Complex Systems 4 (1990) 603–622

Exploiting Neurons with Localized Receptive Fields to Learn Chaos

K. Stokbro
The Niels Bohr Institute,
Blegdamsvej 17, DK-2100 Copenhagen Ø, Denmark

D. K. Umberger
The Niels Bohr Institute and NORDITA,
Blegdamsvej 17, DK-2100 Copenhagen Ø, Denmark

J. A. Hertz
The Niels Bohr Institute,
Blegdamsvej 17, DK-2100 Copenhagen Ø, Denmark

Abstract. We propose a method for predicting chaotic time series that can be viewed either as a weighted superposition of linear maps or as a neural network whose hidden units have localized receptive fields. These receptive fields are constructed from the training data by a binary-tree algorithm. The training of the hidden-to-output weights is fast because of the localization of the receptive fields. Numerical experiments indicate that for a fixed number of free parameters, this weighted-linear-map scheme is superior to its constant-map counterpart studied by Moody and Darken. We also find that when the amount of data available is limited, this method outperforms the local linear predictor of Farmer and Sidorowich.

1. Introduction

Interest in the problem of predicting the evolution of time series has grown rapidly over the last few years [1–6]. One reason for this is that the problem is extremely general; new results coming from its study have the potential for application in *any* field that involves working with sequences of measurements made on nonlinear dynamical processes. The generality of the problem can be appreciated from its very statement: "learn" the equations of motion for the evolution of some observable of a system given some knowledge of that observable's past history. If a system undergoing some motion has an observable y that is measured in an experiment to produce a time series $\{y(t_i) : i = 1, \ldots, N\}$, the problem is to construct, directly from the time

series, a dynamical system that predicts $y(t)$ for $t > t_N$. As formulated, this problem makes no reference to any explicit type of motion, but the cases of most interest involve time series generated by chaotic processes [7]. The reason for this is that chaos represents the worst case scenario for prediction. When a motion is chaotic, by definition, long-term predictions are impossible to make, even when the equations of motion are known exactly. This is because *any* error in the specification of an initial condition will grow exponentially fast with time on average. Though long time predictions are impossible to make for such motions, short time predictions are possible. Thus, if a method is capable of predicting chaotic time series, it should also apply to simpler cases, such as when the underlying motion is periodic or quasi-periodic.

Not surprisingly, the recent interest in the prediction problem originates in the study of dissipative nonlinear systems, where there has been much emphasis placed on developing techniques for analyzing experimental, nonlinear systems. This emphasis has arisen from the natural desire to make contact between theory and experiment: How can one measure quantities that characterize chaotic motion when only sequences of measurements of a few observables are available? This question led to the development of a technique called phase space reconstruction, introduced by Packard et al. [8] and Takens [9], which can be used to construct a state space directly from observables. The version of this technique most commonly used is formulated in terms of *delay coordinates*: the coordinates of the phase space are the values of the observables measured at equally spaced time intervals. The basic time interval is called the delay time τ, and for an observable y state vectors have the form $\mathbf{x}(t) = (y(t), y(t-\tau), \ldots, y(t-(d_E-1)\tau))$ where d_E, called the embedding dimension, is the dimension of the reconstructed phase space. For a dissipative system having an attractor of dimension d one would expect d_E to be of the order d, and this turns out to be the case. Specifically, Takens [9] proved that d_E needs at most to be $2d+1$. In this formulation, then, our problem is to fit a map that gives, say, $y(t+T)$ as a function of $\mathbf{x}(t)$.

An approach to predicting the full, nonlinear motion in a reconstructed phase space, relevant to this paper is that of Farmer and Sidorowich [1, 2]. They used what they call a "local" approximation scheme [10]. In such a scheme a number of maps, each corresponding to a different region of the reconstructed phase space, are built from a set of examples of state vectors $\mathcal{T} = \{\mathbf{x}(t_m)\}$ and their images (what the vectors evolve into), as obtained from the time series. The resulting global map is thus piecewise continuous and becomes a better and better approximation to the true map as the number of stored data points increases. This method is very effective in terms of computation time because the local maps need not all be computed; those which are needed can be fit "on the fly": Suppose the state of the system at time t is $\mathbf{x}_0$ and the value of y at time $t+T$ is to be predicted. The set $\mathcal{T}$ is searched for some prespecified number of vectors closest to $\mathbf{x}_0$, and a map is built by least-squares fitting from these neighbors and their

images. (The choice of functional form or "representation" for the local maps is arbitrary, though in references [1, 2] the authors worked primarily with linear and quadratic maps.)

A very different approach has been proposed by Lapedes and Farber [5]. They utilized a standard feedforward neural network (a perceptron) trained with backpropagation to produce a "global" predictor. The method is a "global" method in the sense that a single function, valid for the entire reconstructed phase space, is built and composed with itself to predict a segment of a system's trajectory. The method yields good predictions using smaller amounts of training data than the Farmer-Sidorowich method, but because of the inefficiencies inherent in the backpropagation algorithm it requires much greater computing time.

Another method, intermediate in spirit between the above two, was introduced by Casdagli [4]. In some implementations it uses an expansion of the global map in terms of localized radial basis functions. A localized radial basis function is an isotropic multi-dimensional function that has a peak at some location in the phase space, called a center, and vanishes as the distance from this location increases. The global function is expressed as a linear combination of basis functions centered at different points distributed around the input space, and the values of the coefficients in the expansion are determined by least-squares fitting.

Viewed as a neural net, this is a system with a single hidden layer. Each hidden unit corresponds to one of the localized basis functions; its activation is the value of that basis function at the point defined by the input vector $\mathbf{x}_0$. There is a single (linear) output unit, and the hidden-to-output connection weights are just the coefficients in the expansion of the map in the radial basis functions. One can say that each hidden unit has a "receptive field" localized in some region of the input space. (Thus such a unit is quite different from a conventional sigmoidal one, which responds maximally to inputs in a particular half-space.) A very similar kind of network was also proposed in another context by Specht [11].

In a modified version of this scheme proposed by Moody and Darken [6], the centers of the radial basis functions are "learned" from a training set by an unsupervised learning algorithm (discussed in section 2). Apart from this, the main difference between their method and Casdagli's is in the choice of the form of the basis functions used: the Moody-Darken functions are more localized in the input space than those used by Casdagli.

In this paper we explore another scheme that utilizes basis functions that are localized in the input space. It uses elements of the Farmer-Sidorowich algorithm as well as those of the Moody-Darken method. It can be viewed as a streamlined extension of the latter, requiring much less time to implement and yielding much better prediction accuracy. The result is a highly efficient prediction machine implementable on parallel or serial hardware. The method is described in the following section, which begins with a review of the Moody-Darken predictor. The results of a number of numerical experi-

ments comparing its performance to those of other methods are presented in the succeeding section, which is then followed by our conclusions.

