LA-UR -87-2662

LA-UR--87-2662

DE88 006479

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

TITLE: NONLINEAR SIGNAL PROCESSING USING NEURAL NETWORKS: PREDICTION AND SYSTEM MODELLING

AUTHOR(S): A. Lapedes, T-DOT
R. Farber, T-DOT

SUBMITTED TO: IEEE-Neural Networks,
San Diego, CA
June 1987

DISCLAIMER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

July 1987

NONLINEAR SIGNAL PROCESSING USING NEURAL NETWORKS: PREDICTION AND SYSTEM MODELLING

Alan Lapedes
Robert Farber
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87545

A B S T R A C T

The backpropagation learning algorithm for neural networks is developed into a formalism for nonlinear signal processing. We illustrate the method by selecting two common topics in signal processing, prediction and system modelling, and show that nonlinear applications can be handled extremely well by using neural networks. The formalism is a natural, nonlinear extension of the linear Least Mean Squares algorithm commonly used in adaptive signal processing. Simulations are presented that document the additional performance achieved by using nonlinear neural networks. First, we demonstrate that the formalism may be used to predict points in a highly chaotic time series with orders of magnitude increase in accuracy over conventional methods including the Linear Predictive Method and the Gabor-Volterra-Weiner Polynomial Method. Deterministic chaos is thought to be involved in many physical situations including the onset of turbulence in fluids, chemical reactions and plasma physics. Secondly, we demonstrate the use of the formalism in nonlinear system modelling by providing a graphic example in which it is clear that the neural network has accurately modelled the nonlinear transfer function. It is interesting to note that the formalism provides explicit, analytic, global, approximations to the nonlinear maps underlying the various time series. Furthermore, the neural net seems to be extremely parsimonious in its requirements for data points from the time series. We show that the neural net is able to perform well because it globally approximates the relevant maps by performing a kind of generalized mode decomposition of the maps. Use of trigonometric sin's, instead of the usual sigmoids, for the neural net transfer function leads to a type of generalized Fourier analysis. One may also view the approximation procedure of the neural net in relation to a spline fitting method, which in many instances is known to be preferrable to a simple polynomial fitting method. The specific simulations that are presented are intended to illustrate some of the capabilities of the formalism and are not thought to exhaust the range of application.

I. Introduction

Adaptive signal processing is a topic of considerable practical interest. A common approach to signal processing uses predominantly linear analysis, which not surprisingly, does not perform as well as desired when used to process signals emitted by a nonlinear system. We show that a natural extension of linear methods into the nonlinear domain is provided by the nonlinear neural net learning algorithm called "back propagation."[1] As explained in Section II, our technique stems from the relatively simple idea of inserting "hidden units" (a layer of nonlinear neuron-like elements) into the linear "Adaline" adaptation framework of Widrow-Hoff,[2],[3] and then using back propagation to control the weights. The input and output elements are kept as linear elements in order to provide an extended dynamic range. The original "Adaline" or "Least Mean Square"[2] adaptation rule (in wide use in modern signal processing) may be thought of as a learning rule for a totally linear neural network. There is then a logical progression to the nonlinear networks, that we use, which may be summarized as follows:

Adaline (Least Mean Square) → Widrow-Hoff → Perceptron → Backpropagation

Back propagation so far, has mainly been used in situations where the inputs and outputs of the network are nonlinear and achieve binary values. That is, it has been predominantly used in processing symbolic information.[1] Letting the input and output neurons be linear elements extends the dynamic range, and allows processing of real valued inputs/outputs such as occur in signal processing applications. The "hidden" nonlinear neurons use a continuous, nonlinear (and nonpolynomial)activation function. It is the ability to control nonlinearity in the neural net that allows prediction in chaotic time series with an

accuracy far exceeding conventional methods. Chaotic time series are emitted by deterministic nonlinear systems and are sufficiently complicated that they appear to be "random" time series. However, because there is an underlying deterministic map that generates the series there is a closer analogy to pseudo random number generators than to stochastic randomness. Nonlinear neural nets are able to perform well because they extract, and very accurately approximate, these underlying maps. Deterministic chaos has been implicated in a large number of physical situations including the onset of turbulence in fluids;[4], [5] chemical reactions,[6] lasers,[7] and plasma physics[8] to name but a few. Furthermore, chaotic systems can also display the full range of less complicated nonlinear behavior (e.g. attraction to a fixed point and limit cycles) if various parameters in the system are changed. They therefore provide an excellent test bed in which to investigate nonlinear signal processing techniques. We have selected two chaotic time series: one generated by an explicit nonlinear iterated map (the logistic or Feigenbaum map) and another generated by a nonlinear, differential delay equation (the Mackey-Glass equation). Prediction using nonlinear neural networks exceeds conventional methods by orders of magnitude in accuracy.

In addition to possible applications, the domain of real valued signal processing also provides a nice setting in which to investigate properties of the back propagation algorithm itself. One property, the ability to form limited generalizations, has frequently been tested with "symbolic" (binary) input/output pairs.[1],[9] Unfortunately, it has been difficult to obtain a really clean example of this ability to generalize. On the other hand, Section IV provides an example in nonlinear system modelling in which it is clear that the neural net has inferred from a finite data set the correct algorithm that transforms input to output. The somewhat mysterious ability of neural networks to "deduce" algorithms and to "gen-

eralize" is shown to be nothing more than real valued function interpolation when viewed in the context of signal processing. The modeling example we chose to analyze in Section IV is a "plant" (to use control system termonology) that implements $x(t) \rightarrow \dot{x}^2(t)$ (see Figure 1). Here $x(t)$ is an arbitrary input wave form and the network has to learn to output $\dot{x}^2(t)$ by learning on a training set consisting of input/output pairs that are samples at discrete times. We trained the network on a set of input/output pairs from a specific, broadband $x(t)$, and used back propagation to adjust the network weights. If the net correctly inferred the algorithm $x \rightarrow \dot{x}^2(t)$ then input (after training) of a different, arbitrary wave form $x(t)$ should result in the correct $\dot{x}^2$ of the new signal. This is the case. Furthermore, if the input to the network is $x(t)$ and $x(t - \Delta t)$, then the network output should be an approximation to $[x(t) - x(t - \Delta t]^2/\Delta t^2$. Thus, a graph of the output, $\dot{x}^2$, versus the inputs $x(t)$, $x(t - \Delta t)$ should be an approximation parabolic trough. We plot the output of the neural net versus $x(t)$, $x(t - \Delta t)$ (see Figures 11, 12) and the resultant parabolic trough is explicit graphic verification that the network has indeed learned the correct algorithm from a finite set of input/output pairs.

A competing approach to processing nonlinear signals would be to form polynomials in the data terms (the polynomials providing nonlinearity in the data) and to adjust the linear weight factor coefficient for each polynomial term using the Least Mean Square algorithm. This approach was advocated by Gabor and is related to the Volterra-Weiner expansion[10] of nonlinear systems. It has the advantage that polynomial nonlinearities may be modelled exactly, and that one is always assured of finding a global minimum to the Least Mean Square problem. Disadvantages, however, are considerable. First of all, nonpolynomial nonlinearities must be modelled by polynomials, which is widely known to be a undesirable procedure due to

the rapid oscillation of polynomials. Secondly, one has an explosion in the number of the polynomial coefficients as the system size, or order of the polynomial, is increased. Finally, a polynomial approximation is wildly unstable under iteration. Iteration, as we demonstrate in Section III, is the key to achieving accurate predictions over long times. Acceptable accuracy may be achieved by polynomial methods over short times, which is clearly a much less interesting situation in comparison to long-term prediction. The nonlinear neural net is orders of magnitude more accurate for long-term prediction. Finally, we demonstrate in Sections III and IV that if the relevant system nonlinearity is indeed a polynomial, then very good approximations to the polynomial may be achieved by using nonlinear nonpolynomial neural nets. We therefore feel that the nonlinear neural net method presented here has considerable advantages in both accuracy, and flexibility, over the more conventional methods.

The reason that the neural net formalism for signal processing works well seems to be related to the fact that the network is performing a kind of generalized mode decomposition of the underlying maps. Changing the neurons's transfer function from sigmoids to sin$ changes the analysis to a generalized Fourier analysis. Other nonlinear, neural transfer functions are also possible and should be choosen to make a best match to the problem at hand. Another interpretation (Section V) is related to spline fitting procedures. The difference between mode decompoposition vs simple polynomial fitting, distinguishes neural networks from the Gabor Weiner, Volterra polynomial analysis[10] of nonlinear systems.

The examples of prediction and nonlinear system modelling were choosen somewhat arbitrarily as a means to illustrate the capabilities of the formalism. The success achieved in these examples might reasonably be taken as an indication that further development of these methods could have

wider applicability. Results of experiments on specific applications will be reported elsewhere.

II. The Linear Predictive Method and Back Propagation

Prediction is a useful ability in signal processing that also has application in many other areas such as data compression. A common method of prediction in signal processing is the Linear Predictive Method.[11] In this approach one uses the values of a continuous signal, $x(t)$, at a set of discrete times in the past, to predict $x(t)$ at a point in the future. For example, one might use three values in the past, $x(t)$, $x(t - \Delta)$, $x(t - 2\Delta)$ to predict a value that is some time in the future, perhaps $x(t + 2\Delta)$. Δ is a time increment. The predicted value is a linearly weighted sum of the delayed (past) $x(t)$ values. Representing this algorithm in a diagram (Fig. 2) makes it clear that one can view this method as a linear, feedforward, neural net with no "hidden units." Each line in the figure linearly weights the corresponding input so that the output is a linearly weighted sum of input values. (See Figure 2.)

The weight values, T_{ij} are determined in the Linear Predictive Method by training the system using a set of discrete time samples from a segment of known signal. Labelling the neurons from $i = 0$ to 3 yields (for Figure 2)

$$X_3(t + 2\Delta) = T_{30}X_0(t) + T_{31}X_1(t - \Delta) + T_{32}X_2(t - 2\Delta) + I_3 \qquad (1)$$

or more generally, if there is more than one output,

$$X_i = \sum_j T_{ij}X_j + I_i \qquad (2)$$

For nonlinear neural nets I_i is referred to as "threshold" and we will continue to use that notation even for linear networks. If we label discrete times in the training set as t_p, then T_{ij} may be determined

by minimizing the mean square error, E

$$E = \sum_{ip} [X_i(t_p,\Delta) - \sum_j T_{ij} X_j(t_p,\Delta) - I_i]^2 \qquad (3)$$

In the above, i ranges over the output units, which for the example of equation (1) contains just one term for i = 3. p indexes the discrete times in the training set. This is a usual, linear, least mean squares problem that may be solved, for example, by steepest descents. Steepest descents is implemented by successively changing T_{ij} by an amount ΔT_{ij} where

$$\Delta T_{ij} = - \varepsilon \frac{\partial E}{\partial T_{ij}} \qquad (4)$$

where ε is a small number. I_i is determined in a similar manner. The form of Eqns. (1) and (4) show that the commonly used Linear Predictive method for signal processing is the trivial (i.e. linear) limit of the back propagation algorithm for nonlinear neural networks.

