Identifying Nonlinear Dynamic Systems Using Neural Nets and Evolutionary Programming

by

R. G. Hutchins
Electrical and Computer Engineering
Code EC/Hu, Naval Postgraduate School
Monterey, CA 93943

Abstract

Nonlinear system behavior is not always well characterized by linearized system models, especially if the system is chaotic. This research studies the use of a neural network algorithm structure to model two nonlinear systems, a quadratic system and a chaotic system. An evolutionary programming approach is employed to train the neural nets so that the training process might better avoid selecting weighting parameters that represent a local minimum rather than a global minimum. This training approach is compared with the more standard back propagation technique.

Introduction

Neural nets are becoming widely used in the study of dynamic systems, both in the areas of system identification and control [1-4]. This research focuses on system identification using a neural network architecture to map inputs to outputs.

Actual system models examined here have state and output equations of the following form:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t)$$

where the input, $\mathbf{u}(t)$, and the output, $\mathbf{y}(t)$, are two dimensional, and the state, $\mathbf{x}(t)$, is three dimensional. The input and output matrices are given by:

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}^T, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Two functional forms for the function $f(\mathbf{x})$ are considered. The quadratic function is of the form:

$$f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \begin{bmatrix} \mathbf{x}^T \mathbf{A}_1 \mathbf{x} \\ \mathbf{x}^T \mathbf{A}_2 \mathbf{x} \\ \mathbf{x}^T \mathbf{A}_3 \mathbf{x} \end{bmatrix}, \text{ where}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & -2 \end{bmatrix}, \mathbf{A}_1 = \mathbf{A}_2 = -0.01 \times \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix},$$

$$\mathbf{A}_3 = -0.01 \times \begin{bmatrix} 5 & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 5 \end{bmatrix}$$

The chaotic function is given by:

$$\dot{x}_1 = 10(x_2 - x_1)$$

$$\dot{x}_2 = x_1(28 - x_3) - y$$

$$\dot{x}_3 = x_1 x_2 - (8/3)x_3$$

These last equations are the Lorenz equations for the attractor of the same name. The two-dimensional output trajectory of the Lorenz attractor for a 30 second simulation is given in Figure 1. The distinctive butterfly pattern is clearly discerned.

The input vector, $\mathbf{u}(t)$, used to generate the figure consists of two piecewise constant functions of time, where the amplitude sequence for each function is an independent sequence of pseudo random variables chosen

from the standard normal distribution. The time over which $\mathbf{u}(t)$ is constant is 0.1 second. This same input function is also used to generate the data for the quadratic system.

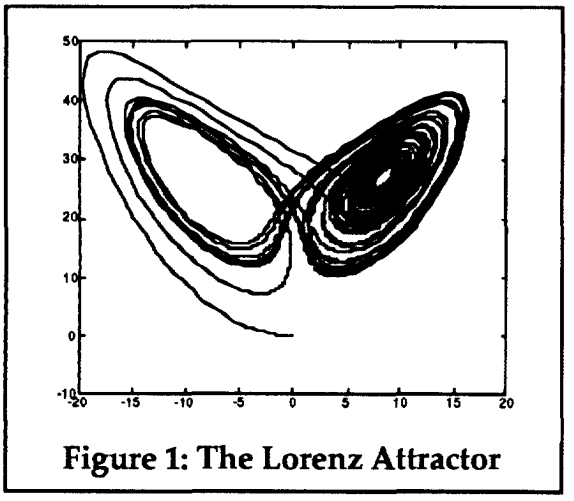

Figure 1: The Lorenz Attractor

Neural Net Topology

Inputs for the system identification problem consist of current and past input vectors, as well as current and past output vectors for the corresponding sampled data system, where the object is to determine the output one time step in the future [2]. A neural network is trained on the measured input-output data for a realization of the actual system over a set of measured sampling times. The sampling rate used throughout is 10 samples per second.

The neural network topology used here is that of a multilayer perceptron [2,4]. This topology is depicted in Figure 2. The input retina at sample k, $\mathbf{r_k}$, consists of current and past input vectors, as well as current and past output vectors. Following some preliminary tests, input retinas of dimension 20 were the main focus of study. The single hidden layer contains 20 nodes (also the result of preliminary tests) and uses the hyperbolic tangent function, tanh, to obtain a differentiable sigmoidal function with asymptotes at -1 and 1. The output layer is a purely linear function of the hidden nodes. This topology does not include bias terms in the hidden or output layers because the underlying systems, both quadratic and chaotic, are zero when all past outputs and inputs are zero.