2. Method

The Moody-Darken [6] method employs a set of normalized radial basis functions to build a global mapping f that approximates the dynamics of a time series. This mapping has the form

$$f(\mathbf{x}) = \frac{\sum_{\alpha=1}^{M} f^{\alpha} R^{\alpha}(\mathbf{x})}{\sum_{\alpha=1}^{M} R^{\alpha}(\mathbf{x})} \tag{2.1}$$

where

$$R^{\alpha}(\mathbf{x}) = \exp[-(\mathbf{x} - \mathbf{x}^{\alpha})^2 / 2{\sigma_{\alpha}}^2], \tag{2.2}$$

the $\mathbf{x}^{\alpha}$ are d_E-dimensional vector parameters, and the σ_{α} and f^{α} are scalar parameters. The parameters $\mathbf{x}^{\alpha}$ and σ_{α} are, respectively, the center and the width of the response function of the αth hidden unit, and f^{α} is the connection weight from this unit to the output unit. Bringing the denominator of the right-hand side of equation (2.1) into the sum of the numerator, we see that the αth hidden unit has the overall response function

$$P^{\alpha}(\mathbf{x}) = \frac{R^{\alpha}(\mathbf{x})}{\sum_{\beta=1}^{M} R^{\beta}(\mathbf{x})}. \tag{2.3}$$

In reference [6] the parameters $\mathbf{x}^{\alpha}$, σ_{α}, and f^{α} are calculated in three successive steps. First the M vectors $\mathbf{x}^{\alpha}$ are computed using a set of vectors obtained from a time series. The idea is to distribute the $\mathbf{x}^{\alpha}$ according to the natural measure of the attractor, where the density of data points is high so is the density of the $\mathbf{x}^{\alpha}$. The authors accomplish this by using an adaptive formulation of an algorithm called the "k-means clustering algorithm" [6] to obtain the $\mathbf{x}^{\alpha}$, and refer to them as "cluster centers." (We will use this terminology throughout the paper.)

Once cluster centers have been constructed, the parameters σ_{α} are calculated by setting

$$\sigma_{\alpha} = \frac{1}{\sqrt{2}} \langle (\mathbf{x}^{\alpha} - \mathbf{x}^{\beta})^2 \rangle_p^{1/2} \tag{2.4}$$

where the average is over the p cluster centers closest to $\mathbf{x}^{\alpha}$. (Note: p is a new parameter.) Although this is the prescription given, in reference [6] a uniform, global value of σ_{α} is used in all their examples. This value is the average of equation (2.4) over all cluster centers.

Having obtained the widths, the weights f^{α} are adjusted by using a set of input vectors $\{\mathbf{x}_i : i = 1, \ldots, N\}$ and their known images $\{y_i : i = 1, \ldots, N\}$ to minimize the cost function

$$E_f = \frac{1}{2} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2. \tag{2.5}$$

The representation defined in equation (2.1) can be thought of as arising from a weighted combination of "constant maps." (A constant map is a mapping whose image has the same value irrespective of its argument.) To motivate this idea, imagine that a set of N training vectors $\mathcal{T} = \{\mathbf{x}_i\}$ and their known images have been obtained directly from a time series. Now suppose that the predicted image of an arbitrary input vector $\mathbf{x}$ is taken to be the image of the vector in $\mathcal{T}$ closest to $\mathbf{x}$. Such a predictor is a "nearest-neighbor" predictor. Suppose we don't want to lug around all N vectors of $\mathcal{T}$ with which to make the predictions. We can introduce another set of $M << N$ vectors $\mathcal{C} = \{\mathbf{x}^\alpha\}$ distributed according to the natural measure of the attractor (for example, by using the k-means clustering algorithm of reference [6]). We can then associate an "image" f^α with each $\mathbf{x}^\alpha \in \mathcal{C}$ by searching $\mathcal{T}$ for some number of nearest neighbors of $\mathbf{x}^\alpha$ and setting f^α equal to the average of the images of these nearest neighbors. Now the predicted image of an arbitrary vector $\mathbf{x}$ can be defined to be the image f^α associated with the $\mathbf{x}^\alpha$ closest to $\mathbf{x}$, and we have what we call a "constant" predictor. This winner-take-all predictor is bound to have difficulties. Suppose, for example, that $\mathbf{x}$ lies between two vectors $\mathbf{x}^\beta$, $\mathbf{x}^\gamma \in \mathcal{C}$, but is slightly closer to one than the other. The continuity of the dynamics assures us that the image of $\mathbf{x}$ will be intermediate between f^β and f^γ, so why not incorporate this into the predictor? Of course, the closer $\mathbf{x}$ is to $\mathbf{x}^\beta$, the closer we expect its image to be to f^β, so it makes sense to take the image of $\mathbf{x}$ to be some kind of weighted average of f^β and f^γ. This is what the representation equation (2.1) accomplishes.

Of course, we could stick with a winner-take-all predictor, but improve the scheme above by replacing the constant mappings that yield the f^α by something more accurate, like a linear mapping. Thus we could use the vectors of $\mathcal{T}$ that are the nearest neighbors of $\mathbf{x}^\alpha$ to construct a linear map, in effect making the substitution $f^\alpha \to a^\alpha + \mathbf{b}^\alpha \cdot \mathbf{x}$ for the image associated with it. Now to make a prediction on an arbitrary vector $\mathbf{x}$ we find the vector of $\mathcal{C}$ closest to it, call it $\mathbf{x}^\beta$, and use $a^\beta + \mathbf{b}^\beta \cdot \mathbf{x}$ as the predicted image of $\mathbf{x}$. Note that this recipe is similar to that of Farmer and Sidorowich [1, 2]. We still have a problem when an input vector is in a region between two vectors of $\mathcal{C}$; neither mapping will achieve the desired result. However, we can again apply a weighted average as we did for the constant predictor. This time, though, the average is an average of *linear maps*. The result of all this is a representation that looks like

$$f(\mathbf{x}) = \sum_{\alpha=1}^{M} (a^\alpha + \mathbf{b}^\alpha \cdot \mathbf{x}) P^\alpha(\mathbf{x}) \tag{2.6}$$

where $P^\alpha(\mathbf{x})$ is given by equation (2.3). (Note that the choice of replacing the constants f^α by linear maps is arbitrary. Any polynomial, rational function, or other local representation valid for the Farmer-Sidorowich method can be used instead.)

We can still view equation (2.6) as arising from a neural network. This network still has d_E linear inputs and one linear output, but now each hidden

unit is replaced by $d_E + 1$ units, corresponding to the extra terms associated with the linear map. This interpretation of equation (2.1) shows that it is possible to implement our prediction algorithm on parallel hardware, thus making high speed predictions possible. However, the implementation we now describe, tailored for serial hardware, is also quite fast.

