Using neural networks for controlling chaos

P. M. Alsing,* A. Gavrielides,† and V. Kovanis‡
Nonlinear Optics Center, Phillips Laboratory, 3350 Aberdeen Avenue SE, Kirtland Air Force Base, New Mexico 87117-5776
(Received 9 August 1993)

A feed-forward backpropagating neural network is trained to achieve and maintain control of the unstable periodic orbits embedded in a chaotic attractor. The controlling algorithms used for training the network are based on the now standard scheme developed by Ott, Grebogi, and Yorke [Phys. Rev. Lett. **64**, 1196 (1990)], including variants that utilize previous perturbations and/or delayed time-series data.

PACS number(s): 05.45.+b, 42.79.Ta

I. INTRODUCTION

Chaotic systems are characterized by their sensitive dependence to small perturbations. In recent years, an abundance of theoretical and experimental research has been developed to capitalize on this fact and utilize it to control chaotic systems by applying very small, appropriately timed perturbations. Researchers have demonstrated the control of chaos in a host of physical systems ranging from lasers and electronic circuits to chemical and biological systems. For an excellent, recent review article see [1] and references therein.

Neural networks are useful for a vast array of applications involving pattern and symbol recognition as well as for numerical computations. Recently, neural networks have been employed in the arena of dynamical systems as a tool for recognizing chaos in noisy experimentally obtained signals [2,3]. In this work we combine the two ideas to train a feed-forward backpropagating neural network to control a chaotic dynamical system. We demonstrate the ability of the neural network to act as a chaotic controller which can be trained on a variety of controlling algorithms.

In this paper we train the neural network from a time series generated from a known, deterministic chaotic map. However, it is important to note that the training of the neural network does not rely on a detailed, *a priori* knowledge of this mapping. The training of the neural network is accomplished solely from the data it receives at its inputs. Whether these data are generated deterministically from a known mapping or come from a given experimental time series is of no consequence to the training of the neural network. Our decision to use data generated analytically from a map was made purely for computational convenience. In Sec. IV C we discuss how current time-series analysis techniques can be used to train the neural network given only an experimental time series.

This paper is organized as follows. In Sec. II we briefly review the central ideas involved in controlling chaotic trajectories. In Sec. III we present a brief overview of backpropagating neural networks. In Sec. IV we demonstrate the ability of the neural network to control an unstable periodic orbit of a chaotic map employing three different controlling schemes. Finally, in Sec. V we summarize our results and conclude.

II. CONTROLLING CHAOS

The notion of controlling chaos in nonlinear dynamical systems has found widespread popularity since the publication of the seminal paper by Ott, Grebogi, and Yorke [4]. The central idea for controlling chaos is based on the observation that a chaotic attractor has embedded within it an infinite number of unstable periodic orbits. These unstable periodic orbits are characterized by having a stable manifold for which nearby trajectories are attracted to this orbit, and an unstable manifold for which nearby trajectories are repelled away from this unstable orbit. By applying judiciously chosen perturbations to an accessible system parameter the authors showed that such unstable periodic orbits could be stabilized. The advantage of their algorithm, colloquially called the *OGY method*, is that it does not require knowledge of the system equations. In addition, by using the method of embedding coordinates [5], the information required for controlling chaos can be constructed from a single experimental time series (e.g., a current or voltage signal).

Let $\boldsymbol{\xi}_n$ be the phase-space values at the crossing of some surface of section through which the trajectories of some dynamical system pass. There exists a Poincaré map $\boldsymbol{F}$, depending on some control parameter p, which relates the values of successive iterative crossings via $\boldsymbol{\xi}_{n+1} = \boldsymbol{F}(\boldsymbol{\xi}_n, p)$. In their original paper, Ott, Grebogi, and Yorke derived a formula for the perturbations necessary to control the unstable periodic orbits

$$\delta p_n = \frac{\lambda_u}{\lambda_u - 1} \frac{\boldsymbol{f}_u \cdot (\boldsymbol{\xi}_n - \boldsymbol{\xi}_F(p_0))}{\boldsymbol{f}_u \cdot \boldsymbol{g}}. \tag{2.1}$$

Here, λ_u is the eigenvalue of the unstable contravariant

*Electronic address: alsing@arom.plk.af.mil
†Electronic address: tom@photon.plk.af.mil
‡Electronic address: kovanis@xaos.plk.af.mil

eigenvector $\boldsymbol{f}_u$ of the local mapping $\boldsymbol{M} = \boldsymbol{D}_{\boldsymbol{\xi}}\boldsymbol{F}(\boldsymbol{\xi}_F, p_0)$, evaluated at the fixed point $\boldsymbol{\xi}_F$, of $\boldsymbol{F}$, and $\boldsymbol{g} \equiv \boldsymbol{D}_p\boldsymbol{F}(\boldsymbol{\xi}_F, p_0)$ is the shift of the fixed point due to a change in the control parameter. The OGY formula is derived by requiring that, upon application of the perturbation, the next iterate of the mapping falls on the stable manifold of the fixed point. If this is the case, then successive iterates of the mapping will be attracted to the fixed point, and a period-1 orbit will have been extracted from the chaotic attractor.

