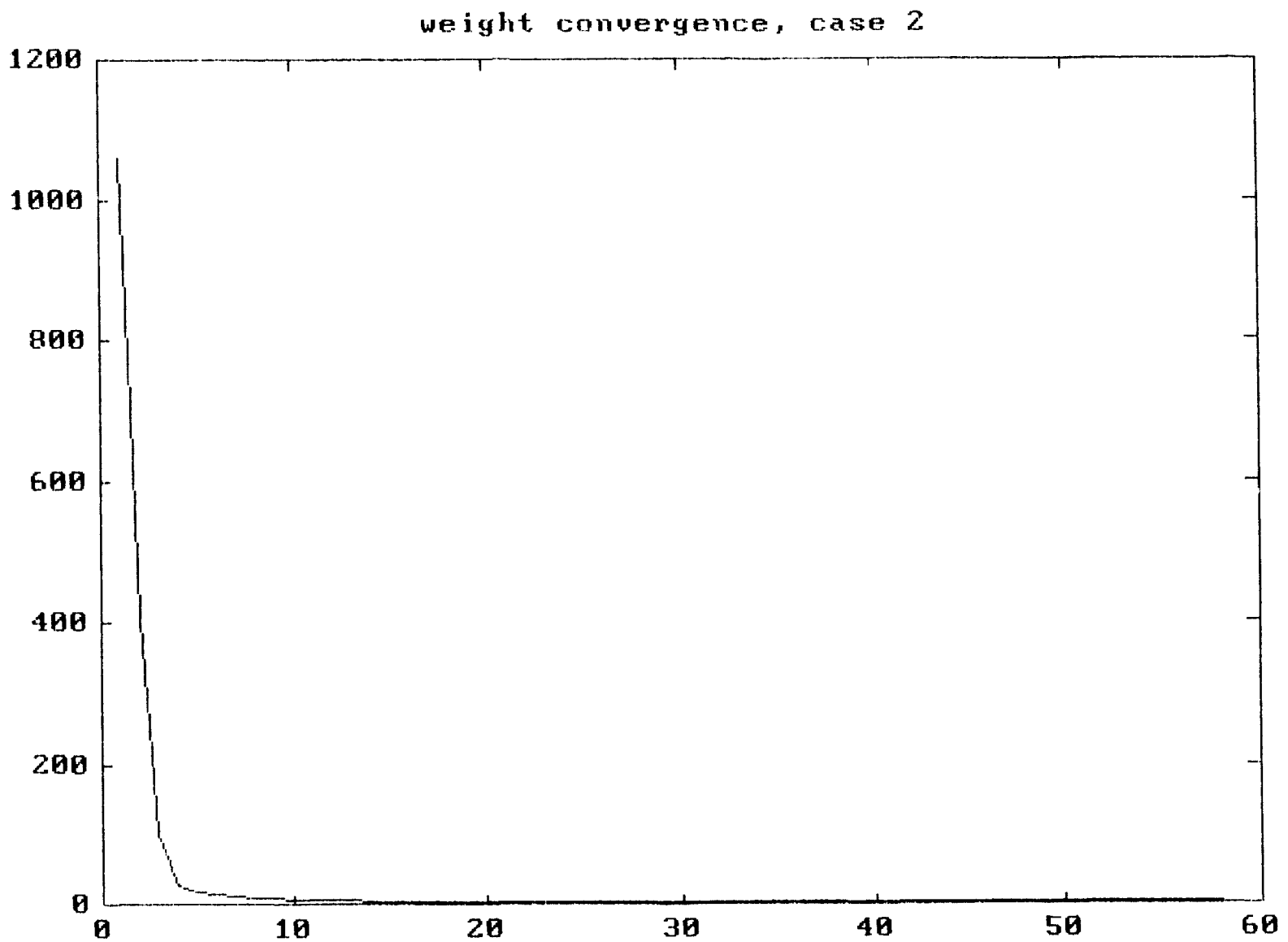
[1] Wasserman, P. D., *Neural Computing: Theory and Practice,* Van Nostrand Reinhold, New York, N. Y., 1989.