As is the case with the Moody-Darken scheme, we obtain equation (2.6) in three steps. The cluster centers $\mathbf{x}^\alpha$ are determined first, and from these the cluster widths σ_α are calculated. Finally, the parameters a^α and $\mathbf{b}^\alpha$ are determined.

We want the distribution of cluster centers $\mathbf{x}^\alpha$ to be roughly proportional to the natural measure of the attractor. Suppose we have obtained a set of vectors $\mathcal{T}$ from some time series. The vectors in the set represent points on the attractor, and their density approximates the attractor's natural measure. If we cover the attractor with a set of boxes such that all the boxes contain approximately the same number of vectors there will be many boxes in regions where the natural measure is high and few boxes in regions where the natural measure is low. We can then associate a cluster center with each box: set $\mathbf{x}^\alpha$ equal to the average of all vectors in the αth box. The resulting set of cluster vectors will have the desired distribution, provided that the partitioning of $\mathcal{T}$ is done in some reasonable way.

The method of partitioning *can* be crucial: note that there may be several ways of covering the vectors of $\mathcal{T}$ in the manner just described. Some of these "coverings" may be undesirable for our purposes. For example, a given covering may have a large number of boxes with most of their vectors lying near their boundaries, with large unpopulated regions in their interiors. Such boxes would have cluster centers in regions where the natural measure is close to zero, which is precisely what we *don't* want. A more suitable cover would be one in which the spread of the vectors in each box is kept relatively small.

Such a partitioning of $\mathcal{T}$ is accomplished as follows. At the first stage of construction we divide $\mathcal{T}$ into two disjoint subsets $\mathcal{L}$ and $\mathcal{R}$. The vectors in $\mathcal{T}$ have components labeled by an index j, which runs from 1 to d_E. For a fixed value of j we put the vectors of $\mathcal{T}$ in ascending order according to the values of their jth components. Denoting the reordered vectors by $\mathbf{X}^\gamma$ with γ an ordering index and letting X_j^γ be the jth component of $\mathbf{X}^\gamma$, the ordering is defined by $X_j^{\gamma-1} < X_j^\gamma < X_j^{\gamma+1}$. Now, for some integer n between 1 and N we divide $\mathcal{T}$ into two subsets defined as $\mathcal{L}_n^j = \{\mathbf{X}^\gamma : \gamma = 1, \ldots, n\}$ and $\mathcal{R}_n^j = \{\mathbf{X}^\gamma : \gamma = n+1, \ldots, N\}$. The vectors in $\mathcal{L}_n^j$ and $\mathcal{R}_n^j$ will have means $\langle \mathbf{X} \rangle_{\mathcal{L}_n^j}$ and $\langle \mathbf{X} \rangle_{\mathcal{R}_n^j}$, respectively, and the sum of the corresponding mean-square deviations from these means will be $\xi_n^j = \langle (\mathbf{X} - \langle \mathbf{X} \rangle_{\mathcal{L}_n^j})^2 \rangle_{\mathcal{L}_n^j} + \langle (\mathbf{X} - \langle \mathbf{X} \rangle_{\mathcal{R}_n^j})^2 \rangle_{\mathcal{R}_n^j}$. We choose $\mathcal{L}$ and $\mathcal{R}$ to be the sets $\mathcal{L}_n^j$ and $\mathcal{R}_n^j$ for which ξ_n^j is a minimum over all values of j and n. Next, we divide the daughter sets $\mathcal{L}$ and $\mathcal{R}$ of $\mathcal{T}$ in the same manner as we did $\mathcal{T}$ to produce two new daughters a piece. These daughters are then divided, and so on, until the resulting subsets have less than some prespecified number of vectors in them, whence the process is stopped. Such a partitioning of $\mathcal{T}$ can be accomplished recursively by using a

binary tree structure appropriate for ordering and storing multi-dimensional objects. Such a structure is called a "k-d tree" [12, 2]. (The term "k-d tree" stands for k-dimensional tree. In our case k= d_E, the embedding dimension.) The desired subsets of $\mathcal{T}$ are stored in the terminal nodes of the tree called buckets.

Once the partitioning of $\mathcal{T}$ has been done we calculate the mean vector for each of its subsets as stored in the buckets. The αth cluster center is stored in the αth bucket. We then apply equation (2.4) to obtain the widths that, like the $\mathbf{x}^\alpha$, are stored in their respective buckets. Note that the search for nearest neighbors can be accomplished quickly and efficiently since the cluster centers are stored in a binary tree.

All that remains now is to determine the parameters a^α and $\mathbf{b}^\alpha$ of equation (2.6) by adjusting them in order to minimize the cost function given by equation (2.5). To do this we replace equation (2.6) by the following, equivalent form

$$f(\mathbf{x}) = \sum_{\alpha=1}^{M} \left(a^\alpha + \mathbf{b}^\alpha \cdot \frac{(\mathbf{x} - \mathbf{x}^\alpha)}{\sigma_\alpha} \right) P^\alpha(\mathbf{x}) \tag{2.7}$$

in equation (2.5). The reason for this change is that we have found that it results in more stable numerics during the iterative minimization of the training process.

Having a good initial guess for the values of the parameters a_α, $\mathbf{b}_\alpha$ will reduce the amount of training necessary. It is natural to think of these parameters as being associated with the αth cluster center, or, equivalently, the αth bucket of the tree. By associating a linear map with the αth bucket, we initialize the parameters of the αth bucket by fitting a linear map to the subset of $\mathcal{T}$ associated with that bucket and its corresponding images. Thus, a_α, $\mathbf{b}_\alpha$ are initialized by minimizing

$$E_\alpha = \frac{1}{2} \sum_i \left[y_i - \left(a^\alpha + \mathbf{b}^\alpha \cdot \frac{(\mathbf{x}_i - \mathbf{x}^\alpha)}{\sigma_\alpha} \right) \right]^2 \tag{2.8}$$

where the sum is taken over the vectors of $\mathcal{T}$ in the bucket. The local linear method of references [1, 2] are thus used to initialize our predictor.

The usual way of minimizing the cost function equation (2.5) is by using the method of gradient descent. Let us collectively relabel the parameters a^α and $\mathbf{b}^\alpha$ by c_k so that if there are M cluster centers the index k runs from 1 to $(d_E + 1)M$. In each iteration the c_k are changed by an amount

$$\Delta c_k = \eta (y_i - f(\mathbf{x}_i)) \frac{\partial f(\mathbf{x}_i)}{\partial c_k} \tag{2.9}$$

where η is some small number called the learning rate. We have found that this method converges very slowly.