In their original paper on controlling chaos [4] the authors demonstrated that the algorithm, Eq. (2.1), was tolerant of noise. The amount of noise for which control can still be successfully maintained can be quantified. By setting δp_n equal to the maximum allowable perturbation δp_* and inverting Eq. (2.1), one can find the maximum distance ξ_*^u from the fixed point in which the applied perturbations induces control,

$$\xi_*^u \equiv \boldsymbol{f}_u \cdot (\boldsymbol{\xi}_* - \boldsymbol{\xi}_F(p_0)) = |(1 - \lambda_u^{-1})\boldsymbol{f}_u \cdot \boldsymbol{g}|\delta p_*. \quad (2.2)$$

Suppose one now adds noise to the dynamical system of the form $\epsilon\boldsymbol{\delta}_n$ where $\boldsymbol{\delta}_n$ is a Gaussian random variable of mean zero and unit variance and ϵ is a small number specifying the intensity of the noise. The control condition in the absence of noise, $\xi_*^u = 0$, becomes $\xi_*^u = \epsilon\delta_n^u$ in the presence of noise, where $\delta_n^u \equiv \boldsymbol{f}_u \cdot \boldsymbol{\delta}_n$. Therefore, if the noise level is bounded $|\delta_n^u| < \delta_{\max}$, then control will be maintained as long as $\epsilon\delta_{\max} < \xi_*^u$.

The controlling algorithm can be generalized for situations in which there is more than one unstable direction [6]. It has been adapted for a variety of situations in which the mapping $\boldsymbol{F}$ may depend on the previous perturbation [7,8]. The restriction of achieving control by using all the phase-space variables $\boldsymbol{\xi}$ can be lifted and replaced by one in which a single phase-space variable is used. This requires a modification of Eq. (2.1) to include a history of a finite number of previous perturbations. All these schemes are important refinements of the controlling algorithm, but the central, innovating theme for controlling unstable periodic orbits contained within a chaotic attractor is contained in the original OGY formula, Eq. (2.1).

III. NEURAL NETWORKS

The field of neural networks and neural computing is so vast that we give here only a brief introduction to the subject. For the application of using neural networks for controlling chaos we restrict ourselves to a discussion of the feed-forward backpropagating neural networks. Such networks have been used to predict chaotic time-series time [2] and to reconstruct chaotic attractors using noisy data [3]. For a general survey of neural networks and for further references see [9].

The general backpropagating neural network consists of a series of input and output ports connected by way of a set of intermediate units arranged in a series of *hidden layers*. In this work we used both single and double hidden layer neural networks, but found no significant difference in performance between the two. For simplicity we discuss only the single hidden layer network.

Each unit of the neural network has an associated input and output function. The input to a particular unit is the weighted sum of all the outputs of the previous layer plus an offset. This input is then passed through an activation function, typically some type of sigmoidal function. The output of the unit is then a continuous, smooth, monotonic function of the input which approximates the firing (the on-off nature) of real neurons. This output, along with the rest of the outputs for that layer are then used as the inputs for the succeeding layer, and the process repeats itself.

Moving from left to right in Fig. 1 we can follow the progression from initial input to final output for our single hidden layer neural network. Let us designate the inputs as $\{x_1, x_2\}$ and the single output as y^{out}. The input to the jth unit of the hidden layer is given by $I_j^{(1)} = b_j^{(1)} + \sum_i w_{ij}^{(1)} x_i$. (Although we only have one hidden layer here, the superscripts enumerating the particular layer are included for purposes of indicating extensions to more than one layer.) The quantities $w_{ij}^{(1)}$ are called the *weights* and $b_j^{(1)}$ are called the *biases*. The input $I_j^{(1)}$ is converted to an output $O_j^{(1)}$ by being passed through an activation function, $O_j^{(1)} = \frac{1}{2}(1 + \tanh I_j^{(1)})$.

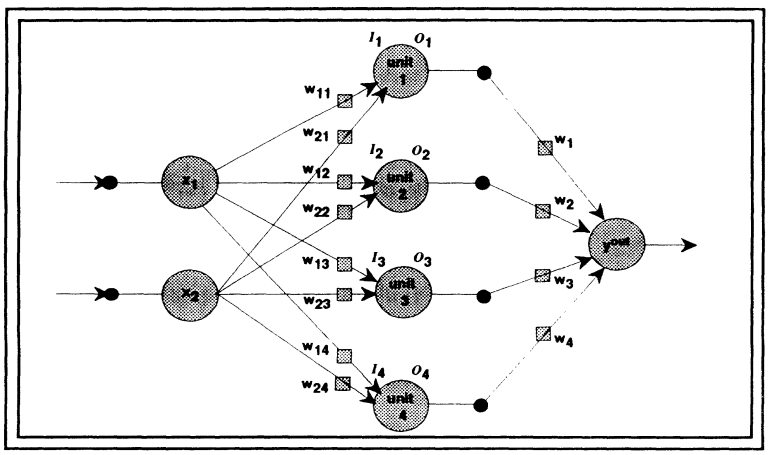

FIG. 1. A backpropagating neural network shown here with two inputs, one hidden layer consisting of four units, and one output.