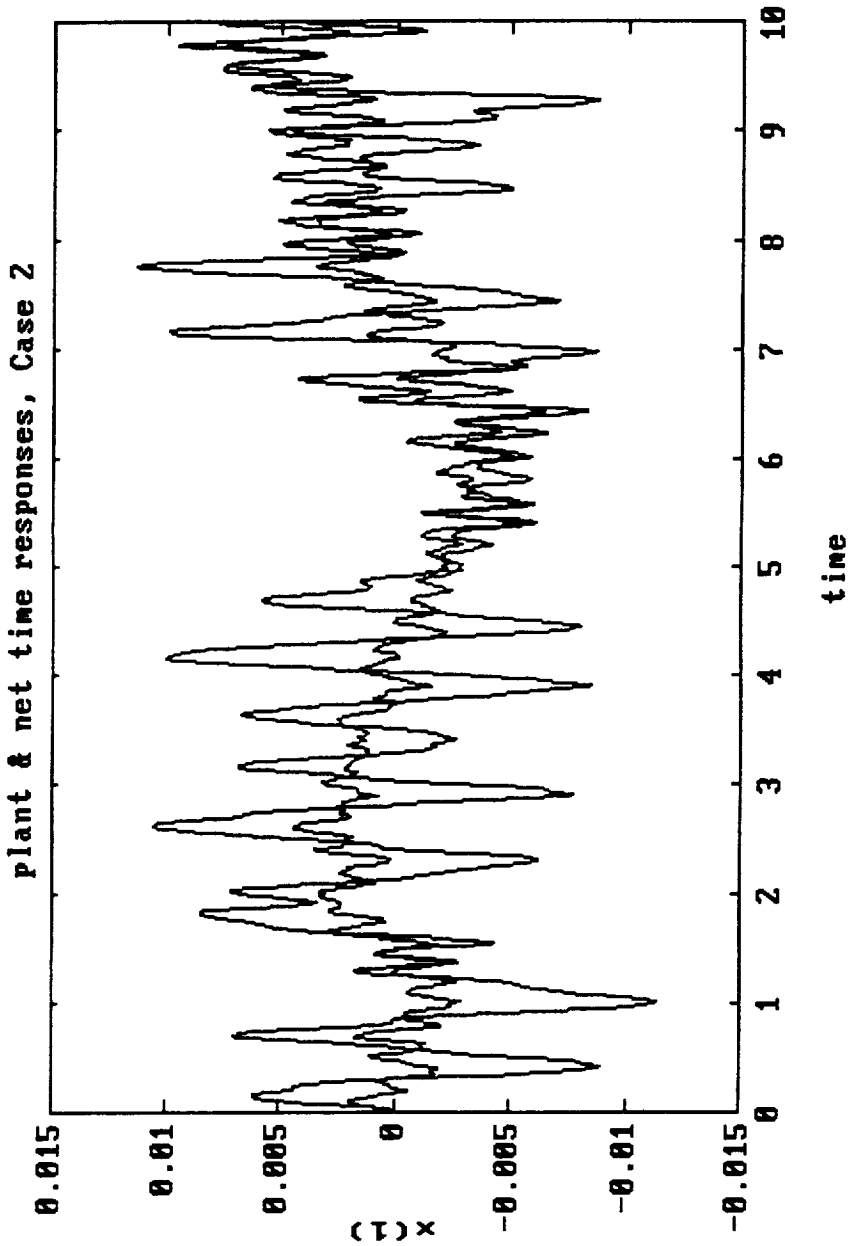
[2] Nguyen, D. H. and Widrow, B., "Neural Networks for Self-Learning Control Systems," *IEEE Control Systems Magazine*, April 1990, pp. 18-23.