Instead we use a method where the full energy function is used in each iteration step. The functions $P^\alpha(\mathbf{x})$ and $\frac{(\mathbf{x}-\mathbf{x}^\alpha)}{\sigma_\alpha}P^\alpha(\mathbf{x})$ are collectively relabeled as $\Phi_k(\mathbf{x})$ where the index k runs again from 1 to $(d_E+1)M$. Rewriting equation (2.7) as

$$f(\mathbf{x}) = \sum_k c_k \Phi_k(\mathbf{x}) \tag{2.10}$$

the cost function equation (2.5) is the quadratic form

$$\left[\frac{1}{2}\sum_i y_i^2\right] - \sum_k c_k \left[\sum_i y_i \Phi_k(\mathbf{x}_i)\right] + \frac{1}{2}\sum_{k,l} c_k c_l \left[\sum_i \Phi_k(\mathbf{x}_i)\Phi_l(\mathbf{x}_i)\right] \tag{2.11}$$

Note that the sums over i in the three terms of expression (2.11) form a constant, a vector, and a matrix, respectively. These parameters need only be computed once for a fixed training set. There is no further need to store the training data. Also, if more training data become available they can simply be added. Thus the training scheme is adaptive. Most of the computer resources in terms of speed and storage are spent on the matrix during training. But the matrix is symmetric, and most of its elements are negligible, so it is necessary to store and compute only a small fraction of them.

The literature on methods to minimize quadratic forms is extensive. We have chosen a conjugate gradient descent method [13], which we have found to be very effective. The CPU time needed to find the minimum of E_f using this method is of the order of the time spent on calculating the matrix part of the cost function of expression (2.11).

The method of building the function of equation (2.6), just described, can also be applied to the representation of equation (2.1). To save space in writing, henceforth we will call the representation of equation (2.6) the Weighted Linear Predictor (WLP) and the representation of equation (2.1) the Weighted Constant Predictor (WCP).

3. Numerical Results

In this section we present the results of some numerical experiments used to test the representation equation (2.7). The first set of experiments are done to compare the performance of the WLP with that of the WCP. The WCP is a streamlined version of the method of Moody and Darken [6]; both methods utilize the same functional representation equation (2.1) and nearly the same distribution of the cluster centers $\mathbf{x}^\alpha$. Thus, direct comparisons of the WLP and the WCP yield indirect, but meaningful comparisons of the WLP and the Moody-Darken algorithm, except when computational speeds are involved (the WCP is roughly 2 to 3 times faster than the Moody-Darken algorithm). Of course, to make the comparisons fair we will always use the same number of free parameters for the WLP and the WCP.

The second set of numerical experiments compare the WLP with the local linear predictor (LLP) of Farmer and Sidorowich [1, 2]. The purpose of these

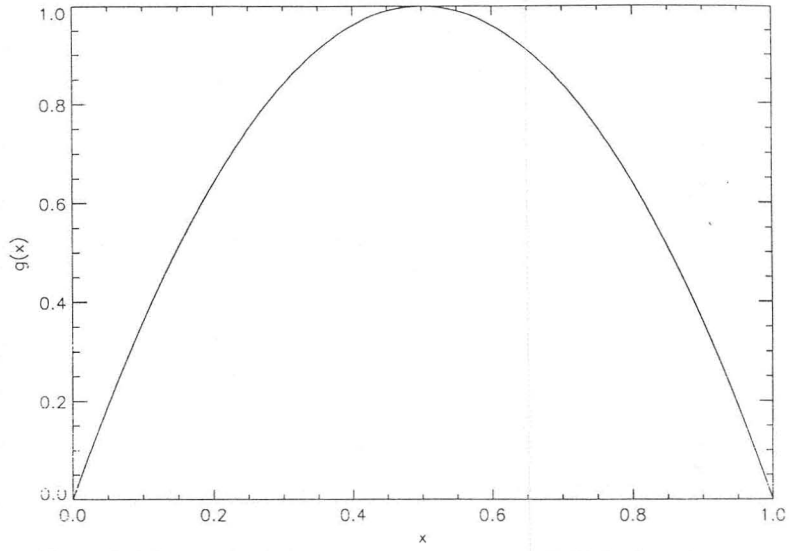

Figure 1: The graph of the logistic map equation (3.2) for $\lambda = 4$.

experiments is to check whether the WLP is a reasonable alternative to the LLP when the amount of data available is limited. (Although the LLP may be the "method of choice" when large amounts of data are available, it can break down when this is not the case.)

In all of the experiments presented below, we generate a time series from some dynamical system and divide it into two parts. One part, called the training set, is used to build the predictors. The second part, called the test set, is used to test the performance of the resulting predictors. The prediction accuracies are measured in terms of the mean normalized error, or simply the prediction error, defined as

$$\bar{E}^2 = \frac{\langle (y(t+T) - f(\mathbf{x}(t))^2 \rangle}{\langle y(t) - \langle y(t) \rangle)^2 \rangle} \tag{3.1}$$

where the averages are over the test set. To obtain good statistics, we will always use 1000 input-output pairs to evaluate this quantity.

We begin with a study of a one-dimensional mapping, the logistic map, defined by

$$x_{n+1} = g(x_n) = \lambda x_n (1 - x_n) \tag{3.2}$$

where λ is a parameter. The graph of equation (3.2) is a parabola, and λ controls its height. When $\lambda = 4$ for almost any initial condition x_0 in the interval $[0, 1)$ the iterates of equation (3.2) form a chaotic sequence confined to that interval. The graph of equation (3.2) for this case is shown in figure 1.

Can the WLP approximate the graph of figure 1 better than the WCP? To answer this, we picked an initial condition at random from the unit interval,

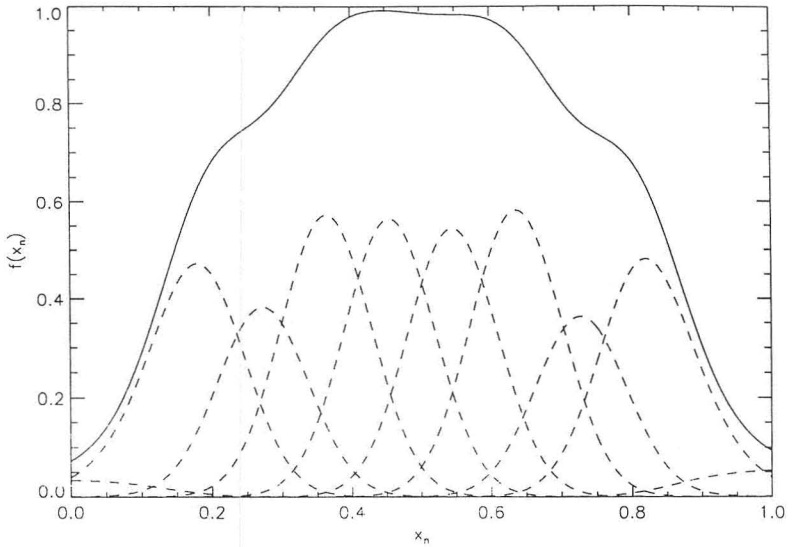

Figure 2: The WCP of equation (2.1) that results from training on a set of 200 input-output pairs that were produced by iterating equation (3.2). Ten cluster centers were used, and a prediction error of 0.118 was obtained. The dashed curves represent the individual terms in equation (2.1)

iterated the map until transients died out, then produced a time series of 200 elements to obtain the training set. For simplicity, we chose $d_E = 1$ and the cluster centers to be equally spaced in the interval. Also, the widths of the Gaussians were taken to be the distance between the cluster centers. Figure 2 shows the function produced by the WCP (solid curve) when 10 cluster centers are used. Each dashed curve in the figure corresponds to a single term in equation (2.1). Note that the function learned by this predictor is "bumpy." This should be compared to figure 3, which shows the function learned by the WLP when only five cluster centers are used. The "bumps" do not appear. This suggests that the WLP yields a smoother interpolation between the points of the training set.