Other activation functions, such as the sigmoid $O(x) = [1 + \exp(-kx)]^{-1}$, are commonly used in the literature. The output may be symmetrized to $(-1, 1)$ instead of $(0, 1)$ used above. The requirement for the activation function is that it be monotonic, bounded from above and below and everywhere differentiable. The output is obtained in a similar fashion as a weighted sum of the outputs plus a bias, $y^{\text{out}} = b^{(o)} + \sum_j w_j^{(o)} O_j^{(1)}$. The superscript (o) indicates the output layer. If there were more than just one output unit, these weights and bias would have an additional index relating the ith previous input to the jth output (i.e., $y_j^{(o)} = b_j^{(o)} + \sum_i w_{ij}^{(o)} O_i^{(1)}$). The neural network is easily generalized to include an arbitrary number of hidden layers by using the following input and output functions for the jth unit in the nth hidden layer, $I_j^{(n)} = b_j^{(n)} + \sum_i w_{ij}^{(n)} O_i^{(n-1)}$ and $O_j^{(n)} = \frac{1}{2}(1 + \tanh I_j^{(n)})$.

The neural network can be made to produce a desired output by training it on a large set of inputs. Designating the inputs as a discrete time series $\{x_n\}$, the training proceeds as follows: (1) begin with a random assignment of weights and biases, (2) calculate the output $\{y_n^{\text{out}}\}$ for all sets of input data, (3) compute an error function, $Q = \sum_{n=1}^{N} (y_n - y_n^{\text{out}})^2$ where N is the number of points in a given input data set, and (4) adjust the weights and biases to minimize the error.

The crucial step is the last one. The name *backpropagating* is derived from the procedure used for propagating the error backwards to adjust the weights and biases. The standard method of steepest descent is used to perform the minimization. The weights and biases are treated as independent parameters $\{\alpha_i\}$, where i takes on as many values as there are weights and biases. By differentiating the error Q with respect to α_i one can compute the change in the error for a given change in an independent parameter [3], $\delta Q = (\partial Q/\partial \alpha_i)\, \delta \alpha_i$ where $(\partial Q/\partial \alpha_i)$ is evaluated at the current value of the parameters. To insure that adjustments in α_i produce a decrease in the error, one chooses $\delta \alpha_i = -\gamma (\partial Q/\partial \alpha_i)$. The quantity $\gamma > 0$ is called the *learning rate* and is chosen to be small enough so that the desired accuracy in the error can be achieved, but large enough so that this can be obtained in a reasonable amount of time. This formula for $\delta \alpha_i$ can be altered to steer the adjustments of the parameters in the same direction regardless of the direction determined by the gradient. This is accomplished by adding a proportional value of the previous adjustment to the formula above, $\delta \alpha_i = -\gamma (\partial Q/\partial \alpha_i) + \mu \delta \alpha_i^{\text{prev}}$, where $\mu > 0$ is another rate, appropriately called the *momentum*.

In a recent paper, Albano *et al.* [3] successfully trained a single hidden layer backpropagating neural network to reconstruct the chaotic attractors for the logistic and Hénon maps using noisy as well as noise-free input. The noisy data were generated by adding Gaussian random noise to the clean data produced from the dynamical equations. Enough noise was added to produce a noise variance equal to the signal variance, so that the attractor was obscured. The neural network was trained on the clean data until a minimum error Q was obtained. By then feeding in various length input sets of noisy data the neural net was used to predict future values of the times series and construct a first return map, a plot of x_{n+1} vs x_n. In addition, the authors were able to use these reconstructed attractors generated from the neural network to calculate a correlation dimension to within 10% of the theoretical values.

The authors point out that there did not exist consistent criteria for determining the optimal architecture of the network (i.e., the number of input and hidden layers), or of the optimal size of the training set. In addition, different initial random weights often led to different minima of the error function, resulting in different levels of performance even when the same input data were used. They pointed out that one set of initial weights and biases might lead to a time series which successfully constructed the whole attractor, while another might reconstruct only short segments, or get trapped in a periodic motion on a small set of points. The latter behavior could often be alleviated by reconstructing the attractor by producing several sets of outputs generated from randomly chosen starting points in the input data set [3].

IV. USING BACKPROPAGATING NEURAL NETWORKS TO CONTROL CHAOS

The objective of this work is to demonstrate that a neural network can be used to implement the OGY formula, Eq. (2.1). Since the formula for δp_n in Eq. (2.1) is linear in the phase-space variables $\boldsymbol{\xi}_n$, a neural network with input ports being fed a time series of the phase-space variables can be trained to produce the perturbation times series necessary for control. The neural network (NN) need not be trained over the whole of the chaotic attractor. Instead, it is necessary to train the network only in the vicinity of the desired fixed point of the unstable periodic orbit. Since the output δp_n^{NN}, of the network does not depend on any previous perturbations, the neural network is not being used, in this case, to predict the next value of the chaotic time series near the fixed point. It is in fact being taught the simple task of learning the coefficients $\{\beta_i\}$ of a linear function of the inputs of the form $\delta p_n = \sum_{i=1}^{N} \beta_i (\xi_n^i - \xi_F^i)$, which is the structure of Eq. (2.1). We also demonstrate that a neural network can be used to implement control in cases where Eq. (2.1) is modified to include the previous perturbation δp_{n-1}. In this case, the output of the neural network δp_n is no longer a simple linear function of the inputs due to the recursive nature of the control formula. The neural network then implicitly predicts the value of the chaotic time series in the neighborhood of the fixed point and uses it to construct the controlling perturbation.

