Local Averaging Optimization for Chaotic Time Series Prediction ⋆

James McNames

Electrical & Computer Engineering, Portland State University, Post Office Box 751, Portland, OR 97207-0751. Email: mcnames@ee.pdx.edu. Web: http://ece.pdx.edu/∼mcnames.

Abstract

Local models have emerged as one of the most accurate methods of time series prediction, but their performance is sensitive to the choice of user-specified parameters such as the size of the neighborhood, the embedding dimension, and the distance metric. This paper describes a new method of optimizing these parameters to minimize the multi-step cross-validation error. Empirical results indicate that multi-step optimization is susceptible to shallow local minima unless the optimization is limited to ten or fewer steps ahead. The models optimized using the new method consistently performed better than those optimized with adaptive analog forecasts.

Key words: Local models, time series prediction, metric optimization, chaos, embedding dimension

1 Introduction

Local models generate predictions by finding local portions of the time series that closely resemble a portion of the points immediately preceding the point to be predicted. The prediction is estimated as an average of the changes that occurred immediately after these similar portions of points.

One of the most vexing problems facing users who wish to construct a local model is how to choose appropriate values for the model parameters. Since the best parameter values depend on the properties of the data set in a manner that is generally unknown, there is little to guide users in making this decision. This paper describes a new method for optimizing these parameters to

⋆ Extended version of paper presented at the ESANN 2000 conference.

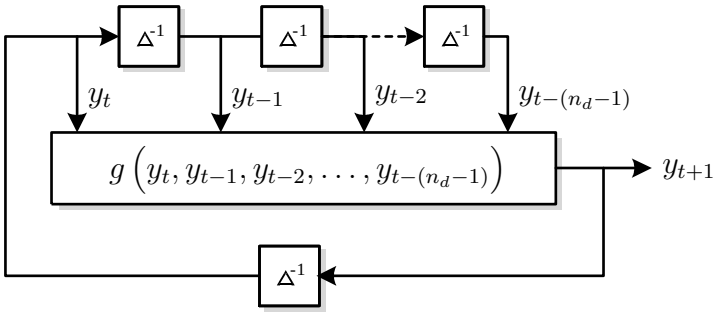


Fig. 1. System equivalent of a deterministic dynamic system. Takens' theorem implies that, under general conditions, a function $g(\cdot)$ exists such that perfect prediction is possible given a sufficient segment of points preceding the prediction. A unit delay is denoted by $\triangle^{-1}$.

minimize the multi-step prediction error. This relieves the user of having to specify critical parameter values, it gives the user control of the computation used to build and optimize the model, and it generally improves the model accuracy.

The next section discusses the importance of Takens theorem and the effect of embedding parameters on prediction accuracy. Section 3 describes local averaging models. Section 4 discusses iterated prediction models and how to assess the accuracy of multi-step predictions. Section 5 describes a new method of local averaging optimization and Section 6 summarizes the results from some empirical studies of model performance and sensitivity to local minima.

2 Takens' Theorem

Given a time series of points, $[y_1, y_2, \ldots, y_{n_y}]$, sampled from a dynamic system, the goal of the model is to predict the next n_a points in the series, $[y_{n_y+1}, \ldots, y_{n_y+n_a}]$. Nonlinear autoregressive models are the most popular approach to this problem because Takens has shown that, under very general conditions, the state of a deterministic dynamic system can be accurately reconstructed by a finite window of the time series [32]. This window,

$$x_t \triangleq [y_t, y_{t-\delta}, \ldots, y_{t-(n_d-1)}], \tag{1}$$

is called a time delay embedding, where y_t is the value of the time series at time t and n_d is the embedding dimension. Takens' work was later generalized and shown to apply to a broader class of systems [29]. This is an important theorem because it implies that if general assumptions are satisfied, there exists a function $g(x_t)$ such that $y_{t+1} = g(x_t)$. This idea is illustrated by Fig. 1.

Since perfect predictions would be possible if the function $g(x_t)$ were known and since a time series provides many examples of the mapping $y_{t+1} = g(x_t)$, the time series prediction problem can be naturally framed as a autoregressive modeling problem where the goal is to construct an estimate $\hat{y}_{t+1} = \hat{g}(x_t)$.

Takens' theorem requires the time series to be noise-free and the function $g(x_t)$ to be known. In practice, these conditions are rarely satisfied and the selection of the embedding dimension, n_d, and other model parameters may have a strong impact on the prediction accuracy. Many researchers have recognized this problem and proposed methods to find n_d for time series with and without noise [8–10, 21]. The goal of these methods is usually to minimize n_d without sacrificing the accuracy of the reconstructed state x_t. Although a compact reconstruction is efficient computationally, it does not necessarily maximize the model prediction accuracy. Section 5 describes a new approach to finding n_d based on optimization of the local modeling metric.