To see how well the WCP approximates equation (3.2) we superimpose the function of figure 2 on its graph. This is shown in figure 4. This should be compared with figure 5, which is for the WLP of figure 3. The prediction errors corresponding to the figures were found to be 0.118 and 0.005, respectively. Thus, with the same number of free parameters the WLP makes a one-step prediction that is more than an order of magnitude better than the WCP.

Figures 6 and 7 are similar to figures 2 and 3 except that the number of cluster centers used for the WCP and the WLP have been increased to 60 and 30, respectively. For this case the prediction error for the WLP is 6.8×10^{-5},

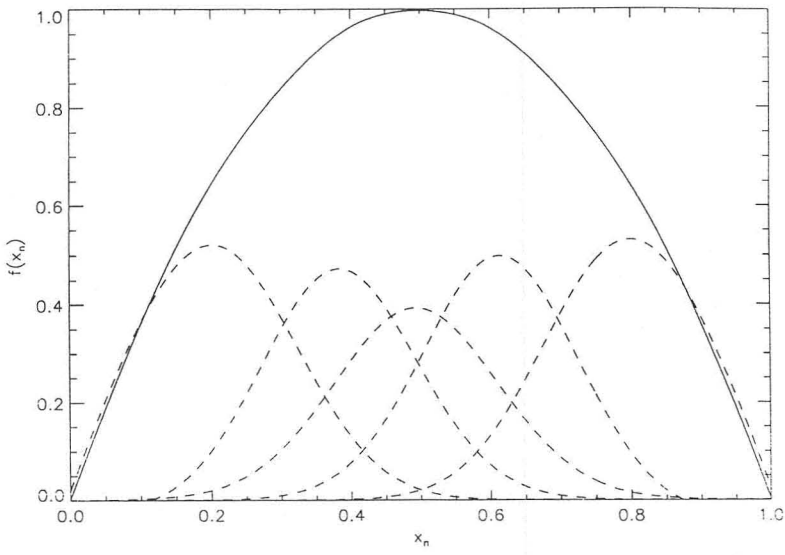

Figure 3: Similar to figure 2, but using the WLP of equation (2.6) with five cluster centers. A prediction error of 0.005 was obtained, showing that with the same number of parameters the WLP beats the WCP by more than an order of magnitude in the prediction error. The dashed curves represent the individual terms in equation (2.6). Note: we have not drawn portions of them that are below $f(x_n) = 0$.

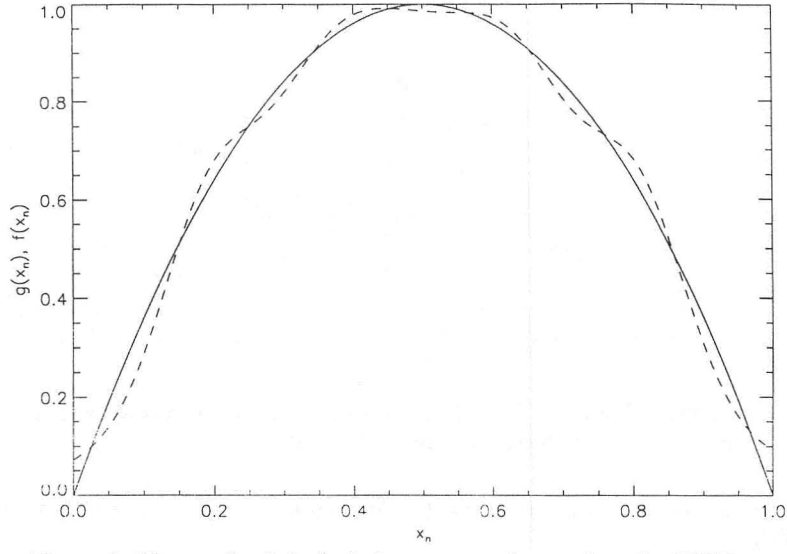

Figure 4: The graph of the logistic map superimposed on the WCP of figure 2.

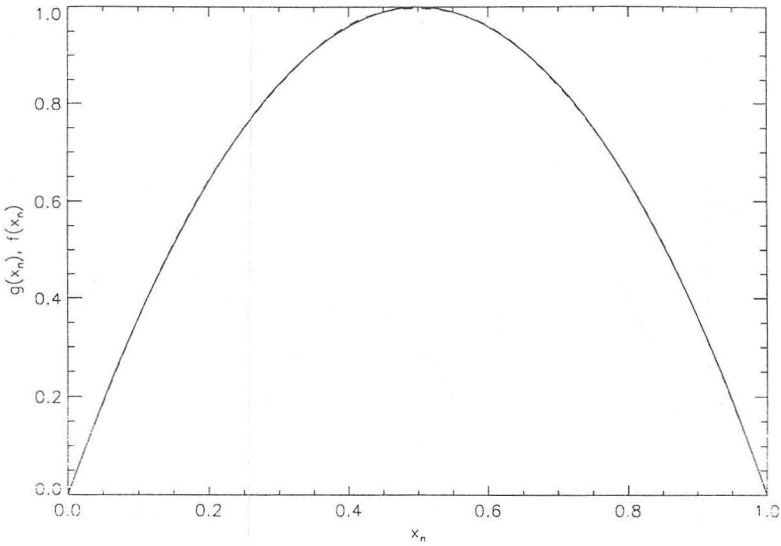

Figure 5: Similar to figure 4, but with the WLP of figure 3.