A. Training the neural network on a linear function of the inputs

In this work we demonstrate our results on the Hénon map. This is a two-dimensional mapping with a correlation dimension $\simeq 1.25$ given by

$$x_{n+1} = A - x_n^2 + Bx_{n-1}. \quad (4.1)$$

For parameter values $A = 1.29$ and $B = 0.3$ the mapping is chaotic. Figure 2 is a plot of x_{n+1} vs x_n that displays the Hénon attractor. The Hénon map is a convenient tool to work with since all the structural coefficients in Eq. (2.1) can be worked out analytically [6]. Equation (2.1) then takes the specific form

$$\delta p_n^H = 1.840(x_n - x_F) - 0.300(x_{n-1} - x_F), \quad (4.2)$$

where $x_F = 0.838\,486$ is the value of the fixed point for the period-1 unstable orbit. We used a neural network consisting of two input ports, one hidden layer with four units and one output port. The training of the neural network was performed using pairs of data (x_n, x_{n-1}), generated from Eq. (4.1), which were within a given $\epsilon_{\rm NN} = 0.05$ of x_F, and sending these to the first and second input ports of the network. During the training process the perturbations generated by the neural network were *not* used to alter the input time series. The network was therefore trained on an unperturbed chaotic signal, Eq. (4.1). When used for controlling, the output generated by the neural network was used to perturb the time series, which was sent to its inputs. The perturbed Hénon map then produced iterates of the form $x_{n+1} = A + \delta p_n^{\rm NN} - x_n^2 + Bx_{n-1}$.

The single output generated by the network $\delta p_n^{\rm NN}$ was compared to δp_n^H computed from Eq. (4.2). The error Q was computed upon each output of the network. When the OGY formula was used to lock onto the unstable period-1 orbit, the values of δp_n^H were chaotic and fluctuating around a mean absolute value of $\simeq 0.1 - 1.0 \times 10^{-3}$. With this in mind, the number of data points used to train the neural network was chosen so that the error (which is seen to follow a decaying exponential scaling with the number of input data points) was reduced to a value near or below 1.0×10^{-3}. This number was typically in the range of $10^3 - 10^4$ when Eq. (4.2) was utilized for training the neural network. Once trained, the perturbations generated by the neural network were used to perturb the time series upon each iteration of the Hénon map, not simply when the iterates fell within the usual controlling region specified by the OGY formula (see discussion of ξ_*^u in Sec. II above). The neural network produces unacceptably large perturbations when the iterates fall outside the controlling region. However, when an iterate ergodically falls within the controlling region, for which the neural network has been trained, it produces the perturbations necessary to achieve and maintain control. To prevent the large perturbations, which were sometimes generated when the Hénon iterates were far from the fixed points, from causing the time series to diverge to infinity, we applied only those perturbations which were less in magnitude than some predefined maximum value, $\delta p_{\rm max}$. This maximum was chosen to ensure that the applied perturbations did not kick the Hénon map outside of the chaotic regime. We typically used a value of $\delta p_{\rm max} = 0.05 - 0.10$. Figure 3 shows a comparison of δp_n^H and $\delta p_n^{\rm NN}$ generated by the neural network once training has been completed and a plot of the unstable period-1 orbit of the Hénon map controlled by the neural network. Note the three separate lines relating $\delta p_n^{\rm NN}$ to

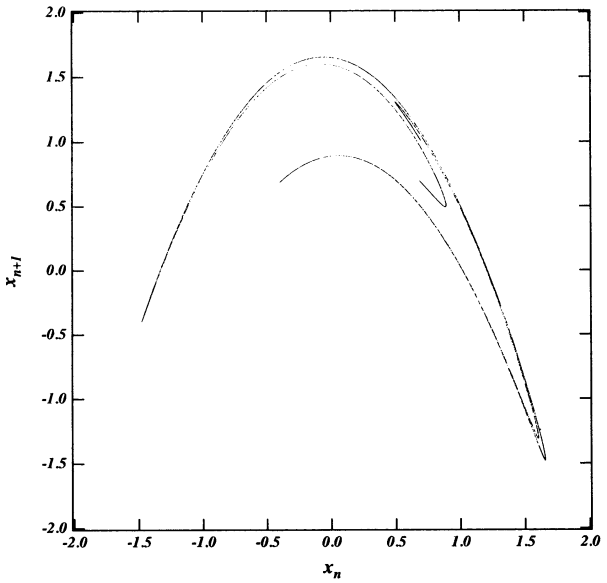

FIG. 2. The Hénon attractor generated from Eq. (4.1) with $A = 1.29$ and $B = 0.3$.