The Gabor Polynomial Predictive Method[10] is a straightforward extension of these ideas. In this formalism, each input neuron represents one term in a polynomial expansion of the data. For example, if the polynomial is specified to be second order, then there will be three first order terms (already represented) and an additional six neurons representing the six possible cross terms of of $X_0(t)$, $X_1(t - \Delta)$, $X_2(t - 2\Delta)$. The weights appear as linear coefficients of these cross terms. Therefore, the Linear Least Mean Square algorithm works for the Gabor Polynomial Method with virtually no change in implementation. One disadvantage of the polynomial method is already clear. If there are d data items to be combined into a general m $\underline{\text{th}}$ order polynomial (for the above example d = 3, m = 2) then the number of terms grows like $(m + d)!/m!d!$, which explodes exponentially as either d or m gets large.

Given Figure 2, and a familiarity with the Backpropagation Algorithm it is natural to insert a layer of nonlinear "hidden units" and to use back propagation to control the weights (Fig. 3). "Hidden units" are elements that do not have a linear neural transfer function. Instead, the neural transfer function is sigmoidal as shown in Figure 4. Hidden units greatly extend the power of neural networks and can be controlled with the back propagation algorithm. In addition to the weights, T_{ij}, there are also the thresholds, I_i, that shift the position of the sigmoid. Thus, if X_i are inputs to a hidden unit, the output of the unit is not merely the linearly weighted sum, $\Sigma_j T_{ij}X_j + I_i$ but the output of the sigmoidal transfer function $g(\Sigma_j T_{ij}X_j + I_i)$. The I_i shifts the sigmoid to the left or right. Training the system involves minimizing an E function which is now somewhat more complicated than Eqn. 3 because of the nonlinear g( ) functions. Nevertheless, a steepest descents algorithm is often used in back propagation to minimize E.

Back propagation may thought of as a particular, nonlinear, least squares algorithm. It may also be thought of as a generalization of the Perceptron formalism where the discontinuous, Heaviside step function used for the Perceptron's neural transfer function is smoothed into the continuous, sigmoidal transfer function. It is a natural, nonlinear, extension of the linear nets commonly used in adaptive signal processing. Use of the chain rule in computing derivatives of E provides a useful interpretation to the minimization process and allows an easy generalization to multilayers of nonlinear hidden units.[1] For one or more output units one minimizes

$$E = \sum_{pi} [targ_i^{(p)} - o_i^{(p)}]^2 \qquad (5)$$

where the $\text{targ}_i^{(p)}$ are the specified target outputs for the p^{th} input pattern, and $O_i^{(p)}$ is the actual output of the network's i^{th} output unit given the p^{th} input pattern and the present set of weight, T_{ij}, I_i. E is to be considered as a function of T_{ij} and I_i. For the linear predictive net considered earlier, expression (5) collapses to a simple form. For this case, the sum over i contains just one term i = 3, while the target output $\text{targ}_i^{(p)}$ would be $\text{targ}_i^{(p)} = X_3(t_p + 2\Delta)$, and $O_3^{(p)}$ would be $O_3^{(p)} = \sum_j T_{ij}X_j(t_p) + I_i$, i.e. a linear function of the inputs to unit 3. If a nonlinear layer of hidden neurons were inserted into Figure 2, then $O_3^{(p)}$ would also contain contributions from the outputs of the hidden layer. Because the hidden layer has the nonlinear transfer function g( ), the output of the hidden layer is now a nonlinear function of its inputs, and E in Eqn. (5) becomes the square of a nonlinear function of the weights because the hidden layer outputs feed into the topmost output layer.

Steepest descents is performed in the normal fashion by letting

$$\Delta T_{ij} = - \varepsilon \frac{\partial E}{\partial T_{ij}} \tag{6}$$

Defining some intermediate quantities simplifies the partial derivatives in Eqn. 6. Let

$$\text{net}_i = \sum_j T_{ij}O_j + I_i \tag{7}$$

be the net input to unit i from the outputs, O_j, of other neurons in previous layers connected to neuron i. The output of neuron i will then be

$$O_i = g(\text{net}_i) = g(\sum_j T_{ij}O_j + I_i) \tag{8}$$

If one introduces another quantity δ_i defined as

$$\delta_i = - \frac{\partial E}{\partial \text{net}_i} \tag{9}$$

then one obtains:

$$\frac{\partial E}{\partial T_{ij}} = - \delta_i O_j \tag{10}$$

Thus gradient descent is implemented by making changes in T_{ij} by the amount ΔT_{ij}, where

$$\Delta T_{ij} = - \varepsilon \frac{\partial E}{\partial T_{ij}} = \varepsilon \delta_i O_j \tag{11}$$

where ε is a small number. δ_i may be computed by the chain rule. If unit i is an output unit then δ_i becomes:

$$\delta_i = \sum_p (\text{targ}_i^{(p)} - O_i^{(p)})\, g'(\text{net}_i) \tag{12}$$

where $g'(\ )$ is the derivative of $g(x)$ with respect to x. If i is not an output unit, then δ_i may be computed recursively starting at the topmost layer (which is the output layer):

$$\delta_i = g'(\text{net}_i) \sum_j T_{ji} \delta_j \tag{13}$$

Equations 11, 12, and 13 define the backpropagation steepest descents procedure for nonlinear neural nets as outlined in Reference (1). The name "back propagation" arises from Eqn. 13 where an error signal is propagated back from the output neurons to other neurons of the network.

III. Prediction

To illustrate the use of the nonlinear neural net formalism, we choose to predict in such a complicated time series that it is "random" and ergodic. The series is generated by iterating the classic logistic, or Feigenbaum, map[12]

$$x(t + 1) = 4bx(t)[1 - x(t)] \tag{14}$$

where b is set to 1.0. This map is not known to the investigator, of course. He has only a set of samples from the time series and is required to use these samples to perform prediction. This iterated map produces an ergodic, chaotic time series if b is choosen equal to 1. (Other values of b lead to fixed points, limit cycles or chaos as documented in Reference 12.) Although the time series passes virtually every test for randomness, it is generated by Eqn. 14 and therefore may be thought of in analogy to a pseudo random number series. It is widely conjectured that important instances of randomness in Nature[4, 5, 6, 7, 8] (e.g. the onset of turbulence) are due to the deterministic chaotic behavior produced by similar nonlinear iterated maps.

Because the map, Eqn. 14, is polynomial, it is clear that the conventional Gabor method [10] would also work very well if one used a second order polynomial. We chose this simple nonlinear problem to introduce the procedures we will be using, and to demonstrate that nonlinear neural nets can very accurately model polynomials, in addition to more general nonlinearities. Also, we will return to this example in Section V, where we are able to graphically demonstrate how the nonlinear neural net adds up sigmoidal nonlinearities to approximate quite arbitrary functions. A much more complicated example will be considered shortly in which polynomial methods are clearly inferior to nonlinear neural net methods.

Our goal is to use the back propagation algorithm to adjust the T_{ij}, I_i, enabling a prediction of the next point x(t + 1) in this "random" series given the present point x(t). We chose a network architecture with 5 hidden units as illustrated in Figure 5 and trained the system, using back propagation, on 1000 sets of (x(t),x(t + 1)) pairs. The output unit was a linear unit. The trained network was then used to predict one time step into the future for 500 additional points. We always assume that the "past" data needed to perform the prediction, in this case x(t), is

obtained from observing the actual time series. Thus one makes a pre-diction, observes what actually occurred, and uses the actual, observed value to make the next prediction. The normalized root mean square prediction error was 1.4×10^{-4}. "Normalized" means that the root mean square deviation of the predicted values from the actual values is divided by the standard deviation of the data. We will refer to this normalized quantity as the "index." This measure is independent of the dynamic range of x(t). Because the series is "random" and ergodic, the only way that the net can perform so well is if in the training procedure it learns to very closely approximate the underlying nonlinear map, Eqn. 14, that generates the series upon iteration. Recall that the network sees only "random" numbers and has no apriori knowledge that a mapping exists between these numbers.

In this situation, the map is simple (polynomial) and prediction is not done very far into the future. In Section V, we explicitly show how the neural net approximated the map of Eqn. 14 using data from the time series. This simple quadratic map could also have been exactly recovered from the time series by using a linear network and including multipliers at the tap lines to form polynomials in the data. The E function to be minimized would still be quadratic in the weights although the data terms would now be a general polynomial including powers beyond quadratic. This is the method of Gabor, Weiner, Volterra[10]. Although prediction can be improved over that achieved by the normal Linear Predictive Method, (for this simple example, the polynomial map could be recovered exactly) in general this multiplicative method will be inferior in predictive ability to that provided by nonlinear neural nets (in the following more complicated example it is worse by orders of magnitude). Furthermore, the multiplicative method suffers from an explosion in the number of weights as the number of tap delays and the order of the polynomial is increased.

We should also point out that we chose one input neuron in the network architecture, Figure 5, solely for illustrative purposes. Adding more input neurons (i.e. choosing additional delayed values from the time series for input) actually increases the predictive accuracy, at least for the case of 3 input neurons that we tested.

A second, much more complicated test of predictive ability, was suggested to us by D. Farmer and J. Sidorovitch.[13] In this example the time series is generated by a delay differential equation

$$\frac{dx(t)}{dt} = \frac{ax(t - \tau)}{1 + x^{10}(t - \tau)} - bx(t) \qquad (15)$$

that was first investigated by Mackey and Glass.[14] Keeping the parameters a and b fixed at a = .2 and b = .1 leaves τ as the only adjustable parameter. As τ is varied the system exhibits fixedpoint, limit cycle, or chaotic behavior. Choosing τ = 17 yields chaotic behavior, and a strange attractor,[15] with fractal dimension approximately 2.1. τ = 30 yields a strange attractor with the fractal dimension approximately 3.5. Higher values of τ yield higher dimensional chaos. Note that because of the delay, $x(t - \tau)$, the phase space of this system is infinite dimensional. However, as time progresses the system collapses onto the low dimensional strange attractor. Other infinite dimensional chaotic systems, such as nonlinear partial differential equations, also display collapse onto low dimensional attractors. Thus, the Mackey-Glass equation (15) exhibits in the simpler setting of nonlinear, differential equations behavior that occurs in much more complicated systems such as nonlinear partial differential equations. A detailed analysis of the chaotic properties of equation (15) may be found in Reference (16). At τ = 17, x(t) appears to be quasiperiodic and the power spectrum is broadband with numerous spikes

due to the quasiperiodicity. At $\tau = 30$ X(t) is even more irregular. Figure 6 shows a plot of x(t) vs t for a time span of 500 time steps for both $\tau = 17$ and $\tau = 30$. A constant function was used as the initial values, and transients were allowed to die out before the plot was started.