Neural Net Training

Learning algorithms examined here are back propagation [2,4] and evolutionary programming [5]. Evolutionary programming is an optimization technique designed to inhibit the selection of a local optimal set of parameter values in place of the true globally optimal set for a given problem. Here, the weights and biases involved in the neural network constitute the parameters to be chosen optimally.

The specific form of the evolutionary programming learning algorithm used here is one in which there are n_1 parents in each generation and each parent generates n_2+1 children across a generation. A parent or child is characterized by the particular values for the weighting matrices associated with it. A child is generated from a parent using the equations:

$$\mathbf{W}_1^{child}(i,j) = \mathbf{W}_1^{parent}(i,j) + N(0,\sigma)$$

$$\mathbf{W}_2^{child}(i,j) = \mathbf{W}_2^{parent}(i,j) + N(0,\sigma)$$

where $N(0,\sigma)$ is a zero mean, normally distributed random variable with variance σ.

The variance σ was chosen by experiment. The learning rate (step size used in gradient descent during back propagation training) was chosen to be 10e-4. Larger values actually caused divergence in training for the chaotic system, while smaller values caused reduced learning rates in both systems. Values of σ on the order of 1 tended to produce almost no training effect during very lengthy evolutionary programming training sessions. Values of σ on the order of the back propagation learning rate of 10e-4 also tended to improve very slowly. Values of σ on the order of 10 to 100 times the back propagation learning rate worked best.

Also associated with each child is an error value representing the SSE associated with the corresponding weighting matrices when applied to the training set. The additional child in the n_2+1 children is actually a replication of the parent, so parent parameter

values can live across generations if they are superior. When all children from all parents have been created, the n_1 with the lowest SSE are chosen to become parents for the subsequent generation. Here $n_1=n_2=10$.

Results

Figure 3 shows the results of 1000 training epochs using back propagation in training the net to identify the quadratic system. Figure 4 shows the final 300 epochs of this training. These data depict the volatile nature of this training algorithm, which is clearly not uniformly decreasing. The minimum SSE attained during this training was approximately 18.

The training for the evolutionary programming algorithm does produce a uniform decrease in SSE across generations. This was accomplished by mixing training rates: half the children in each generation were generated using a σ value of 100 times the learning rate used in back propagation training, while the other half used a σ only 10 times this learning rate. Lower and higher values of σ trained substantially more slowly. Overall, however, this training is much slower than using back propagation, and the final SSE is nearly 50% higher after longer training.

A final technique which combined evolutionary programming with back-propagation was also studied. In this implementation, each newly generated child is trained for 100 iterations using back propagation. The results of training in this fashion across 7 generations is presented in Figure 7. Here, the first two generations achieve approximately the same minimum SSE as does back propagation training, while subsequent generations are able to break out of the local minimum found by back propagation and achieve a reduction in SSE by a factor of 3. However, this achievement was bought at the expense of an order of magnitude increase in training time. The final trajectories for the true and neural network system dynamics are presented in Figure 8.

These techniques were then transferred to the study of the chaotic system where reasonable results were not forth-coming. Again, the combined training technique achieved a substantial performance gain (a 50% reduction in SSE) over that achieved by back-propagation alone. However, in this case the final results were merely half as awful (SSE of 10e4). Alternative network topologies, larger input retina and other modifications have yet to provide any improvement over this dismal showing.

REFERENCES

[1] K. S. Narendra and S. Mukhopadhyay, "Intelligent Control Using Neural Networks," *IEEE Control Systems Magazine*, Vol. 12, No. 2, pp. 11-18, 1992.

[2] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, Vol. 1, pp. 4-27, 1990.

[3] A. Stubberud, H. Wabgaonkar, and S. Stubberud, "A Neural-Network-Based System Identification Technique," *Proceedings of the 30th IEEE Conference on Decision and Control,* Vol. 1, pp. 869-870, December 1991.

[4] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, New York, 1990.

[5] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving Neural Networks," *Biological Cybernetics*, Vol. 63, pp. 487-493, 1990.