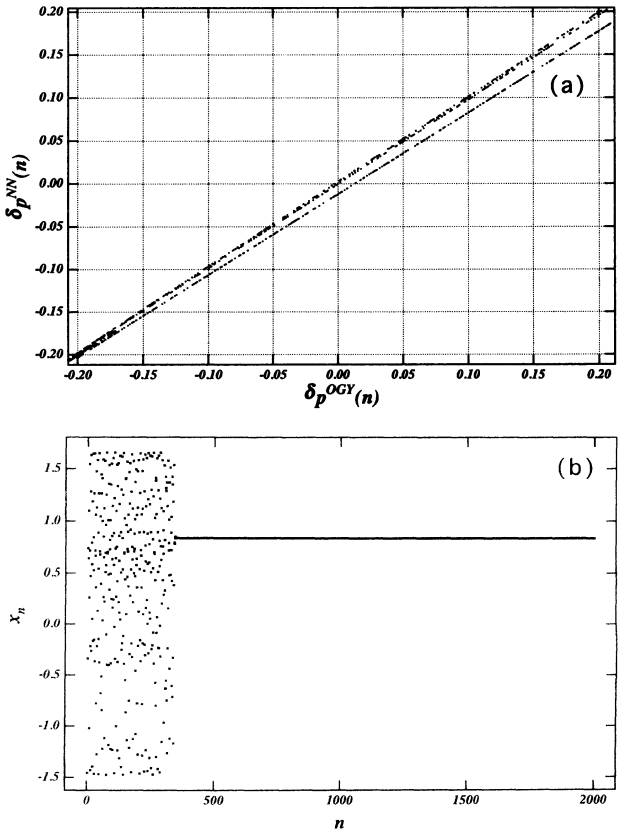

FIG. 3. (a) $\delta p_n^{\rm NN}$ vs δp_n^H for a neural network trained on Eq. (4.2) with inputs (x_n, x_{n-1}); (b) x_n vs n. Control was applied continuously by the neural network to x_n for $n > 100$, not just when $|x_n - x_F| < \epsilon \ll 1$.

δp_n^H in Fig. 3(a) for this number of training points. The error has been reduced in this case to $\simeq 0.01$, a factor of 10 over the criteria stated above. Even with an error this large, control can be achieved and maintained. The lines in Fig. 3(a) can be compressed into a single line passing through the origin by training on a larger set of input points.

There is a subtlety associated with the process of control performed by the neural network. As stated above, the network is trained on an unperturbed map; i.e., Eq. (4.2) is used for training the network, but during training, no perturbations are applied to the input time series. However, once the network is trained the mapping, Eq. (4.2), is altered by the applied perturbations generated by the network, $x_{n+1} = A + \delta p_n^{\rm NN} - x_n^2 + B x_{n-1}$. In essence, the fixed points of the altered map are shifting around due to the applied perturbations. Even though the neural net has not been trained on such an input set, it is still able to achieve control. During the route to controlling the chaotic signal, the perturbations generated by the neural network are shifted away from the values δp_n^H that would be obtained if Eq. (4.2) were used to control the Hénon map. However, when an iterate falls within the controlling region the difference between $\delta p_n^{\rm NN}$ and δp_n^H is generally much smaller than ξ_*^u. The neural network capitalizes on the fact that the OGY scheme is robust, and the small but finite differences between the values generated from the neural network and Eq. (4.2) can be treated as tolerable noise. Analysis of the ability of the neural network to maintain control in the presence of noise reveals that it is as tolerant of noise as the OGY formula, Eq.(4.2), upon which it was trained.

B. Training the neural network on a nonlinear function of the inputs

The neural network can be trained to achieve control utilizing a controlling scheme which involves the previous perturbation. In the following, we used a neural network with three inputs, one hidden layer consisting of six units and one output. During the training process, the inputs were $(x_{n-1}, x_{n-2}, \delta p_{n-1}^H)$ and the output was again $\delta p_n^{\rm NN}$. Once the network was trained, the inputs were switched to $(x_{n-1}, x_{n-2}, \delta p_{n-1}^{\rm NN})$ and again the perturbations were applied, as in Sec. IV A, to each iterate of the Hénon map during the controlling phase. Two training formulas were investigated, the usual OGY formula, Eq. (4.2), and

$$\delta p_n^H = \lambda_u \big(- \delta p_{n-1}^H + [1.840(x_{n-1} - x_F) - 0.300(x_{n-2} - x_F)] \big), \qquad (4.3)$$

where λ_u is the unstable eigenvalue of the local map $\boldsymbol{M}$ having the value -1.84 in the case of the Hénon map. Note that the usual OGY formula, Eqs. (2.1) and (4.2), uses the most current data pair (x_n, x_{n-1}) to form the perturbation δp_n^H. In this case, the third input port containing δp_{n-1}^H is idle since Eq. (4.2) is independent of this quantity. Equation (4.3) is an iterated form of the OGY formula, which uses the previous pair of iterates (x_{n-1}, x_{n-2}) to calculate δp_n^H. Because past information is used, Eq. (2.1) must be modified to include the previous perturbation δp_{n-1}^H. Here the third input port was definitely not idle and the weights and biases associated with it played a fundamental role in the training of the network. The factor of λ_u in Eq. (4.3) arises because the use of past data in the controlling formula involves using essentially the "square" of Eq. (2.1) in order to produce δp_n^H. This *delayed* OGY formula and its generalization to predict δp_n^H using past data of the form (x_{n-k}, x_{n-1-k}), where k is a small finite integer, is derived and discussed in detailed in [10]. It has potential applications for controlling chaos in dynamical systems which involve fast time scales [11].

For both training formulas, Eqs. (4.2) and (4.3), the neural network had to implicitly calculate the next iterate x_n of the Hénon map and then use that iterate to form $\delta p_n^{\rm NN}$. Again the networks were trained only in the vicinity of the fixed point. Figures 4(a) and 5(a) show plots analogous to Fig. 3(a) for the case of training with Eqs. (4.2) and (4.3), respectively.