Packard et al.,[17] have demonstrated that an attractor may be reconstructed from a time series by using a set of time delayed samples of the series. If Δ is a time delay, and m is an integer, then one may write for points on the attractor

$$x(t + P) = f(x(t), x(t - \Delta), x(t - 2\Delta) \ldots x(t - m\Delta)) \qquad (16)$$

where P is a prediction time into the future and f( ) is a map. This may be viewed as an m + 1 dimensional surface. Thus, the "embedding dimension" d_E, is defined to be m + 1. Takens[18] has proved that a least upper bound exists for which f( ) will be a smooth map. If the dimension of the attractor is defined to be, d_A, then one needs an embedding dimension less than or equal to $2d_A + 1$, i.e.:[18]

$$d_E \leq 2d_A + 1 \qquad (17)$$

A minimal requirement is that $d_E \geq d_A$. It is perhaps surprising that a smooth functional form, such as Eqn. 16, relates values in a time series generated by complicated nonlinear differential equations such as Eqn. 15. That such mapings exist is a consequence of Takens theorem,[18] however the theorem provides no information on the form that f( ) may take. We will show below that the neural net uses data from the time series to provide an explicit, analytical, expression that globally approximates f( ) to a sufficient degree to be able to perform prediction using equation (16) with an accuracy that exceeds the conventional Linear Predictive method and the Gabor, Weiner, Volterra method by orders of magnitude.

We will now test the predictive accuracy of the nonlinear neural net and compare it to the conventional methods of Linear Prediction[2],[3] and Gabor Polynomial Prediction[10]. The test will be performed twice, once using a time series generated by the Mackey-Glass equation (15) at $\tau = 17$ (fractal dimension = 2.1), and once using the Mackey-Glass equation at $\tau = 30$ (fractal dimension = 3.5). First consider the $\tau = 17$ time series shown in Fig. 6a. Using Eqn. (17) we select an embedding dimension, d_E, equal to 4. This specifies m in Eqn. (16) to be m = 3. It now remains to chose Δ and P. To facilitate later comparison to an alternative predictive method of Farmer et al.[13], we choose $\Delta = 6$. These choices of m and Δ imply that a prediction made P time steps into the future past the last observed point x(t) will be made using observed data at times: x(t), x(t - 6), x(t -12), and x(t -18). There are therefore four inputs to the nonlinear neural nets, representing these values of x(t), and one linear output element representing the value x(t + P). We chose 20 hidden units arranged in a two layer architecture. Therefore, the architecture of the network appears as Fig. 7. Each neuron in Fig. 7 is connected to all the neurons in the directly previous layer. This architecture was choosen rather arbitrarily and seemed to yield quite acceptable performance. Other architectures gave comparable performance.

We now need to choose the prediction time P. It is desirable to test the predictive accuracy as a function of how far the prediction is made into the future, so we will choose several values of P. For any given P there are two ways that a prediction P time steps into the future past the last observed data point may be made. The first way is to train a separate network for each choice of P. For example, if P = 6 then one may train a network using a set of samples: $x(t_i)$, $x(t_i - 6)$, $x(t_i - 12)$, $x(t_i - 18)$ on the four inputs, and $x(t_i + 6)$ on the single output. This network, after training, will then map any future set of $x^0(t_i)$, $x^0(t_i - 6)$,

$x^o(t_i - 12)$, $x^o(t_i - 18)$ (where superscript "o" indicates observed values) into the set $x(t_i + 6)$. In this method, one assumes that the data needed to predict at, say, t = 1,000 is the observed values of the time series at $x^o(994)$, $x^o(988)$, $x^o(982)$, $x^o(976)$. The network might have been trained, for example, on data taken from $t < 0$. A prediction at any arbitrary time, t, in the future is made using the last four observed data points. Thus, no matter how far ahead one is predicting in the time series (e.g. t = 1,000), the prediction is never more than P time steps (in this case P = 6) past the last observed data point. If, in the example above, one wished to predict 12 time steps into the future past the last observed point, then a second network would be trained using $x(t_i + 12)$ on the output neuron. This second network would always predict a value 12 time steps past the last observed point.

To see how predictive accuracy degrades with increasing P we trained 8 separate networks to make predictions at P = 6, 12, 24, 36, 48, 60, 72, 84, and 100 time steps past the last observed point. We computed the normalized root mean square index of accuracy (index = (root mean square predictive accuracy)/(standard deviation of the data)) for 500 predictions and plotted the results, for $\tau = 17$ data, in Fig. 8a. We did this for the nonlinear neural net, the Linear Predictive method, and the Gabor Polynomial method. The polynomial order was choosen to be equal to 6, yielding roughly the same number of polynomial coefficients as weights in the nonlinear neural net. This was also done for the second ($\tau = 30$) time series and these results are plotted in Fig. 8b. The embedding dimension in this case ($\tau = 30$) was choosen to be 6. The Linear Predictive method needed 2,000 data points in training to achieve any reasonable accuracy, while the Gabor method and the nonlinear neural net method seemed to do

well with 500 data points. It may be seen from Figures 8a, and 8b that the nonlinear neural net, using the predictive method just described, performs best. There is, however, a second way in which one can make predictions at various P values, in which the neural net performs far better than the method just described, and to which the alternative, conventional methods can offer little competition.

The second way to make predictions at various choices of P is to place previously predicted values on the input lines to bootstrap one's way to higher P values. That is, one iterates the mapping provided by the nonlinear neural net. For example, after training a network to predict at P = 6, one can feed the predicted values back into the inputs to predict at P = 12, 18, 24, ... etc. Thus, instead of training separate networks to predict at P = 12, 18, 24, ... etc. (as described above) one can simple iterate the mapping provided by the P = 6 network. There are tradeoffs implicit in this iterative approach. Because previously predicted values (made with some error) are used to make subsequent prediction, the errors get magnified upon iteration. Iterating a P = 6 net once, to form a P = 12 net will not magnify the errors very much. However, unless the P = 6 map was an extremely good approximation to the actual P = 6 map implied by Takens theorem, further iteration of the P = 6 map will soon get to be a dangerous procedure. It is intuitively clear that a P = 6 map is less irregular than say a P = 36 map (think of how the map changes upon iteration for the classic logistic map) and so it seems reasonable to believe that one does have a possibility of forming an extremely good approximation to the P = 6 map, and avoiding the danger just described. Ultimately, one will magnify the errors to an unacceptable degree, but this may not happen until the effective P is quite large, i.e. for a large number of iterations. To test this conjecture, we iterated the P = 6 map for both the nonlinear neural net and the Gabor polynomial method, and collected results on the index of accuracy in the same way as

before. These are the final two curves plotted in Fig. 8a ($\tau = 17$) and Fig. 8b ($\tau = 30$). It is immediately obvious that for the nonlinear neural net the danger just described was overcome, and that this procedure is a far better procedure for making predictions at large P. It is also readily apparent that the Polynomial method is wildly numerically unstable under this procedure (due to the errors getting grossly magnified by the high order polynomial terms). Figures 8a and 8b clearly show that the iterative, nonlinear neural net procedure is orders of magnitude more accurate than conventional procedures for large prediction times, P. Our choice of iterating the $P = 6$ map was an informed guess, it may well be that another choice of P would yeild even better results, although we have not investigated this. Further increased in accuracy may be obtained by increasing the number of hidden units in combination with increasing the number of training patterns.

A new predictive algorithm has recently been published by Farmer and Sidorovitch.[13] The number, and values of the delays for the nonlinear neural net method for the Mackey-Glass equation were choosen to agree with those used by Farmer et al. in testing their very recent and powerful Local Linear Predictive Method[13] (not to be confused with the conventional Linear Predictive Method described earlier). The accuracy of the Local Linear method and the nonlinear neural net method (perhaps best described as a global, nonlinear method, see Section V) are roughly comparable for this problem.[19] Increases in accuracy in one method over the other can be achieved by twiddling the respective algorithms, however the main conclusion is that both methods are orders of magnitude more accurate than conventional methods (including the global polynomial method of Gabor et al.[10] and the Linear Predictive Method,[2],[3] and indications are that both methods may be used to achieve the fundamental limits on predictive accuracy dictated by the nature of chaos.

It seems that the nonlinear neural net method will be very useful for

performing prediction in real time, and for other real time signal processing applications, such as adaptive control and system modelling. This is due to the natural mapping of neural nets onto parallel hardware,[20] with the resultant possibilities of training times on the order of microseconds. Prediction times (the time taken to make one prediction after being trained on the data) are already quite short, due to the simplicity of feedforward networks, however these times could, of course, also be reduced to microseconds, or so, if done in hardware. Furthermore, the nonlinear neural net method seems to be achieving roughly comparable accuracy to the Local Linear Method of Farmer et al.[13], [19] using only 500 points from the time series, whereas the Local Linear Method uses 10,000 - 20,000 points. Parsimony in the requirements for data points from a time series is a considerable advantage, as a large number of data points may be collected only through using very short sampling times, or else collecting data over a long time. In many applications the use of short sampling times, or long data collection times, is either undesirable (short time samples are more correllated, hence yielding less information) or infeasible (the data over a long time may be unavailable). To be fair, the nonlinear neural net method requires a longer run time for training in computer simulation (30 - 60 minutes on a Cray X-MP, compared to a few minutes on an X-MP, for the Mackey-Glass P = 72 example using the Local Linear Method). However, the neural net method also yields more information than the Local Linear Method as a result of training (see Section V) so it is not yet clear which method is faster (when simulated) in producing the same amount of information

IV. Nonlinear System Modelling

A second subject of considerable interest in signal processing is the system modelling problem. Here one wishes to construct a model for the transfer function of an unknown "plant" using only a finite data set of inputs, and associated outputs, of the plant. Applications include adap-

tive control among other topics. We choose a relatively simple nonlinear transfer function, $x \rightarrow \dot{x}^2$, which is depicted in Figure 1. The reason for choosing this example as an illustration of the method is that there is a nice graphical way to depict both the actual transfer function and the modelled transfer function. A polynomial Gabor, Weiner, Volterra[10] method would also work quite well for this simple example. We chose it to simply illustrate how a nonlinear neural net works, and not to demonstrate the relative effectiveness of the neural net method. This was already done in the previous section.