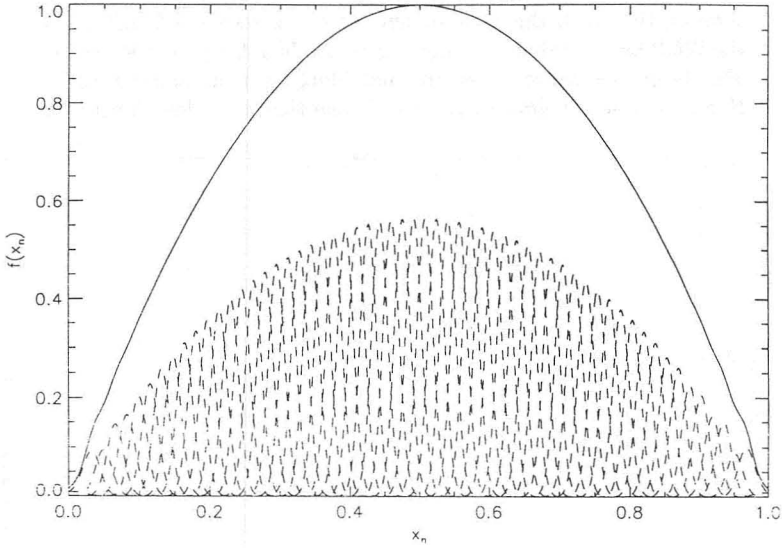

Figure 6: Similar to figure 2, except that 60 cluster centers were used in the WCP. The prediction error is 1.3×10^{-2}.

which is two orders of magnitude better than the value of 1.3×10^{-2} for the WCP.

In the previous section equation (2.6) was viewed as a weighted average of linear maps, where each map was interpreted as approximating the dynamics

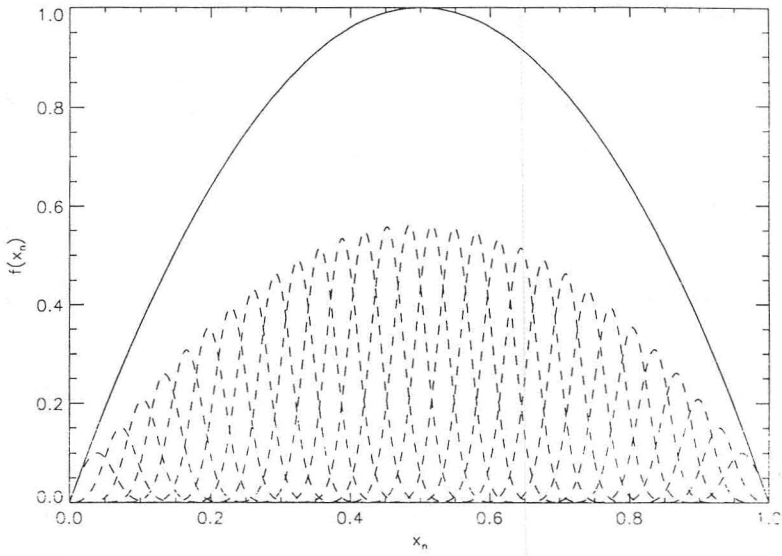

Figure 7: Similar to figure 3, except that 30 cluster centers were used in the WLP. The prediction error is 6.8×10^{-5}, which is more than two orders of magnitude better than the WCP with the same number of parameters.

in the vicinity of its cluster center. If this is the case, we expect that the derivatives of the function being approximated will be closely correlated with the slopes of the individual linear maps forming the distribution. Figures 8 and 9 show the graph of equation (3.2) together with the linear maps corresponding to figures 3 and 7, respectively. The validity of the interpretation appears to get better as the cluster centers become more numerous. Experiments on other one-dimensional maps support this conclusion. However, it is important to note that in cases where we have used small numbers of cluster centers we have observed instances where the derivatives and slopes fail to support this interpretation, although the WLP approximates the corresponding mappings well.

To make further comparisons between the WCP and the WLP, we consider time series generated by the somewhat less trivial Mackey-Glass delay-differential equation:

$$\frac{dx}{dt}(t) = -.1x(t) + .2\frac{x(t-\Delta t)}{1+x(t-\Delta t)^{10}} \tag{3.3}$$

where Δt is a parameter. This system has an infinite-dimensional phase space; its initial state must be specified over an interval of time. However, it does have low-dimensional attractors whose dimensions increase with Δt [14]. This system has been used to test a number of previously proposed prediction algorithms.

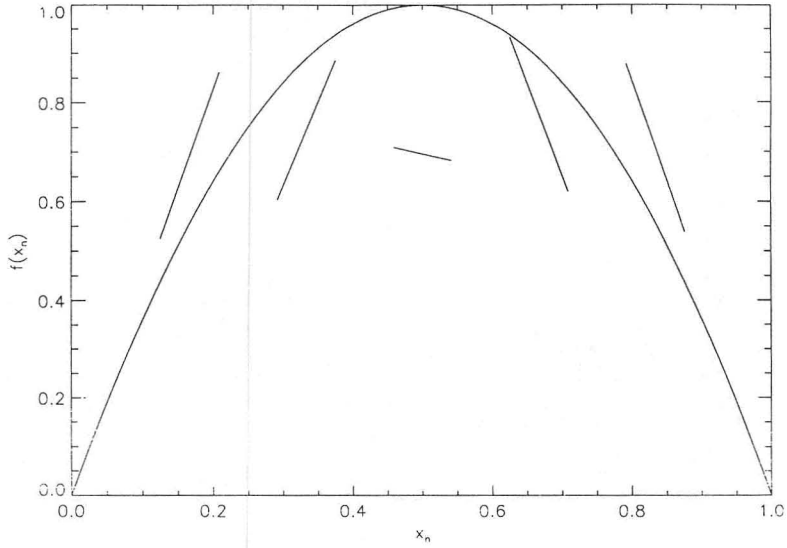

Figure 8: The graph of the logistic map with the individual linear maps of the WLP for the situation of figure 3. Note that the slopes of the linear maps match the derivatives of the logistic map to some extent.

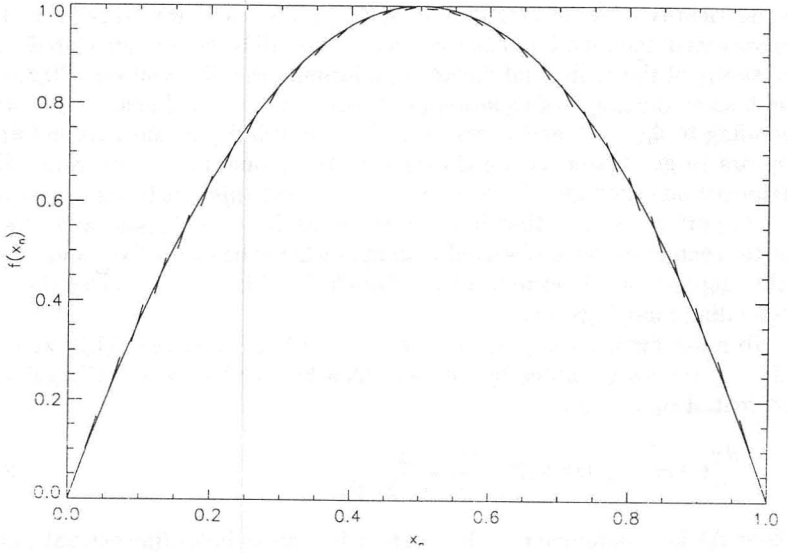

Figure 9: Similar to figure 8, but with 30 cluster centers. Note that slopes and derivatives match better when more cluster centers are used.