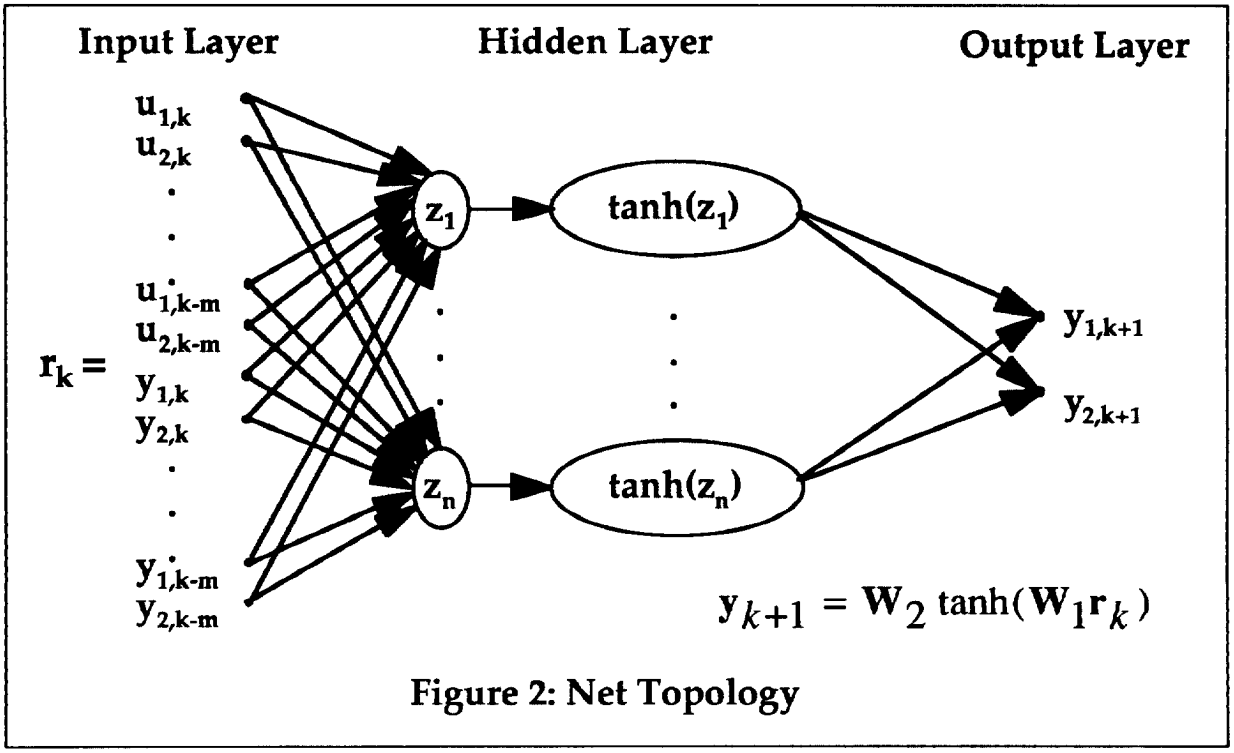

Figure 2: Net Topology

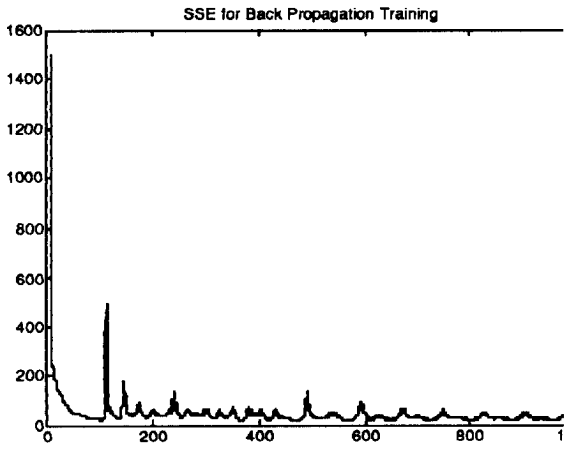

Figure 3: Backprop Training

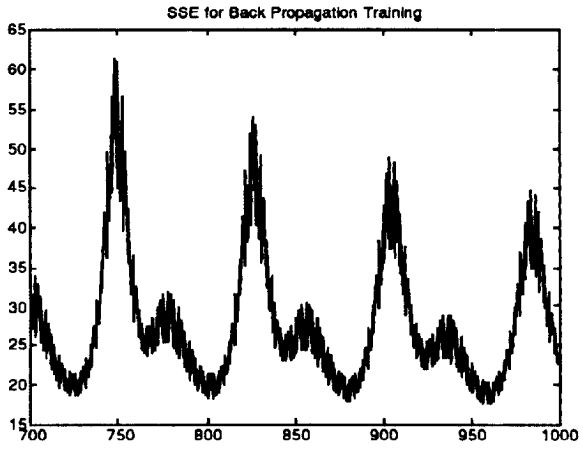

Figure 4: End ofBackprop Training

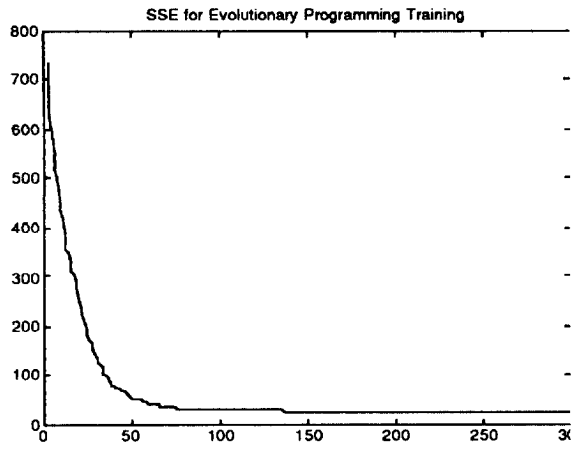