In the process of controlling there is an important difference between the neural networks trained on the usual OGY formula and the delayed OGY formula. In the former case, the output of the neural network had to be adjusted according to $\delta p_n^{\rm NN} \to \delta p_n^{\rm NN} + 1.840\, \delta p_{n-1}^{\rm NN}$ in or-

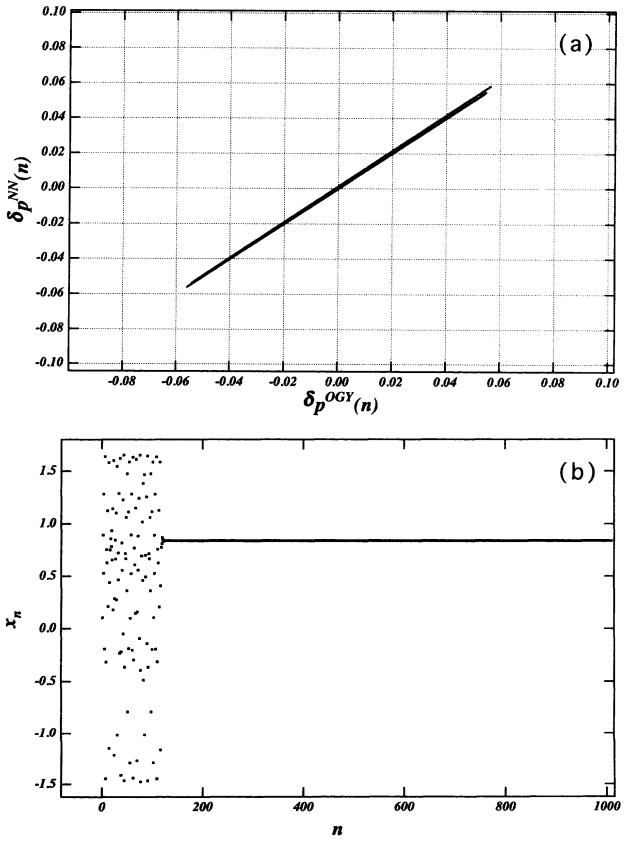

FIG. 4. (a) $\delta p_n^{\rm NN}$ vs δp_n^H for a neural network trained on Eq. (4.2) with inputs $(x_{n-1}, x_{n-2}, \delta p_{n-1}^H)$; (b) x_n vs n. Control was applied continuously by the neural network to x_n for $n > 100$, not just when $|x_n - x_F| < \epsilon \ll 1$.

der to achieve control. In the case of the delayed OGY formula the direct output of the neural network was capable of achieving control. This was true for all training sets, even when 10^7 points were used and the error function was reduced in magnitude to $\sim 0.5 \times 10^{-3}$.

The fundamental reason for this difference lies in the discussion concerning the output of the neural network during the training and controlling stage raised in Sec. IV A. Recall that the network is trained on an unperturbed map, Eq. (4.1), but during control the map is altered by the applied perturbations to $x_{n+1} = A + \delta p_n^{\mathrm{NN}} - x_n^2 + Bx_{n-1}$. Shifting indices from $n \to n-1$ and substituting this perturbed value of x_n into Eq. (4.2), yields to first order the correction $+1.840\,\delta p_{n-1}^{\mathrm{NN}}$. Thus the raw output of the neural network, which has been trained on a map where the fixed points are stationary, has to have this correction factor added to it when the applied perturbations begin shifting the fixed points around.

Such an adjustment to the output of the neural network trained on the delayed OGY equation (4.3) is not necessary. The reason for this is because during training the third input port receiving δp_{n-1}^{H} contains implicit information concerning the shifting of the fixed points. This is the case since the derivation of Eq. (4.3) explicitly makes use of the shift of the fixed point in going from the input pair (x_{n-1}, x_{n-2}) to the output point x_{n+1} (see [10]). When the value of the third input port is changed to $\delta p_{n-1}^{\mathrm{NN}}$ during the controlling process, there is still a constant shifting between the output of the neural network and Eq. (4.3). However, this shifting is now second order in $\delta p_{n-1}^{\mathrm{NN}}$, and therefore acts as tolerable noise. The price paid for using this formula is that the neural network is less robust to noise than if Eq. (4.2) were used for training, since the delayed OGY equation (4.3) is *intrinsically* less robust to noise. In [10] the authors show that the controlling region for the delayed OGY scheme is $|\lambda_u|$ times smaller than that of the controlling region for the usual OGY scheme.