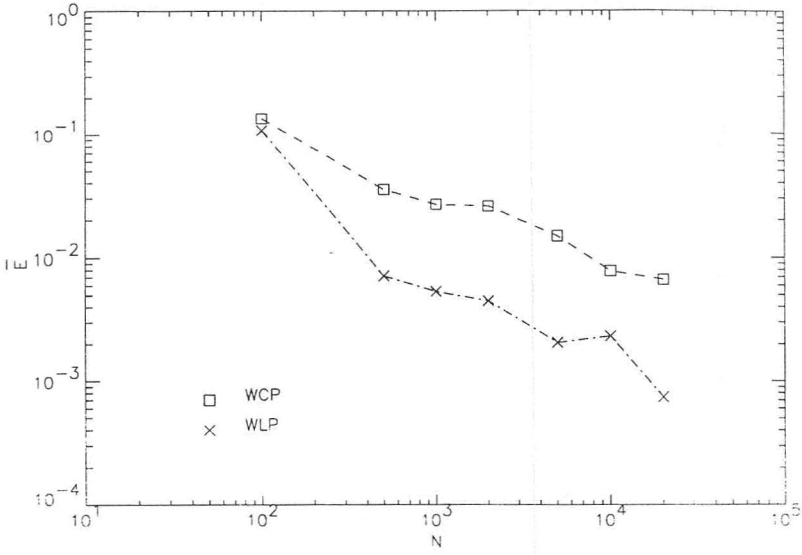

Figure 10: The prediction error, using time series from equation (3.3) with $\Delta t = 17$, for predicting $T = \tau = 6$ into the future as a function of the number of training points used.

We consider the case of $\Delta t = 17$, for which the system has an attractor with a fractal dimension of about 2.1 [14]. Following other published results we have chosen $\tau = 6$ and $d_E = 4$ for the reconstructed phase space [2, 5, 6]. To set up the WCP and the WLP we have to decide how many cluster centers to use for a given amount of training data. We have no criterion for deciding what the "optimal" number should be. However, in all of the examples below we have chosen it in such a way that the number of free parameters is one-fourth the number of training points used. (This choice is arbitrary and definitely not optimal with regard to prediction accuracies.) Thus, for the WCP there are 4 training vectors per cluster center, and for the WLP there are 20 per cluster center.

Figure 10 shows the prediction error $\bar{E}$ (see equation (3.1)) for predicting the time series $T = \tau$ into the future as a function of the number of training points N. The WLP performs better than the WCP for all values of N investigated.

How much slower is the WLP compared to the WCP? Both algorithms require time to set up and train, and this time increases with the amount of training data used. In figure 11 we plot $\bar{E}$ versus the CPU time needed to set up and train the two predictors. The training was done by going through the training set 200 times. This was enough to get within 0.001 of the minimum of E_f (defined in equation (2.5)). The training could have been stopped several steps earlier without affecting the prediction errors noticeably, so the CPU times in the figure represent upper bounds. Figure 11 shows that when

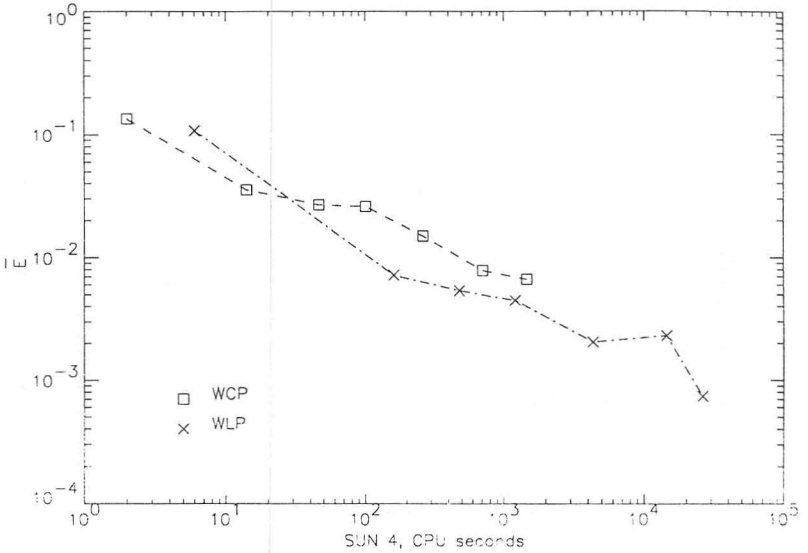

Figure 11: The prediction error as a function of the CPU time needed to set up and train the WCP and the WLP. The calculations are for the same cases as those of figure 10.

high accuracy predictions are desired using the smallest amount of computing time we should use the WLP rather than the WCP. With this in mind, we now abandon the WCP in favor of the WLP for the rest of this section.

We now compare the WLP with the LLP of references [1, 2]. In figure 12 we have used the Mackey-Glass time series with $\Delta t = 17$, using the same phase space reconstruction as above. We constructed maps that predict $T = 6$ into the future using 100 and 500 training-points, respectively. The figure shows the results of iterating the resulting maps. For the 100 data-point case, the LLP is not capable of producing a valid map; the prediction error blows up almost from the start. The WLP does not have this problem; the prediction error grows smoothly as the map is iterated. When 500 data points are used instead, the LLP produces a valid map. Now the error is comparable to the one obtained with the WLP.

To study a higher-dimensional case, we now consider the Mackey-Glass equation with $\Delta t = 30$. At this parameter value, the system has an attractor with a fractal dimension of 3.6 [14]. We reconstructed a phase space using $\tau = 6$ and $d_E = 6$. Because the dimensionality of the attractor is larger than that of the $\Delta t = 17$ case, we use 1000 and 5000 training points to set up the predictors, which predict $T = 6$ into the future. The results of iterating the resulting maps are shown in figure 13. As can be seen from the figure, even 5000 training points are not enough to enable the LLP to make predictions. The prediction error increases dramatically if one tries to forecast beyond a certain time. For both the 1000 and 5000 training points