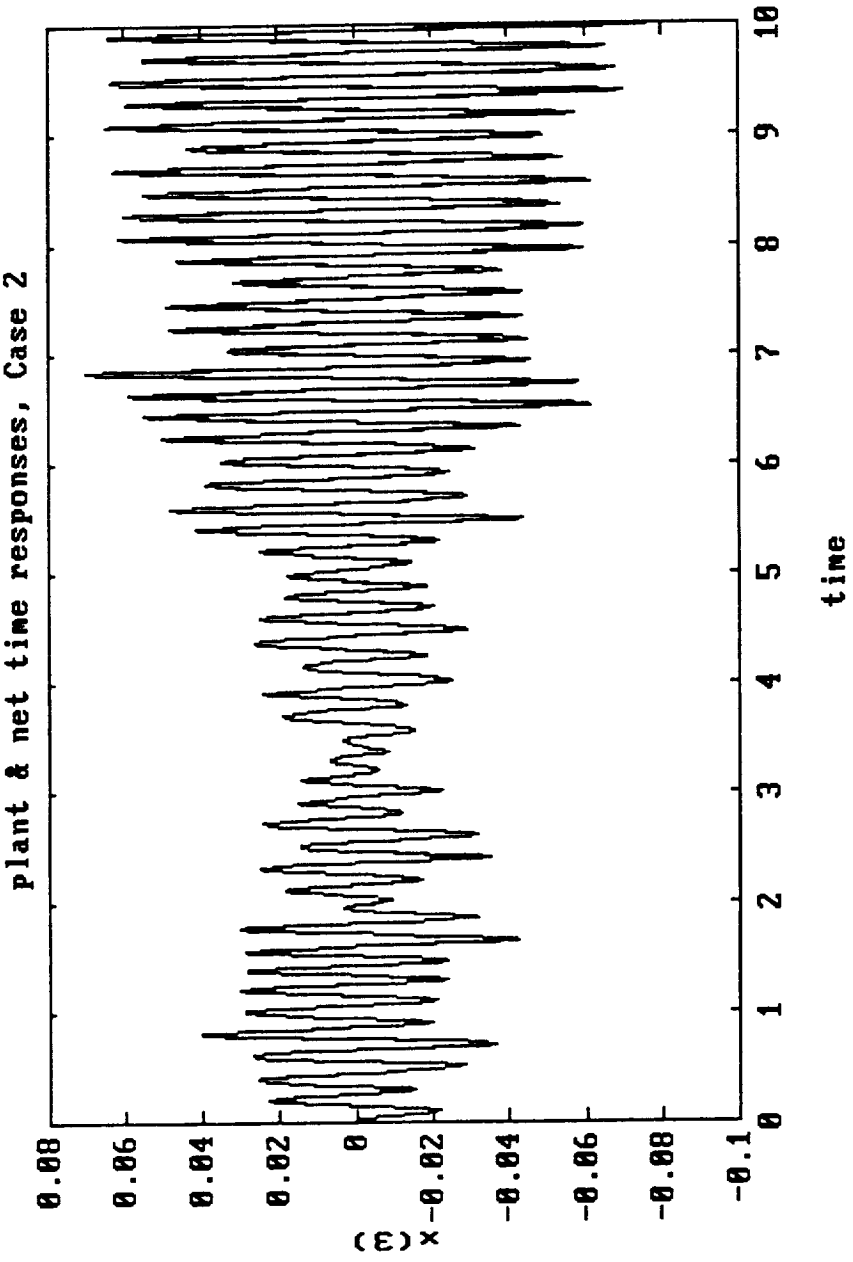
[3] Guez, A. and Selinsky, J., "A Trainable Neuromorphic Controller," *Journal of Robotic Systems,* vol. 5, no. 4, 1988, pp. 363-388.

[4] Chen, F. C., "Back-Propagation Neural Networks for Nonlinear Self-Tuning Adaptive Control," *IEEE Controls Systems Magazine*, April 1990, pp. 44-48.


2 & 3 Layer Nets with Random Training Set
TSS
Epoch Number
10
9.5
9
8.5
8
7.5
7
6.5
6
5.5
5
5
10
15
20
25
30
35
40


weight convergence, case 2
1200
1000
800
600
400
200
0
0
10
20
30
40
50
60


plant & net time responses, Case 2
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
x(1)
0
1
2
3
4
5
6
7
8
9
10
time


plant & net time responses, Case 2
x(3)
time
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
0
1
2
3
4
5
6
7
8
9
10