3 Local Averaging Models

Local averaging models estimate $g(x_t)$ with a weighted average of outputs in the training set with similar input vectors,

$$\hat{g}(x_t) = \frac{\sum_{i=1}^{n_p} w_i^2 y_{i+1}}{\sum_{i=1}^{n_p} w_i^2}, \tag{2}$$

where n_p is the number of data set points that can be constructed from the time series, y_{i+1} is the ith output in the training set, and w_i^2 is a weighting function that controls how much influence the point y_{i+1} has on the model output.

Typically, w_i^2 is a function of the distance between the current input vector and the ith input vector in the data set. The accuracy of local models is not strongly affected by the shape of the weighting function so long as it is a non-negative, monotonically decreasing, smooth function of the distance [1]. A good choice is the biweight function shown in Fig. 2 and defined as

$$w_i^2 \triangleq \mathrm{B}(d_i^2, d_{k+1}^2) = \begin{cases} \left(1 - \left(\frac{d_i^2}{d_{k+1}^2}\right)\right)^2 & d_i^2 \le d_{k+1}^2, \\ 0 & \text{otherwise}, \end{cases} \tag{3}$$

where d_i is the distance to the ith nearest neighbor and d_{k+1} is the distance to the $(k+1)$th nearest neighbor. Since this function has a continuous first derivative that decays to zero for distances greater than or equal to d_{k+1}, the model $\hat{g}(x_t)$ is also guaranteed to have a continuous first partial derivative with respect to the model inputs. It has the further advantage of reducing

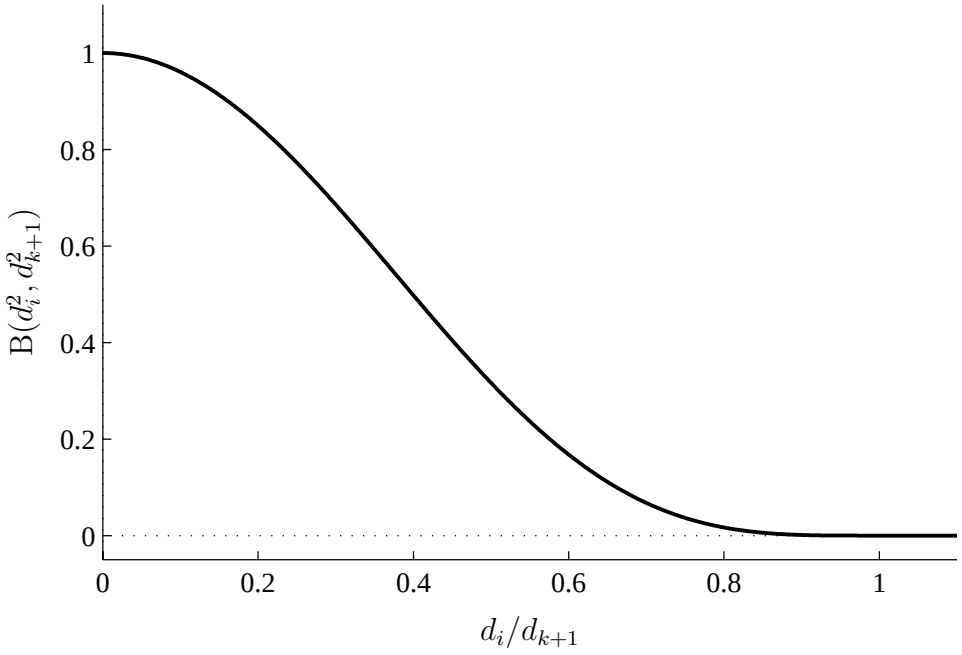


Fig. 2. The biweight function.

computation since only k terms in the sums of Equation 2 are non-zero. Goutte and Larsen give a concise summary of other popular weighting functions and their derivatives [18].

Most of the computation required for local models is used to find the $k+1$ nearest neighbors. Fortunately, this is not prohibitive as long as an efficient nearest neighbor algorithm is used [16, 20].