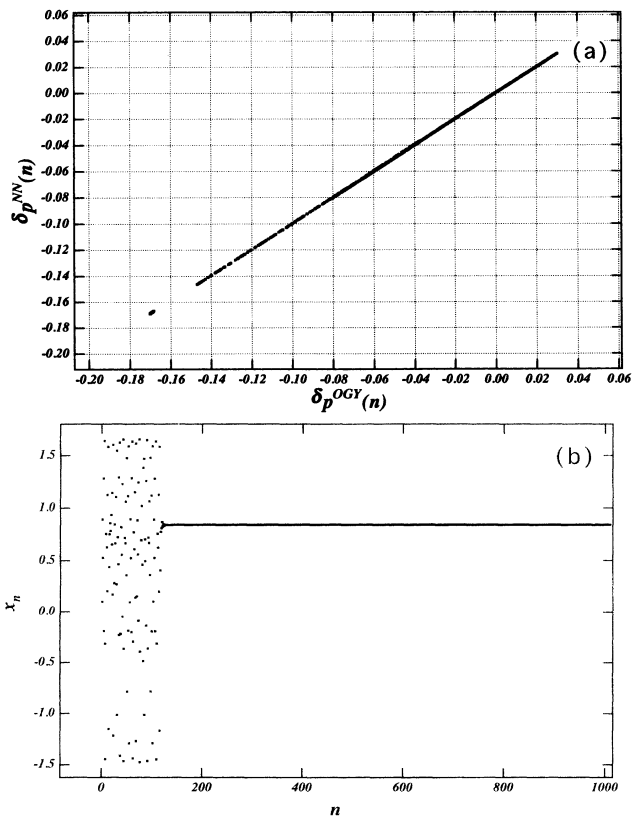

FIG. 5. (a) δp_n^{NN} vs δp_n^H for a neural network trained on Eq. (4.3) with inputs $(x_{n-1}, x_{n-2}, \delta p_{n-1}^H)$; (b) x_n vs n. Control was applied continuously by the neural network to x_n for $n > 100$, not just when $|x_n - x_F| < \epsilon \ll 1$.

C. Training the neural network on an experimental time series

In the previous two sections we have demonstrated how a neural network can be trained to produce the perturbations necessary to control the Hénon map for various types of input data. This may give the erroneous impression that one must *a priori* have the deterministic map in hand in order to train the neural network. However, this is not the case. As indicated in Sec. II, the information necessary for controlling an unstable periodic orbit of a chaotic attractor can be constructed from a single experimental time series of the uncontrolled system. Below we discuss how this can be used to train the neural network.

Suppose that the phase space of the chaotic dynamical system we wish to study is given by the vector $\boldsymbol{Z}(t)$, but one only has access to an experimental, scalar time series $x(t) \equiv H(\boldsymbol{Z}(t))$ (e.g., a current or voltage signal). The attractor, of yet unkown dimension, can be constructed from this scalar time series by the method of *delay coordinates* [5]. One can form the m-dimensional delay vector $\boldsymbol{X}(t) = (x(t), x(t-\tau), \ldots, x(t-(m-1)\tau))$, where the delay τ can be optimally chosen, for example, as the first minimum of the mutual information function of $x(t)$ (see [12]). The components of $\boldsymbol{X}(t)$ act as psuedo-phase-space variables from which a faithful reproduction of the attractor can be produced if m is larger than the dimension of the attractor. The dimension m can be found by standard procedures of computing correlation dimension from time series [13].

Once m is known we can produce a Poincaré section by defining t_n to be those times for which, say, $x(t) = x_0$, where x_0 is a constant, or say, at the peaks of $x(t)$ [i.e., $\dot{x}(t) = 0$ and $\ddot{x}(t) < 0$]. This produces an $(m-1)$-dimensional discrete-time vector $\boldsymbol{\xi}_n \equiv (x(t_n - \tau), x(t_n - 2\tau), \ldots, x(t_n - (m-1)\tau))$. For control, one needs the local mapping $\delta\boldsymbol{\xi}_{n+1} = \boldsymbol{M}\,\delta\boldsymbol{\xi}_n$ about the fixed point $\boldsymbol{\xi}_F$, where $\delta\boldsymbol{\xi}_n = \boldsymbol{\xi}_n - \boldsymbol{\xi}_F$. The local map $\boldsymbol{M}$ can be fitted numerically from the experimental data.

Because of the use of delay coordinates, the Poincaré map $\boldsymbol{F}$, relating $\boldsymbol{\xi}_{n+1}$ to $\boldsymbol{\xi}_n$, will depend on the previous value of the control parameter p_{n-1} as well as the current value p_n [7], $\boldsymbol{\xi}_{n+1} = \boldsymbol{F}(\boldsymbol{\xi}_n, p_n, p_{n-1})$. The OGY control formula is then a modified version of Eq. (2.1) in which δp_n depends on $\boldsymbol{\xi}_n$ and δp_{n-1}. For example, for $m = 3$, the OGY control formula will have the form $\delta p_n = \alpha(x_n - x_F) + \beta(x_{n-1} - x_F) + \gamma\delta p_{n-1}$, [7]. Here the constants $\{\alpha, \beta, \gamma\}$ can be determined from the local map $\boldsymbol{M}$, which

is the numerical fit to the Jacobian of $\boldsymbol{F}$. Once $\delta p_n^{\mathrm{OGY}}$ is known, the neural network can be trained to produce the required perturbations δp_n^{NN} to control the system with or without the presence of small amounts of noise.

In the above sections we used an *a priori* given, deterministic map, i.e., the Hénon map. This was done purely for numerical convenience. From the above considerations and, providing one generates a sufficient accurate local map, it is immaterial to the neural net if $\delta p_n^{\mathrm{OGY}}$ is generated analytically or numerically from an experimental time series. Thus given an experimental time series from a chaotic dynamical system, the necessary information can be numerically extracted to train the neural network to perform the control.