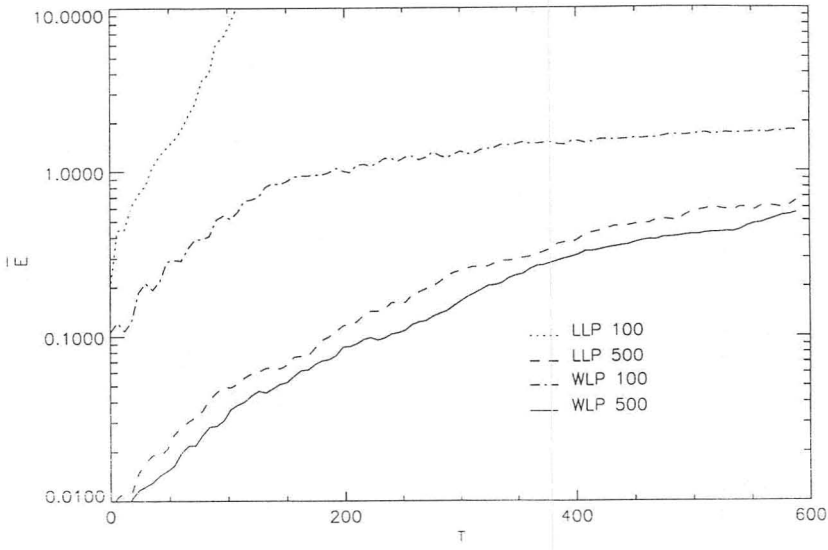

Figure 12: The prediction error as a function of the prediction time T for the WLP and the LLP using time series from equation (3.3) with $\Delta t = 17$. The predictions were made by iterating maps built with 100 and 500 training points that predict $T = 6$ into the future.

cases, the WLP produces a map with a prediction error that is well-behaved when the map is iterated. Furthermore, the error in the first iteration is one order of magnitude less than the one obtained with the LLP.

Recall that local linear predictions can be done "on-the-fly." This means that to make a single prediction, the local map must be *built and evaluated.* Thus, making a single prediction with the LLP requires slightly more time than a global method that uses a function that has already been constructed. However, all the time needed to set up such a global predictor is avoided by the LLP. If one is concerned with computational speed, the choice of method should depend on how many predictions will be made. If only a few predictions are desired, a global approach will be slower than a local one; while a global approach might be faster when a lot of predictions are to be made. For example, on a Sun-4 computer for the 500-data-point case of figure 12, which involves a total of 100,000 predictions, the WLP took 160 seconds to set up, while 766 seconds were required to make the predictions. Thus, it took our scheme a total of 926 Sun-4 CPU seconds for the entire calculation, while the LLP took 1811 Sun-4 CPU seconds.

4. Conclusions

In this paper we have proposed a fast and accurate method for predicting chaotic time series. The method incorporates a representation based on basis

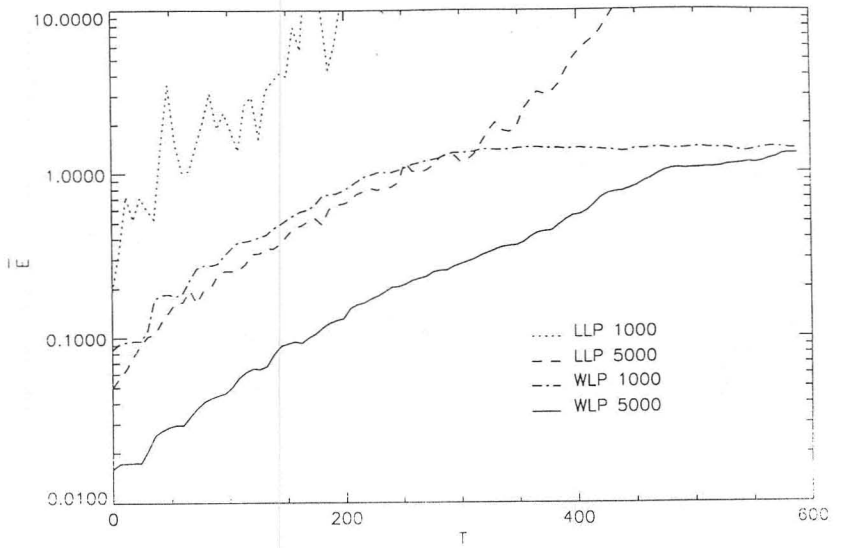

Figure 13: Similar to figure 12, except that time series from equation (3.3) with $\Delta t = 30$ was used. The number of training points was increased to 1000 and 5000, respectively.

functions localized in the input space. The representation can be viewed as a neural network or simply as a weighted superposition of linear maps. The efficiency of the algorithm has been obtained by exploiting the localization properties of the basis functions through the use of certain data structures. The resulting predictor is superior to that of Moody and Darken, which can be seen as a weighted superposition of constant maps. When the amount of data is limited, the short time predictions resulting from our scheme are superior to those obtained by the local linear predictor of references [1, 2].

Our technique might be improved upon by incorporating even higher-order terms. We have investigated the use of quadratic terms in the scheme. In a number of cases, the prediction accuracies thus obtained have been better than those obtained with the linear terms. However, since the number of free parameters increases substantially when quadratic terms are added, the training time required to construct the corresponding network increases dramatically. Thus, we do not believe that using higher-order polynomials is the way to go. Perhaps some other representation of the local mappings, such as rational functions, is more appropriate. In fact we believe that any representation that will work in the local scheme of references [1, 2] will also work in our scheme.

Acknowledgements

We would like to thank M. C. Casdagli, J. D. Farmer, A. F. Illarionov, A. Krogh, J. Moody, H. H. Rugh, S. Solla, E. Sedykh, and J. J. Sidorowich for valuable discussions.

References

[1] J. D. Farmer and J. J. Sidorowich, "Predicting chaotic time series," *Phys. Rev. Lett.*, **59** (1987) 845.

[2] J. D. Farmer and J. J. Sidorowich, *Exploiting Chaos to Predict the Future and Reduce Noise*, Technical Report LA-UR-88-901, Los Alamos National Laboratory, Los Alamos, NM (1988).

[3] J. P. Crutchfield and B. C. McNamara, "Equations of motion from a data series," *Complex Systems*, **1** (1987) 417.

[4] M. C. Casdagli, "Nonlinear prediction of chaotic time series," *Physica D*, **35** (1989) 335.

[5] A. Lapedes and R. Farber, *Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling*, Technical Report LA-UR 87-2662, Los Alamos National Laboratory, Los Alamos, NM (1987).

[6] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, **1** (1989) 281; J. Moody and C. Darken, "Learning with localized receptive fields," in *Proceedings of the 1988 Connectionist Models Summer School* (Pittsburg, 1988).

[7] For a nice introduction to chaos, strange attractors, and fractal dimensions see E. Ott, "Strange attractors and chaotic motions of dynamical systems," *Rev. Mod. Phys.*, **53** (1981) 655.

[8] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw, "Geometry from a time series," *Phys. Rev. Lett.*, **45** (1980) 712.

[9] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence* (Warwick, 1980) and *Lecture Notes in Mathematics 898* (Springer, Berlin, 1981) p. 366.

[10] See reference [2] for short discussion of the history of local approximation schemes.

[11] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, **3** (1990) 109.

[12] J. H. Bentley, "Multidimensional binary search trees in database applications," *IEEE Transactions on Software Engineering*, **SE-5(4)** (1979) 333; J. H. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, **18** (1975) 509.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes* (Cambridge University Press, 1988).

[14] J. D. Farmer, "Chaotic attractors of an infinite dimensional system," *Physica D*, **4** (1982) 366.