A neural network with 2 input units, two layers of hidden units and one output unit were trained on an I/0 data set generated by passing a relatively broadband input function through the block box of Figure 1. The network is shown in Figure 9. The input was a sum over twenty frequencies with random phases in the range [0,1].

$$x(t) = \frac{1}{N} \sum_{\ell=1}^{20} \sin(2\pi\ell t + \phi_\ell) \tag{18}$$

that was sampled at times separated by .001 between t = 0 and t = 1. The factor N is a normalizer that normalized the resultant output ($\dot{x}^2$) to a maximum value of 1.0. The normalization was performed soley for computational convenience. Inputs and outputs having arbitrary dynamic range are easily handled by a scaling agreement, as described in Appendix I. Training was accomplished by taking data points from Eqn. 18 over a time interval [0,1] and setting the inputs to be x(t) and x(t - .001), with the output set to be $\dot{x}^2$. Backpropagation was used to adjust the weights.

After training is complete, one should be able to input a new waveform, $\tilde{x}(t)$, and have $\tilde{\dot{x}}^2(t)$ emitted by the output unit of the neural network. We selected another wave form, similar to Eqn. 18, except with a different choice of random phases. The normalized rootmean square accuracy for the test waveform was .0476.

This example was choosen because it is possible to graphically demonstrate that the neural net actually learned the algorithm, $x(t) \rightarrow \dot{x}^2(t)$, from a finite data set of input/output pairs. First of all, we plot the actual algorithm, $x \rightarrow \dot{x}^2$, by graphing $\dot{x}^2$, vs $(x(t), x(t - .001))$. A finite difference approximation to $\dot{x}^2$ is

$$\dot{x}^2(t) = [x(t) - x(t - .001)]^2 / (.001)^2 \qquad (19)$$

which of course yields a parabolic trough if $Z = \dot{x}^2(t)$ is plotted vs $x(t)$ on the x axis, and $x(t - .001)$ on the y axis. Note that we are not using values for $x(t)$ and $x(t - .001)$ taken from a particular wave form to produce the plot. Instead, $x(t - .001)$ is considered to be an independent variable from $x(t)$, and we plot the functional dependence of $\dot{x}^2(t)$ on the two independent variables. Thus we imagine $x(t)$ to be an x axis variable, $x(t - .001)$ to be a y axis variable, and plot the graph $Z = (x - y)^2/(.001)^2$. The resultant graph ($Z = (x - y)^2/(.001)^2$ is shown in Figure 10. We have rotated the axes so that we are looking straight down the trough.

Next, we consider the algorithm that was learned by the neural network. After training, the output neuron value ($\dot{x}^2(t)$) is a well defined function of the values of the input neurons $x(t)$, $x(t - .001)$. The function is a complicated sum involving tanh( )'s (due to the sigmoidal transfer function of the hidden units), and certain coefficients, which are the actual values of T_{ij} and I_i that were determined by the training algorithm. This function is easily plotted in an analogous manner to the previous figure and is shown in Figure 11. Here again, $x(t)$ and $x(t - .001)$ are taken to be independent variables (i.e. not from a particular time series), and the value of the output neuron is plotted vs the values of the two input neurons in the same rotated coordinate system. Note that a parabolic trough does appear in Figure 11, although as one proceeds away

from the bottom of the trough the sides stop rising and flatten out. This flat region is irrelevant. The region of interest is the bottom of the trough, because it is only in this region that $x(t) \tilde{\sim} x(t - .001)$, indicating a sufficiently small sampling time to be able to approximate $\dot{x}^2(t)$ by the finite difference approximation. We depicted the "flattened region"as well solely for illustrative purposes. In Figure 12, we plot a "blow up" of the relevant region, $x \tilde{\sim} y$, over a range of x between -.1 and +.1 and similarly for y. It is therefore graphically clear that the network did learn the algorithm $x \rightarrow \dot{x}^2(t)$ because the region $x \tilde{\sim} y$, corcorresponding to $x(t) \tilde{\sim} x(t - .001)$ (which is always satisfied for continuous wave forms assuming .001 is a sufficiently small sampling interval) is indeed a parabolic trough. Furthermore, we numerically verify the accuracy of the network's algorithm by calculating the normalized root mean square error of the network in comparison to the output of the finite difference approximation for $\dot{x}^2$ for 10,000 data points evenly spaced over a square of $x\varepsilon[-.05, .05]$; $y\varepsilon[-.05, .05]$. The resultant value was .0403, which is a value of the match of the algorithms. Adding more input units (further delays) actually increases the accuracy, although of course we can't plot the output in 3 dimension.

We also note that the parabolic trough is not generated by some sort of Taylor's series for small input values. The sigmoidal transfer function used in constructing Figure 12 has only odd terms in its Taylor expansion and therefore Figure 12 was generated by adding together full sigmoidal functions with appropriate coefficients (T_{ij}) that are sufficient to closely approximate a parabolic trough in the region of interest. Exactly how a neural net may add together sigmoids to approximate essentially arbitrary functions is described more fully in Section V. This ability to approximate functions, polynomial or otherwise, is the reason for the success of nonlinear neural networks. A strictly linear network, trained

with the usual Least Mean Square Rule of adaptive signal processing, would only be able to produce planes (or hyperplanes, if one used more than two delays on the input lines) if graphed in a similar manner to Figure 12. A second order Gabor Polynomial method would, of course, do very well on this problem, but as seen in the previous section, it offers no competition in more complicated examples. Finally, we point out that the graphs in Figures 11 and 12, are a clear example of "generalization" by neural networks. After training on a finite data set the neural net was able to deduce the correct algorithm (compare the troughs of Figures 10 and 12) such that when new data is presented a correct output is given. We see that the somewhat mysterious ability of neural networks to "infer algorithms" and to perform "generalization" is nothing more than real valued function interpolation, at least in the context of signal processing.

IV. Mode Decomposition by Nonlinear Neural Networks

It is natural to ask "why does a neural net do so well at nonlinear signal processing?" We answer this question by analyzing the simplest prediction example, the logistic map and then remark on the system modelling problem. First, note that at the end of the training period, all the synaptic weights, T_{ij}, and thresholds, I_i, are specified numbers. Therefore the output neuron, which represents $x(t + 1)$, is a specified function of the input neurons $x(t)$. In particular, if we take the weights which led to the root mean square predictive accuracy (1.4×10^{-4}) quoted in Section III, then one obtains from these weights, and Figure (5), the formula

$$\begin{aligned} x(t+1) = &- 0.64g(-1.11x - .26) - 1.3g(2.22x - 1.71) \\ &- 2.285g(3.91x + 4.82) - 3.905g(2.46x - 3.05) \\ &+ 5.99g(1.68x + .60) \\ &+ (.31x - 2.04) \end{aligned} \tag{20}$$

where $g(x) = \frac{1}{2}(1 + \tanh x)$.

We may write this as

$$x(t + 1) = T(x(t)) \tag{21}$$

and compare this map to the actual map that produced the time series

$$x(t + 1) = f(x(t)) = 4x(t)[1 - x(t)] \tag{22}$$

A plot of Eqn. 22, i.e. plotting $x(t + 1)$ vs $x(t)$ yields (of course) a parabolic curve.

A plot of Eqn. (20), again plotting $x(t + 1)$ vs $x(t)$, yields a virtually identical curve over the range $0 \rightarrow 1$. Therefore Eqn. (20) is a very good approximation to the global map generating the time series. The reason this can occur may be seen by considering the sum of just two sigmoids with parameters a,b occurring in a similar fashion to the T_{ij} and I_i's in Eqn. (20)

$$S = a_1 g(b_1 x + c_1) + a_2 g(b_2 x + c_2) \tag{23}$$

Parameters b adjust the slope of the sigmoids, c adjust the shifts, while a adjusts the amplitudes. If, for example, a_1 is positive, while a_2 is negative, and if $c_1 \neq c_2$ then a "bump" will be formed when the sigmoids are added together as in Eqn. (20, 23). The function, Eqn. (22), or virtually any other $C^{(1)}$ function, may be approximated very well by appropriately forming and adding up "bumps." This is somewhat analogous to the method of splines[21] for approximating arbitrary functions. Splines are constructed in such a way to maximize smoothness. The smooth form of the tanh acts in an analogous fashion. Splines are, however, difficult to work with in many dimensions. A similar effect occurs in the Mackey-Glass example of Section II, however the four dimensional embedding dimension precludes plotting in three dimensions.

The way the sigmoids add to approximate a parabola for the logistic map may be seen in Fig. 13. Figure 13 should be read from left top down to bottom right. In each window, a term from Eqn. (20) is plotted as a dotted

line and the sum of the terms appearing in previous window are plotted as a solid line. Thus, window (1) initially shows term (1) as a solid line and term (2), to be added in, as a dotted line. Window (2) then shows term (1) + term (2) as a solid line and term (3), to be added in, as a dotted line, and so on. The final window shows the complete plot of the sum of all terms in equation 20. Each tick mark is one unit, so the final bump appearing between 0 and 1 in window (6) is the approximation to equation 22, which is a parabola with domain of $0 \rightarrow 1$ and range of $0 \rightarrow 1$. We continued the plots in all windows outside the valid range of $0 \rightarrow 1$ just to show more of each sigmoid that is added in. This range is actually irrelevant as far as using the network for prediction is concerned, and was shown for illustrative purposes only.

Adding together sigmoids to approximate a particular function is reminiscent of Walsh analysis. Walsh functions[22] are a complete set of functions, ranging from 0 to 1, that are made up out of step functions. A common technique of signal processing is mode decomposition of a function into sums of Walsh functions. A sigmoid may be viewed as a smoothed step function, and therefore approximating a function with sums of sigmoids seems somewhat similar to a mode decomposition of the function in terms of a Walsh function basis set.

This notion that the neural net performs a type of mode decomposition to approximate a function may be made clearer if one considers transfer functions of the nonlinear neurons involving trigonometric sin's instead of sigmoids.[23] This is, if we replace g(x) by

$$g(x) = \frac{1}{2}\,(1 + \sin(\beta x)),\ \beta = \text{a constant} \tag{24}$$

then g(x) is still an element of [0,1] and the back propagation algorithm will still work irrespective of the form chosen for the transfer function.

(The range condition on g(x) : g(x) ε [0,1] is not important, we chose it for convenience.)

If we now consider Eqn. 23, which is a generic form for output in terms of input for feed forward nets, then we see that the a_i's act like Fourier amplitudes, the b_i's like frequencies, and the c_i's like phases. Recall that the a_i's are the synaptic weights of the hidden to output layer, the b_i's are synaptic weights of the input to the hidden layer, and that the number of g functions (i.e. sin's) that occur in the sum is the number of hidden units in the hidden layer. Thus, training the neural net is, in essence, contructing a discrete Fourier series. However, in contrast to a normal Fourier decomposition in which the values of the frequencies are <u>fixed</u>, the net has the ability to <u>adjust</u> the values of the frequencies to obtain the minimum least mean square error. The number of adjustable frequencies is determined by the number of neurons in the hidden layer. It is, therefore, clear that specifying the number of hidden units specifies the number of frequencies available to the net, and that the net then adjusts the numerical value of these frequencies, and their amplitudes, and phases, to produce a best fit. Presumably, adding more hidden units, i.e. adding more adjustable frequencies, will improve accuracy. Because in conventional Fourier Mode Analysis only the amplitudes are adjusted, and the frequencies are fixed, we label the mode decomposition performed by the neural net "Generalized Fourier Decomposition," where "generalized" refers to the ability to adjust frequencies. Further generalizations are possible by considering multilayer networks and different expressions for the transfer function. We point out that using sin's often leads to numerical problems, and nonglobal minima, whereas sigmoids seemed to avoid such problems throughout all our extensive simulations.