V. CONCLUSIONS

In this work we have demonstrated the feasibility for using neural networks to implement schemes for controlling chaos. By considering controlling schemes which depend linearly and nonlinearly on the inputs to the network, we have shown that neural networks can utilize their ability to predict chaotic time series to construct the perturbations necessary to achieve and maintain control. In previous efforts, neural networks have been trained over the whole range of the attractor in order to recognize geometrical quantities associated with the chaotic dynamical system. For the application of controlling chaos, the neural networks need only be trained on a set of points in the vicinity of the desired fixed point of the unstable periodic orbit. As far as the training of the neural network is concerned, we discussed how the training set can be of analytical origin or extracted numerically from an experimental time series.

In addition, we found that by limiting the output of the neural network to some predetermined maximum value, we could achieve control by letting the neural network operate continuously on the time series. This is to be contrasted with the usual controlling methodology in which perturbations are applied only in the vicinity of the fixed point. When the iterates lay outside the controlling region the neural network produced perturbations too large to be useful. However, when an iterate ergodically came within the controlling region of the fixed point, where the neural network had been trained, the network was capable of producing the perturbations necessary to control the signal. The time to control a chaotic signal using a neural network could be reduced by employing methods of targeting [14] to steer the time series into the vicinity of the fixed point more quickly.

In this work we concentrated on the feed-forward back-propagating neural networks commonly found in the literature. By using new fast-training algorithms based on matrix pseudoinverse methods [15], training sets and therefore training times can be drastically reduced. This is an important consideration for practical applications. Investigations into using fast-training neural networks for controlling large arrays of coupled chaotic systems are currently being explored by the present authors.

ACKNOWLEDGMENTS

One of the authors (V.K.) would like to thank the National Research Council for support of this work.

[1] T. Shinbrot, E. Ott, C. Grebogi, and J. Yorke, Nature **363**, 411 (1993).
[2] A. Lapedes and R. Farber, Los Alamos National Laboratory Technical Report No. LAUR-87-2662 (1987) (unpublished).
[3] A.M. Albano, A. Passamante, T. Hediger, and M.E. Farrell, Physica D **58**, 1 (1992).
[4] E. Ott, C. Grebogi, and J.A. Yorke, Phys. Rev. Lett. **64**, 1196 (1990).
[5] F. Takens, in *Dynamical Systems and Turbulence*, edited by D. Rand and L.S. Younge (Springer-Verlag, Berlin, 1981), p. 230.
[6] E. Ott, C. Grebogi, and J.A. Yorke, in *Chaos: Soviet-American Perspectives on Nonlinear Science*, edited by D.K. Campbell (AIP, New York, 1990), p. 153.
[7] U. Dressler and G. Nitsche, Phys. Rev. Lett. **68**, 1 (1992).
[8] D. Auerbach, C. Grebogi, E. Ott, and J.A. Yorke, Phys. Rev. Lett. **69**, 3479 (1992).
[9] P.J. Denning, Am. Sci. **80**, 426 (1992).
[10] P.M. Alsing, A. Gavrielides, and V. Kovanis, in *Chaos/Nonlinear Dynamics: Methods and Commercialization*, edited by H.S. Wisniewski, SPIE Proc. No. 2037 (SPIE, Bellingham, WA, 1993).
[11] A. Gavrielides, V. Kovanis, and P.M. Alsing, in *Chaos in Optics*, edited by R. Roy, SPIE Proc. No. 2039 (SPIE, Bellinghom, WA, 1993).
[12] H.D.I. Abarbanel, R. Brown, J.J. Sidorowich, and L.S. Tsimring, Rev. Mod. Phys. **65**, 1331 (1993).
[13] P. Grassberger and I. Procaccia, Physica D **9**, 189 (1983).
[14] T. Shinbrot, E. Ott, C. Grebogi, and J. Yorke, Phys. Rev. Lett. **65**, 3215 (1990); **68**, 2863 (1992).
[15] S.D. Pethel, C.M. Bowden, and C.C. Sung (unpublished).

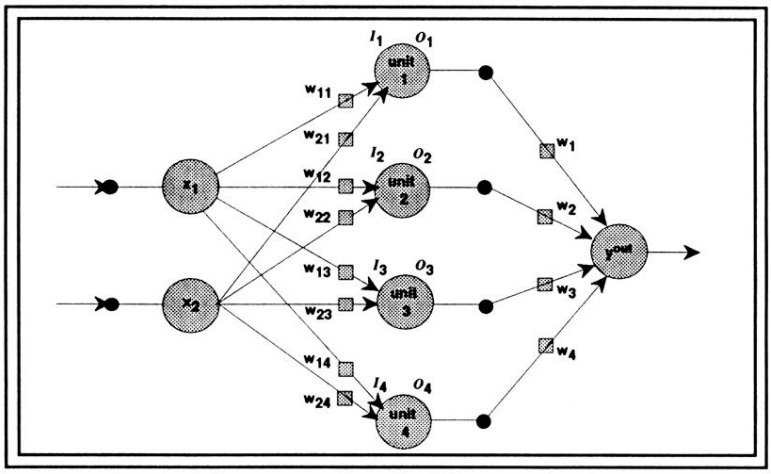

FIG. 1. A backpropagating neural network shown here with two inputs, one hidden layer consisting of four units, and one output.