Figure 5: EP Training

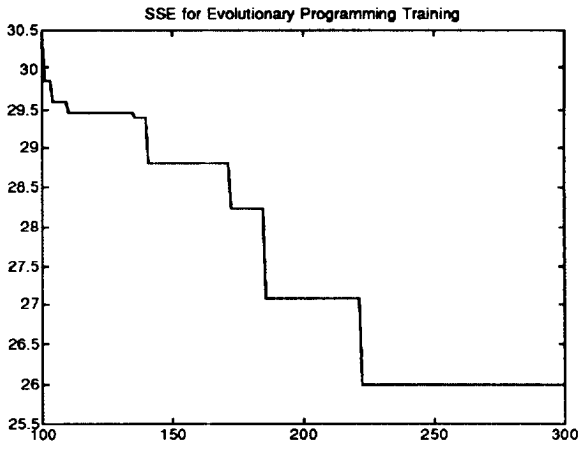

Figure 6: End of EP Training

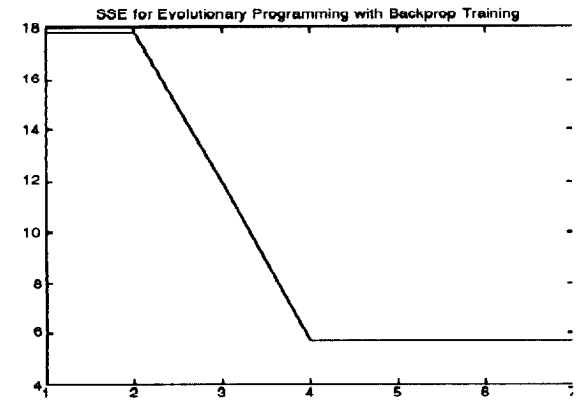

Figure 7: Combined Training

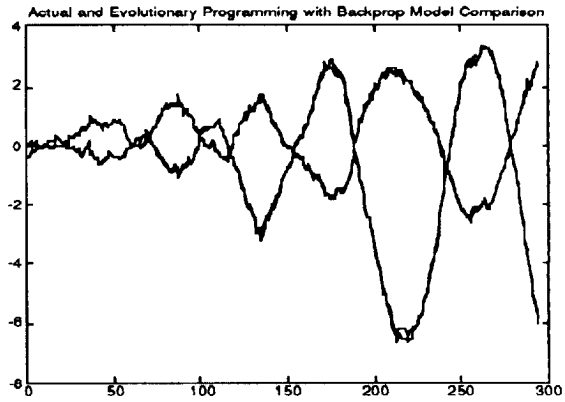

Figure 8: Final Trajectories