It is worth emphasizing that the network is <u>not</u> approximating by a Mode decomposition the time series that was used in training. For chaotic

time series, there is a huge spread of frequencies and we have typically given the neural net a relative few number of hidden units. Instead, the neural net is approximating the underlying map that generates the time series.

In the logistic example the neural net, of course, approximated the parabolic map that was used to generate the time series. In the Mackey-Glass equation a nonlinear differential equation generated the time series. However, it is known on general grounds that a deterministic, nonlinear map underlays the time series, (see Section III) although the form of the map is not known. It is this map that was approximated by the neural net using data from the time series. The ability of neural nets to provide explicit formulae that are excellent approximations to the unknown maps implicit in chaotic time series may be of interest in the analysis of chaos and other nonlinear behavior. Finally, we note that the parabolic trough plots in Section IV are an example in two dimensions of this type of analysis.

VI. Summary

We have shown that the back propagation algorithm for nonlinear neural nets is a natural generalization of the widely used LMS rule or Linear Predictive Method for signal processing. Use of back propagation in the context of signal processing allows solution of nonlinear system modelling problems as well as excellent prediction on complicated, "random," time series. Predictive performance greatly exceeds all known (to us) conventional methods of prediction including Linear Prediction[2],[3], Global Polynomial[10] methods and is competitive with the new Local Linear Method.[13] In addition, the network provides an explicit, global, approximation to the underlying nonlinear map with minimal requirements for data points from the time series. Specific applications have not been discussed, however it seems that there are many. One may expect that other areas of signal processing, in addition to prediction and system modelling, may

be fruitfully investigated with this method. The advent of neural net hardware [20] will make certain real time applications possible.

Among the issues not addressed in this paper are:

(1) the effect of nonglobal minima (not a problem in our simulations)

(2) the effect of noisy data; and

(3) procedures to update predictions "on the fly" a la Kalman-Bucy.[24]

Acknowledgements

The authors would like to express their gratitute to Doyne Farmer and Sid Sidorovitch for so generously sharing their prepublication results and insights on their alternative method (Local Linear Method) for performing prediction[13],[19]. The Glass-Macky example was suggested by them and they provided the data generator. The authors would also like to acknowledge extremely useful conversations with Y.C. Lee involving the theoretical underpinnings of this work. The generalized mode analysis discussed in Section V was developed in conversation with Y.C. Lee, and he made the initial suggestion that such an interpretation may be possible. Finally, the authors would like to thank Carolyn Algire for expert typing of a difficult manuscript under deadline pressure.

Appendix I: Scaling and Dynamic Range

We have found that the certain numerical problems could be avoided by having inputs and outputs in the range of $0 \rightarrow 1$ during training. We will give a simple scaling procedure that scales the weights that come from training on variables $0 \rightarrow 1$, to those appropriate to handle inputs and output variables of arbitrary dynamic range. Note that the numbers for the normalized root mean square index of accuracy that we quote in the text are independent of dynamic range. Suppose that the real world Inputs X^{in}, and Outputs, X^{out} have an extended dynamic range i.e. greater than $0 \rightarrow 1$. We

can always write them in terms of variables x^{in} and x^{out} that range between o and 1:

$$X^{in} = \alpha x^{in} + \beta \qquad \text{(A1)}$$

$$X^{out} = \gamma x^{out} + \delta$$

where α, β, γ, δ are appropriate constants that adjust the range: $0 \rightarrow 1$ of x^{in}, x^{out} to the range: min $X^{in} \rightarrow$ max X^{in} and min $X^{out} \rightarrow$ max X^{out}; where min and max refer to the minimum and maximum values of X^{in} and X^{out} in the training set. One then trains the nonlinear neural net using the variables x^{in} and x^{out} with range $0 \rightarrow 1$. This results in a set of weights and thresholds, T_{ij} and I_i, geared to the range $0 \rightarrow 1$.

We now wish to scale T_{ij} and I_i to enable the network to handle the real world dynamic range of the variables X^{in}, X^{out}. We will describe the scaling procedure for a network with a single hidden layer and Input $\rightarrow$ Hidden $\rightarrow$ Output connectivity. Similar arguments work for any network connectivity.

First, consider the connections between the Input and Hidden layer. If x_j^{in} represents the inputs in a range $0 \rightarrow 1$, then the Hidden layer produces an output involving the nonlinear transfer function g( )

$$g(\Sigma_j T_{ij} x_j^{in} + I) \qquad \text{(A2)}$$

If one now used the original, real world, values X_j^{out} for the Input layer then the hidden layer would, of course, output a different number. We wish to scale T_{ij} and I_i to T_{ij}^{scaled} and I_i^{scaled} so that the hidden layer when using X_j^{in} and T_{ij}^{scaled}, I_i^{scaled}, will output the same number as number as from expression (A2). It will then be trivial to scale the Hidden $\rightarrow$ Output connections to get the outputs in the desired dynamic range.

Thus we require:

$$g(\Sigma_j T_{ij} x_j^{in} + I_i) = g(\Sigma_j T_{ij}^{scaled} x_j^{in} + I_i^{scaled}) \quad (A3)$$

Equations (A3) and (A1) then yield (for Input → Hidden connections)

$$T_{ij}^{scaled} = T_{ij}/\alpha \quad (A4)$$

$$I_i^{scaled} = I_i - (\beta/\alpha)\Sigma_j T_{ij}$$

Since the Output layer sees

$$\Sigma_j T_{ij} g(\ ) + I_i \quad (A5)$$

where j ranges over the Hidden indices, and since $g(\ )$ is outputting the same value for the scaled inputs as for the unscaled inputs, we see from the Eqns. (A1 and A5) that we merely need to define

$$T_{ij}^{scaled} = \gamma T_{ij} \quad (A6)$$

$$I_i^{scaled} = \gamma I_i + \delta$$

for the Hidden → Output connections to achieve the desired dynamic range for the outputs.

Similar scaling arguments work for arbitrary connectivity. If the traning data set is not typical of the range of values to be used for later prediction, then a possibility for error exists. However, one must always assume the training set is a typical data set, or else the whole concept of training fails. We also reiterate that the normalized root mean square index of accuracy used throughout this is insensitive to the scale of the data.

Appendix II: A Note on Simulations

In contrast to simulations of problems involving symbolic data, where an accuracy of .1 is commonly used to generate answers where outputs are identified as 0 or 1's,[1],[9] the real number problems considered in this paper require high accuracy. For this reason, we typically ran our simula-

tions on a Cray X-MP to speed-up convergence to an exact minimum. In addition, we found that use of a conjugate gradient minimizing procedure, instead of the commonly used steepest descents procedure, produced orders of magnitude speedups in convergence. All simulations were run with a general purpose mathematical simulation package that is in development by R. Farber at Los Alamos. The package includes a general purpose, menu driven, interactive frontend module (written in C), which accepts network parameters and also data files. It then communicates them to a computational module that can be written in either C or Fortran, which can then be run on any number of machines, including parallel machines. An automatic code generation module was also developed and used to produce optimally vectorizing Cray Fortran code. Results are automatically collected at a host machine, formatted, saved to disk with unique run name file labels, and a formatted hardcopy is also produced for aiding documentation of the runs. The package is still in an experimental state and is presently tailored to the peculiarities of the Los Alamos computer network. Run times were on the order of minutes for the logistic map prediction and approximately an hour for the longer term predictions in the Glass-Macky equation. These run times were needed to obtain exceedingly accurate approximations to the minima of the back propagation energy function. Acceptable accuracy may be achieved with significantly shorter run times, however we decided to see just how close to the exact minimum one could come given virtually unlimited run time. It is inherent in the operation of the neural net that it produces an excellent global approximation to the maps underlying the time series. If such a global map were produced by patching together a large number of local linear approximations obtained from the method of Farmer and Sidorovitch then presumably run times of similar length would be encountered. The advent of parallel, neural net hardware (chips) should allow the nonlinear neural net method to run in real time for numerous signal processing applications.

References

(1) D. Rummelhart, J. McClelland in "Parallel Distributed Processing," Vol. 1, M.I.T. Press, Cambridge, MA (1986).

(2) B. Widrow, S. Stearns, "Adaptive Signal Processing," Prentice Hall Inc., Englewood Cliffs, NJ (1985).

(3) B. Widrow, M. Hoff, "Adaptive Switching Circuits," 1960 WESCON Convention Record part IV, p96 (1960).

(4) D. Ruelle, F. Takens, Comm. Math. Phys. 20 p167 (1971).

(5) H. Swinney el al., Physics Today 31 (8), p41 (1978).

(6) K. Tomita et al., J. Stat. Phys. 21, p65 (1979).

(7) H. Haken, Phys. Lett. A53, p77 (1975).

(8) D. Russell et al., Phys. Rev. Lett. 45, p1175 (1980).

(9) T. Sejnowski et al., "net Talk: A Parallel Network that Learns to Read Aloud," Johns Hopkins Univ. preprint (1986).

(10) D. Gabor et al., Proc. Inst. Electr. Eng. 1088 (July 1960), see also "The Volterra and Weiner Theories of Nonlinear Systems," M. Schetzen, John Wiley, and Sons (1970).

(11) J. Makhoul, Proc. IEEE 63 No. 4, p561 (1975).

(12) M. Feigenbaum, J. Stat. Phys. 19, p25 (1978).

(13) D. Farmer, J. Sidorvitch (private communication), see also their preprint "Predicting Chaotic Time Series," (Los Alamos National Laboratory, 5/87)

(14) M. Mackey, L. Glass, Science 197, p287 (1977).

(15) for more on strange attractors, see: E. Ott, Rev. Mod Phys. 53, No. 4, p655 (1981).

(16) D. Farmer, Physica 4D, No. 3, p366 (1982).

(17) N. Packard et al. Phys. Rev. Lett. 45, No. 9, p712 (1980).

(18) F. Takens, "Detecting Strange Attractor In Turbulence," Lecture Notes in Mathematics, D. Rand, L. Young (editors), Springer Berlin, p366 (1981).

(19) D. Farmer, J. Sidorovitch (private communication).

REFERENCES (continued)

(20) H.P. Graf et al., "VLSI Implementations of a Neural Net Memory with Several Hundreds of Neurons," p182, A.I.P. Conference Proceedings #151, J. Denker (editor), American Institute of Physics, NY (1986).

(21) C. de Boor, "A Practical Guide to Splines," Springer Verlag, NY (1978).

(22) M. Maqusi, "Applied Walsh Analysis," Heyden and Sons Ltd., London, England (1981).

(23) For a reference to the use of trigometric sin transfer functions in backpropagation, see E. Baum, A.I.P. Conference Proceedings #151, American Institute of Physics, p47 (1986).

(24) "Kalman Filtering," H.W. Sorenson, editor IEEE Press New York, (1985).

FIGURE CAPTIONS

Figure 1 - A plant implementing the nonlinear transfer function $x \to \dot{x}^2$

Figure 2 - A linear feedforward neural net representation for linear prediction.

Figure 3 - A feedforward neural net with a nonlinear hidden layer for prediction. Arrows schematically indicate the feedforward connections from Input to Hidden to Output layers.

Figure 4 - The transfer function g( ) of a hidden unit is commonly taken to be a nonlinear, sigmoidal function with range $0 \to 1$. This is a plot of $g(x) = \frac{1}{2}(1 + \tanh(x))$. The exact analytic form of the sigmoid is not critical to results.

Figure 5 - A network with 5 nonlinear hidden units used to predict $x(t + 1)$ given $x(t)$. The arrows schematically indicate connections from the Input layer to all the units in the Hidden layer, as well as connections from all Hidden units to the Output unit. Also, the Input layer directly connects to the Output layer.

Figure 6a - A plot of $x(t)$ vs t for a time span of 500 time steps in units where $\tau = 17$ for the Mackey-Glass equation (15). There is a quasiperiodicity, however details of the bumps change chaotically over time.

Figure 6b - Figure 6b is a similar plot to Figure 6a for $\tau = 30$.

Figure 7 - The 2 Hidden layer network architecture for prediction in the Mackey-Glass equation (15). The arrows schematically indicate connections from Input to Hidden to Output layers.

FIGURE CAPTIONS (continued)

Figure 8a - A plot of the normalized rootmean square error (the root mean square error divided by the standard deviation of the data) versus prediction time step, P, into the future for the Mackey-Glass equation at $\tau = 17$. Five curves are shown, labelled A, B, C, D, E. A = Sixth order polynomial (iterated from P = 6). B = The Linear Predictive method. C = Sixth order polynomial trained at each prediction time step, P. D = Nonlinear neural net trained at each prediction time step, P. E = Nonlinear neural net (iterated from P = 6). The iterated polynomial is wildly unstable and quickly blows up. The iterated nonlinear neural net performs best.

Figure 8b - An identical plot to Figure 8a except for the Mackey-Glass equation at $\tau = 30$. The labelling of the curves is the same. The success of the iterated nonlinear neural net method is even more evident.

Figure 9 - The 2 Hidden layer network architecture for modelling $x \rightarrow \dot{x}^2$. The arrows schematically indicate connections from Input to Hidden to Output layers. Another network, with a single layer of 40 hidden units performed comparably to the 2 layer network.

Figure 10 - A graph of $\dot{x}^2 = [x(t) - x(t - \Delta t]/(\Delta t)^2$ where $\Delta t = .001$ and $Z = \dot{x}^2$ is plotted as a function of two independent variables $x = x(t)$, $y = x(t - \Delta t)$. The axes have been rotated so that one is looking straight down a parabolic trough. In the rotated coordinate system, the region $x(t) \simeq x(t - .001)$ is along the bottom of the trough. The horizonal, 45 degree and vertical lines represent the x, y, z axes. The z axis has been drawn to the left of the parabola to avoid cluttering the parabolic surface.

FIGURE CAPTIONS (continued)

Figure 11 - The output of the network in Figure 9 plotted as a function of the two input variables. The range of the plot has been considerably extended past the range of validity of the map. The x and y axes extend to from -1 to 1 while Z extends from 0 to 5.125. The range of validity, $x(t) \simeq x(t - \Delta t)$, is along the bottom of the trough because we have rotated axes so that one is looking straight down the trough.

Figure 12 - A blow up of the bottom of the trough in Figure 11. It is only the bottom of the trough in Figure 11 that is relevant. The axes are rotated so that one is looking straight down a parabolic trough.

Figure 13 - A plot of successive sums of terms of equation (20. The first window contains term1 as a solid line and term2, to be added in, as a dotted line. Window(2) shows the sum of term(1) + term(2) as a solid line and term(3), to be added in as a dotted line, and so on. Window(6) contains the sum of all 6 terms of equation (20) plotted as a final solid line. Each tick mark is one unit, so that the approximation to the parabolic map, Eqn. (22), is contained in the region 0 → 1. Recall that the normalized root mean suare predictive accuracy of this map was 1.4×10^{-4}, i.e. it is an excellent approximation. Portions of the graphs in windows(1) to window(6) outside the valid range of 0 → 1 are plotted so that the form of each sigmoid that is added together may be seen.

Figure 1

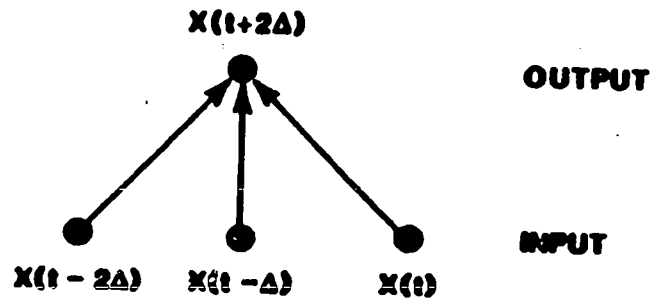

Figure 2

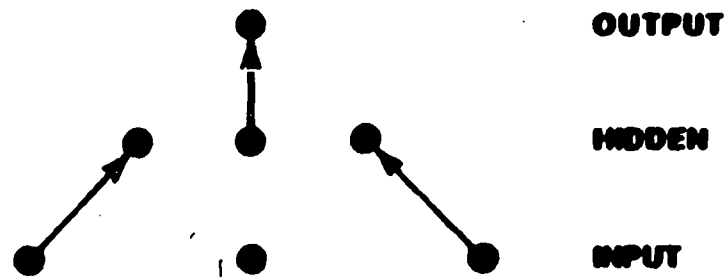

Figure 3

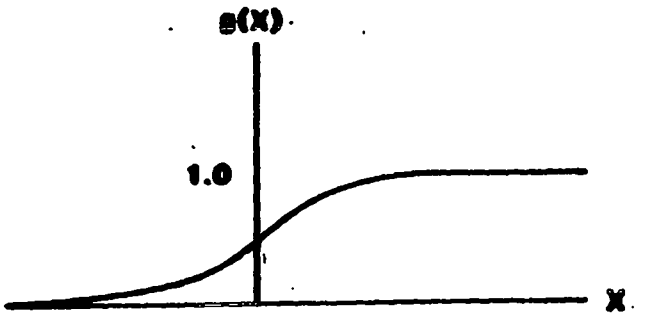

Figure 4

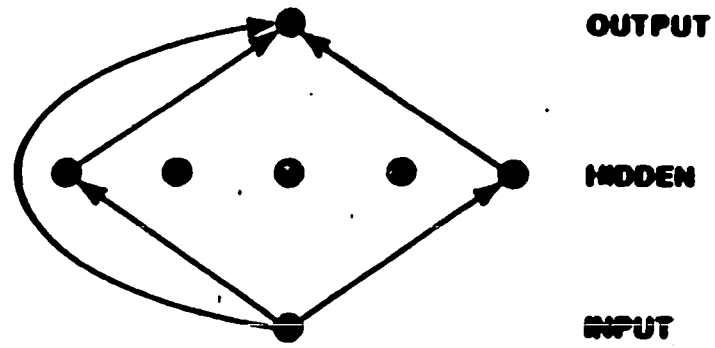

Figure 5

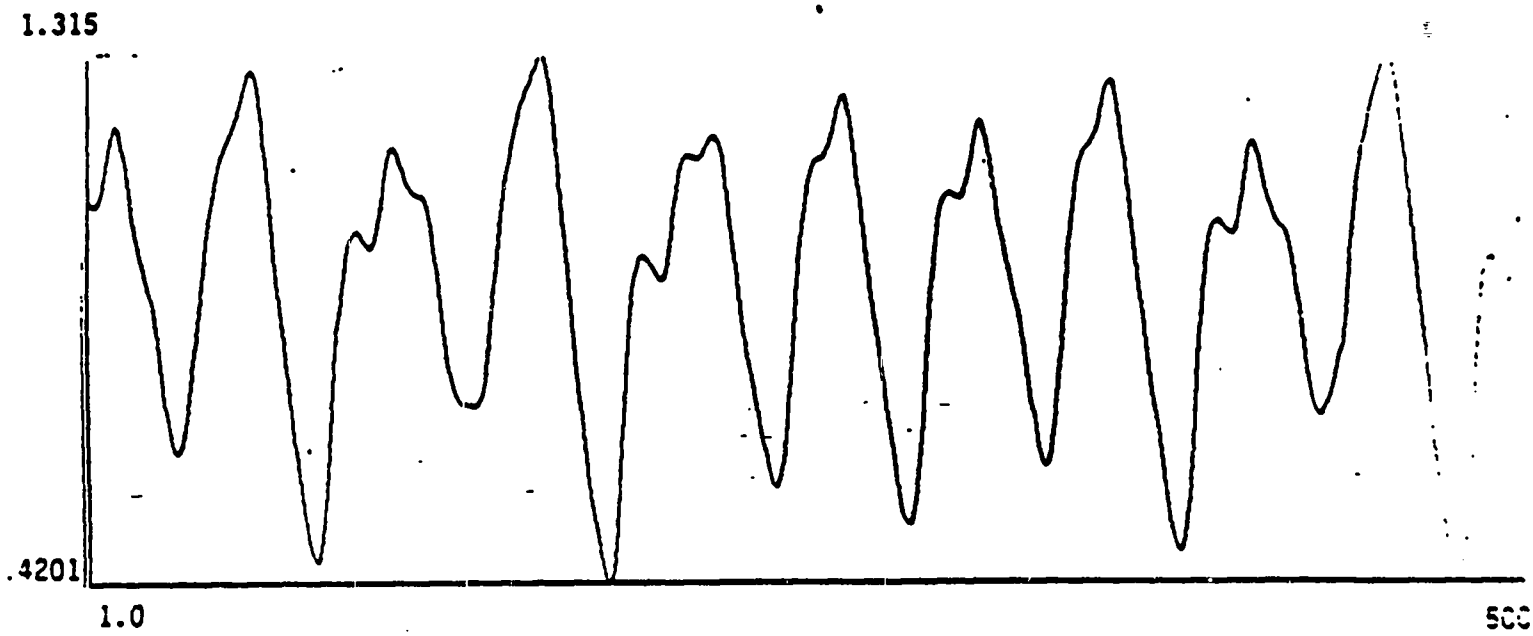

Figure 6a

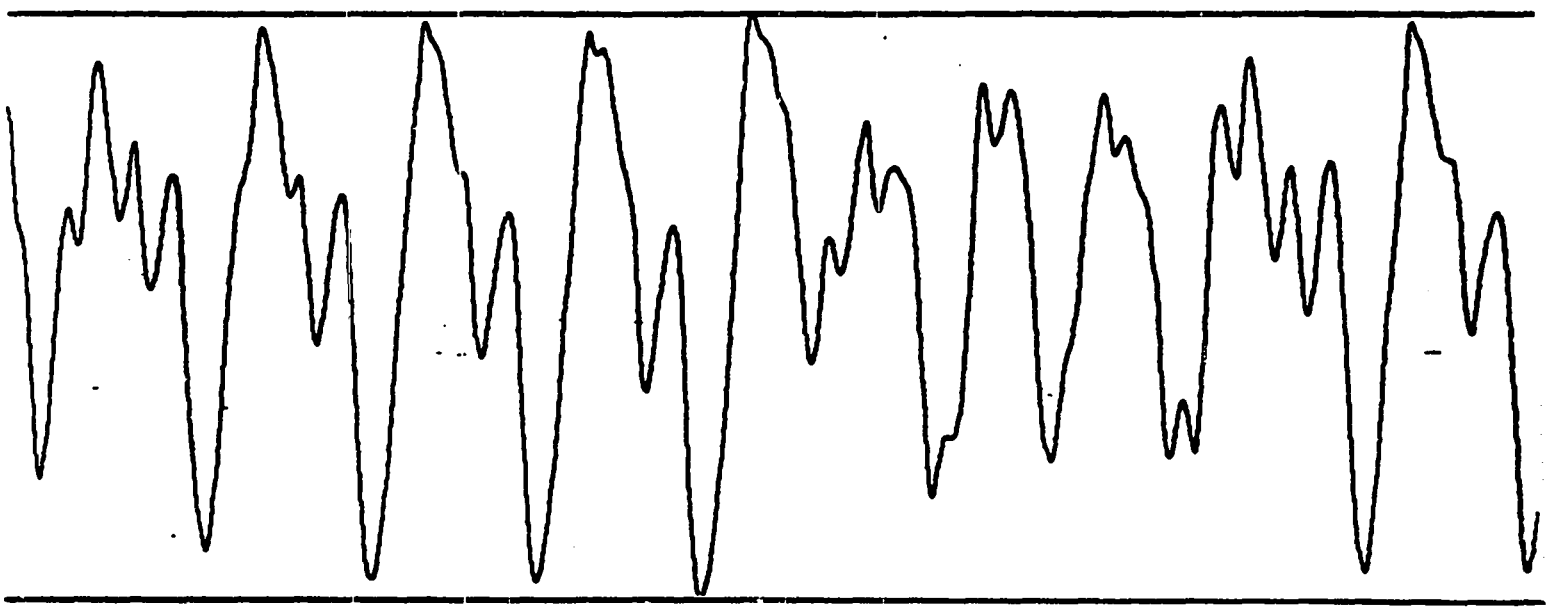

Figure 6b

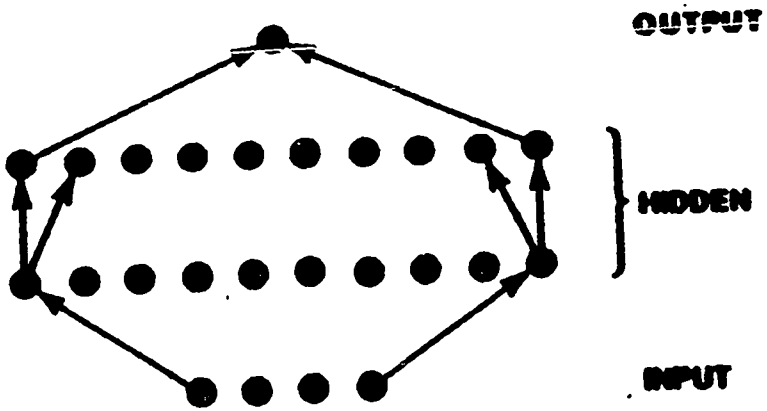

Figure 7

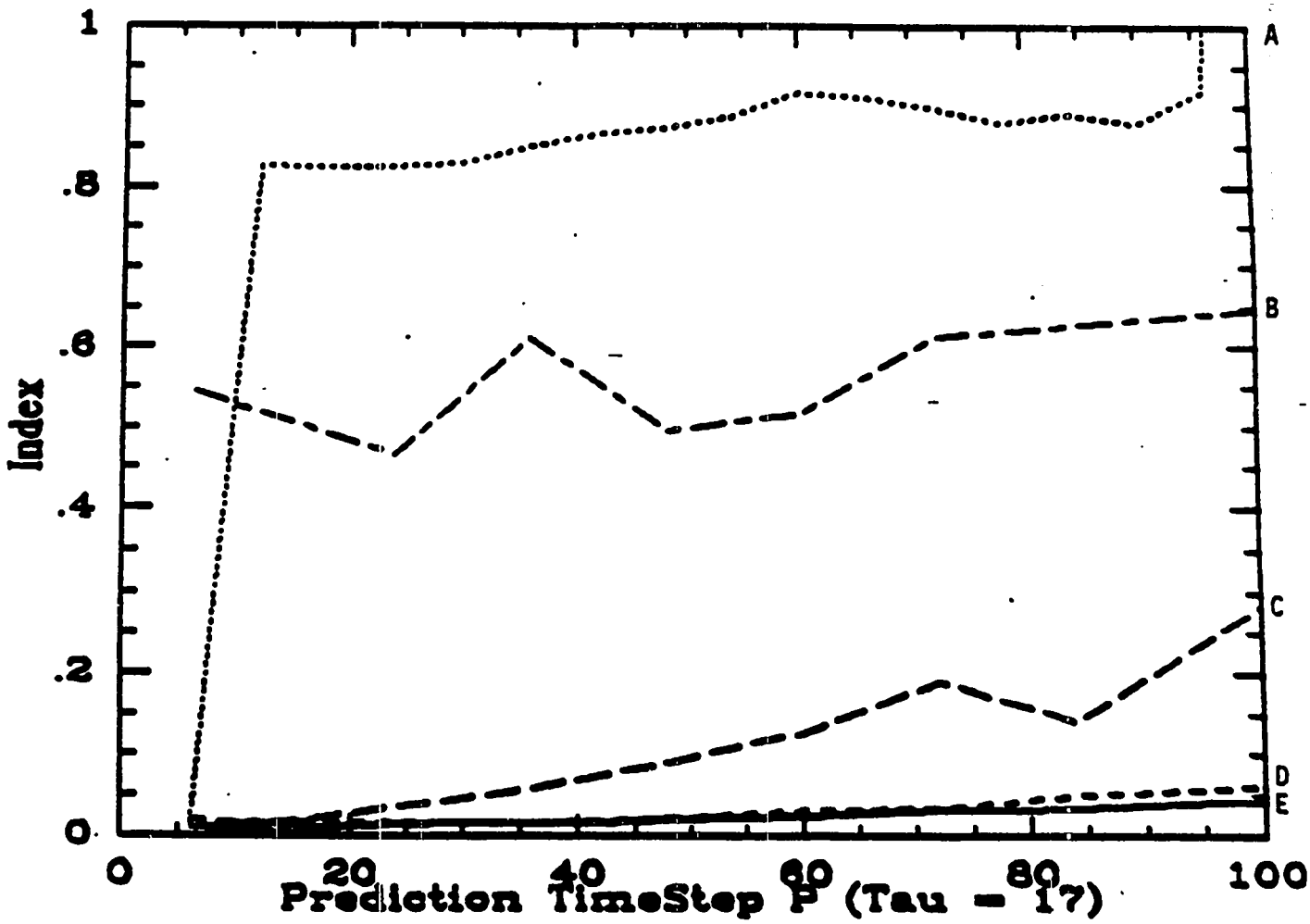
1
.8
.6
.4
.2
0
Index
A
B
C
D
E
0
20
40
60
80
100
Prediction TimeStep P (Tau = 17)

Figure 8a

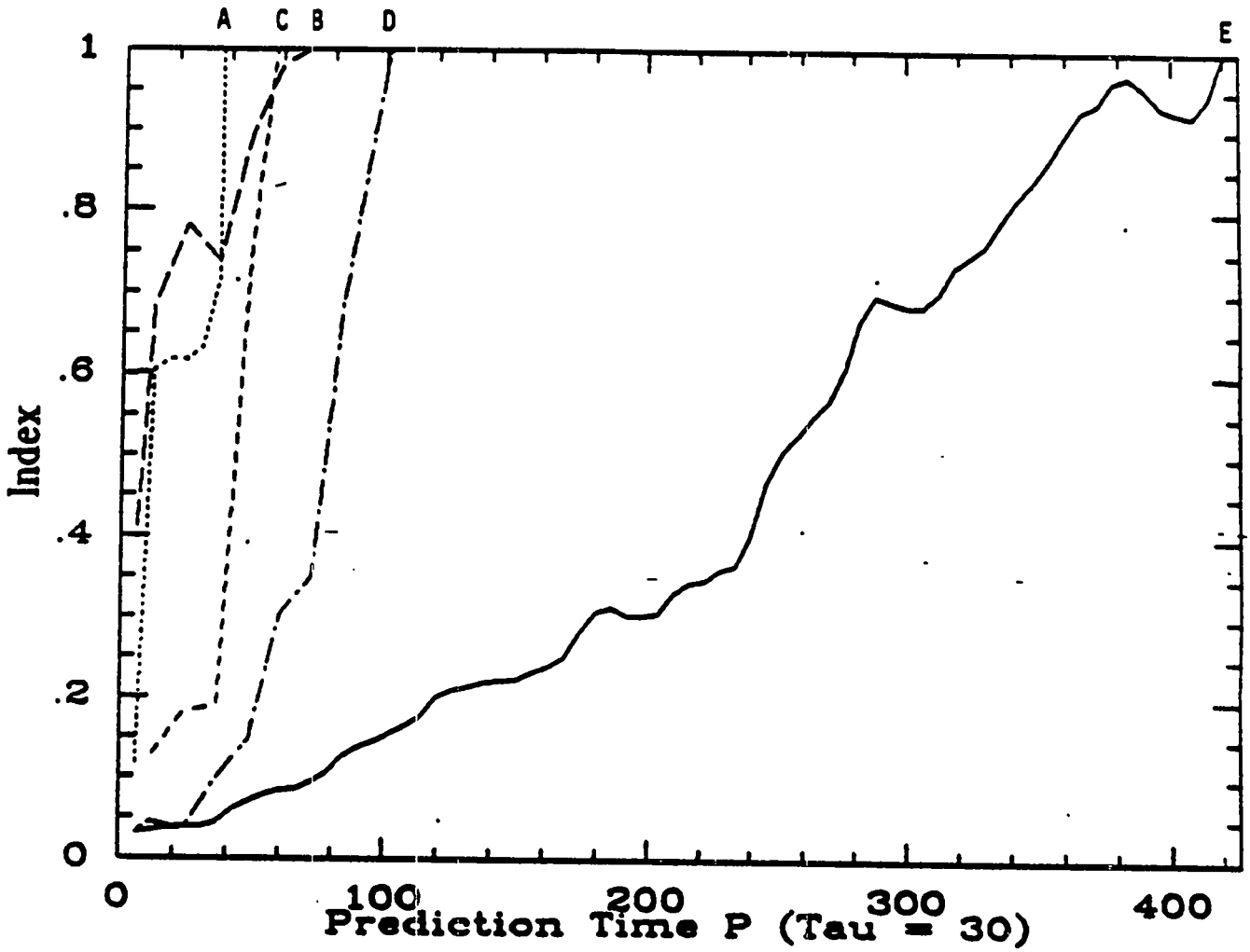

Figure 8b

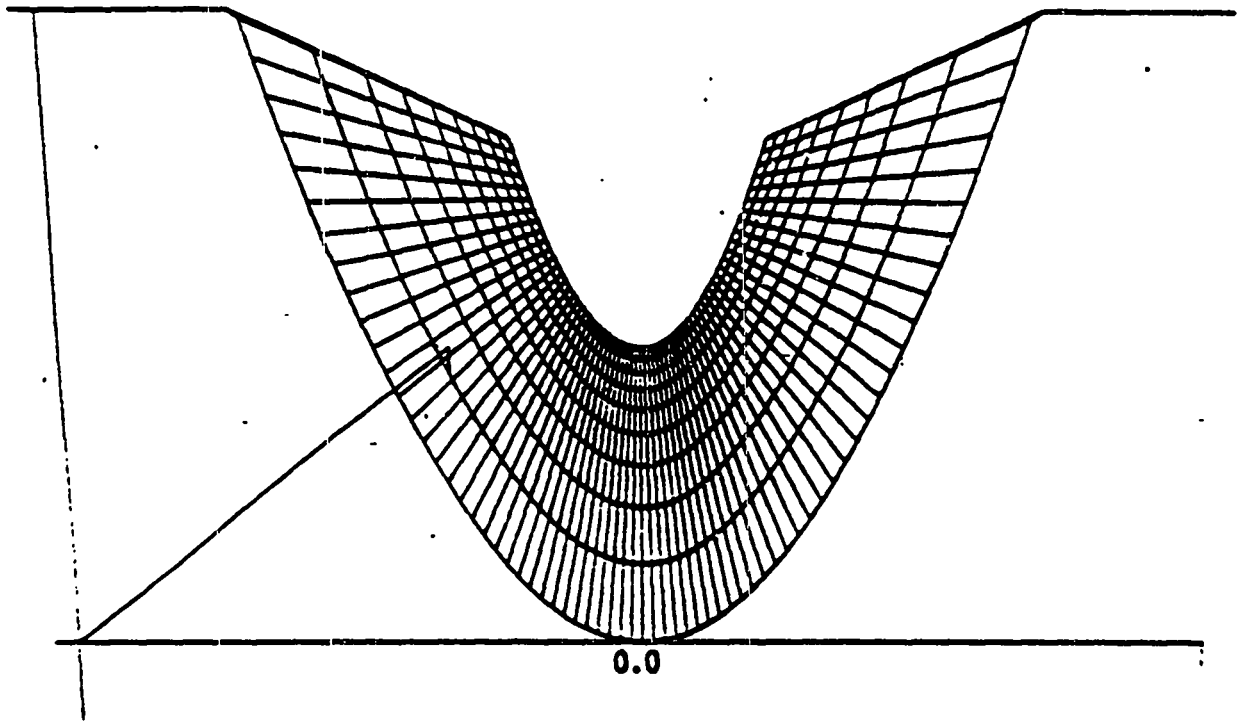

Figure 10

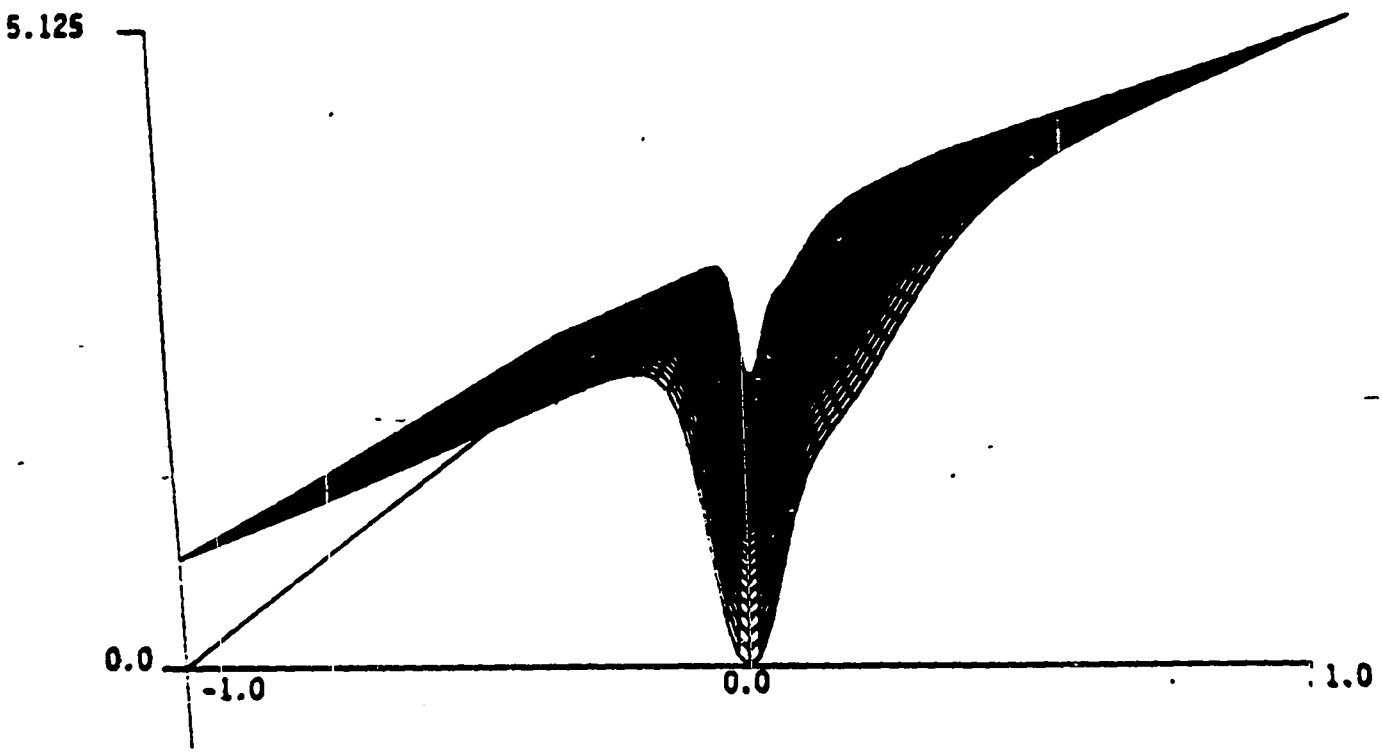

Figure 11

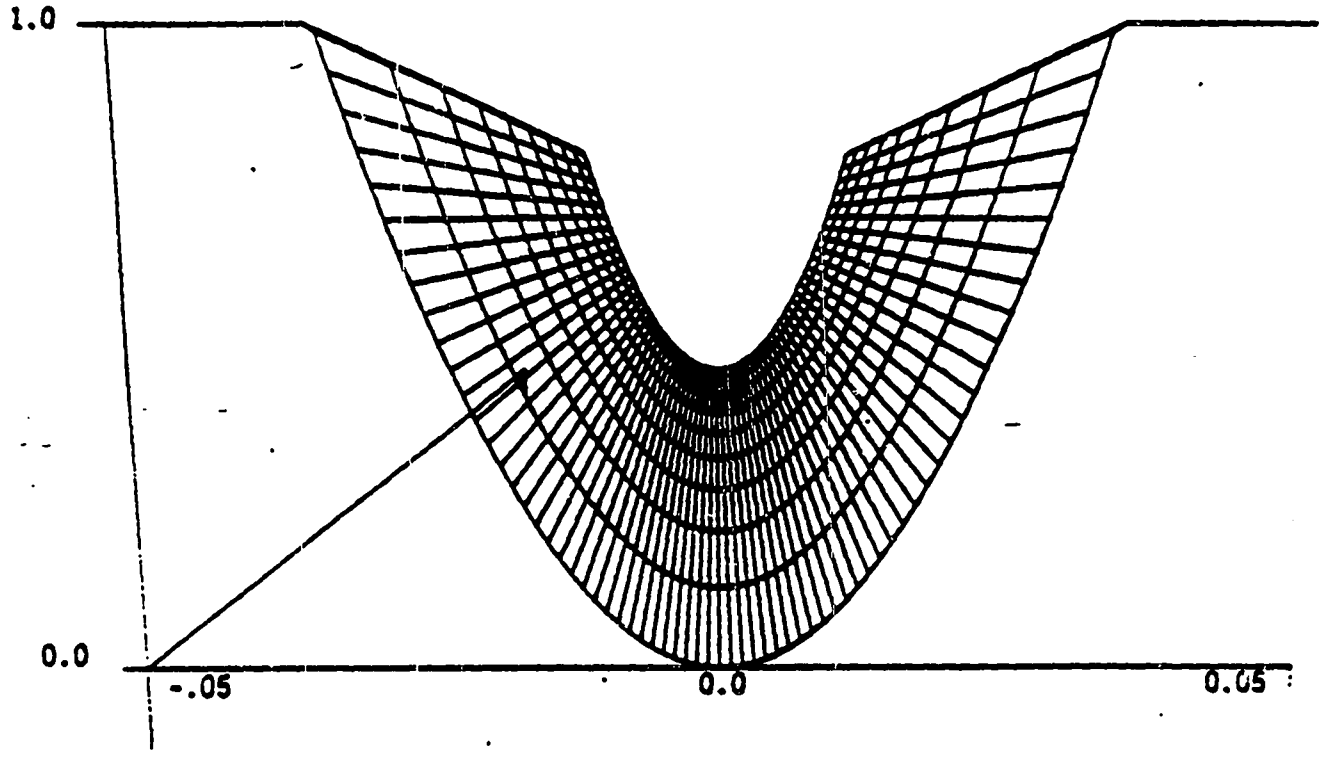

Figure 12