3.1 Distance Metrics

Choosing an appropriate measure of distance for local models is an important decision. The Euclidean distance,

$$\mathrm{D}_{\mathrm{E}}^2\left(x_t, x_i\right) = (x_t - x_i)^{\mathrm{T}}(x_t - x_i) = \sum_{j=1}^{n_d}(x_{t,j} - x_{i,j})^2, \tag{4}$$

where x_t is the model input vector and x_i is the input vector of the ith nearest neighbor, is the most common choice, but there is no reason to believe this is the best choice in general. This metric is chosen primarily because of its simplicity and intuitive geometric appeal. Choosing a more general distance measure with more parameters can substantially improve the model accuracy. The weighted Euclidean distance,

$$\mathrm{D}_{\mathrm{E}}^2\left(x_t, x_i\right) = (x_t - x_i)^{\mathrm{T}}\Lambda(x_t - x_i), \tag{5}$$

where Λ is a positive-semidefinite matrix, is much more flexible. The local averaging model described here uses a diagonal weighting matrix,

$$\mathrm{D}^2_{\mathrm{WE}}(x_t, x_i) = \sum_{j=1}^{n_d} \lambda_j^2 (x_{t,j} - x_{i,j})^2, \tag{6}$$

where λ_j^2 are the elements on the diagonal of Λ. This is the same metric that is optimized for time series prediction by an algorithm developed by Fraedrich and Rückert called Adaptive Analog Forecasts (AAF) [15]. Section 6 compares their method of prediction with the one described here.

Several variations on the weighted Euclidean metric have been considered for time series prediction. Tanaka *et al.* developed an elegant approximation to an optimal metric based on a rank-1 matrix Λ [33]. This metric is based on a first order approximation of the system dynamics, has no user-specified parameters, and is computationally inexpensive. The same metric was developed independently by Garcia *et al.* [17].

Farmer and Sidorowich recommended a diagonal matrix for time series prediction with $\lambda_i = e^{-hi}$, where h is the metric entropy, and claimed that this metric is in some sense linearly optimal [13]. Murray examined the performance of this type of metric and a tridiagonal metric on several chaotic time series for various rates of decay, h, and found that they consistently outperformed the Euclidean metric [28]. The author generated the winning entry in the Lueven time series prediction competition by jointly optimizing the rate of exponential decay, h, and the number of neighbors, k, to minimize the multi-step cross-validation error [25, 27].

Kugiumtzis reported that there was not a significant difference between three types of norms in a brief analysis of linear models for time series prediction [22]. Casdagli divided the training set into a training segment and a test segment and selected the embedding dimension of the Euclidean metric based on the model performance on the test segment [10]. Casdagli and Weigend reported that model accuracy is very sensitive to the choice of the embedding dimension when the Euclidean metric is used [12].

Investigation of alternatives to the Euclidean metric have been more thorough for the applications of classification and nonlinear modeling than for time series prediction. For example, Lowe investigated a method of using gradient-based optimization of the leave-one-out cross-validation error to optimize the diagonally weighted Euclidean metric of a nearest neighbor classifier [23]. Hastie and Tibshirani describe a method of k-nearest neighbor classification that optimizes the diagonal distance metric individually for each input [19]. Schaal and Atkeson described a local model akin to radial basis functions in which Λ is a full rank matrix that is optimized to minimize a stochastic approximation of the leave-one-out cross-validation error for nonlinear modeling [30].

This method is not suitable for models with a large number of inputs, n_d, because the number of free parameters is proportional to n_d^2. Goutte and Larsen described a method of optimizing the diagonal distance metric for kernel smoothers to minimize the v-fold cross-validation error [18].

3.2 Effective Embedding Dimension

One advantage of optimizing the diagonal weighted Euclidean metric is that it effectively eliminates the problem of choosing the embedding dimension, n_d [28]. Since n_d only affects the model output $\hat{g}(x_t)$ through the distance equation (Equation 6), if $\lambda_{n_d} = 0$, the effective embedding dimension is reduced to $n_d - 1$ since the last element of the input vector would have no effect on the model output. Thus, if the embedding dimension is chosen to be large enough, the optimization of λ will result in an effective embedding dimension that maximizes the model accuracy. This is an attractive alternative to employing one of the many methods developed to find the minimum embedding dimension because it directly achieves the end goal of maximizing the model accuracy, rather than minimizing the size of the reconstructed state vector x_t.

4 Multi-Step Predictions

This section discusses two different approaches to generating predictions for more than one step ahead, discusses the importance of picking an appropriate error measure for models that use iterated prediction, and describes a method of recursively calculating the gradient of the multi-step cross-validation error with respect to the metric parameters.

4.1 Iterated versus Direct Prediction

Suppose we are given a time series, $[y_1, y_2, \ldots, y_{n_y}]$, and asked to predict n_a steps ahead. *Direct prediction* is a method in which a model is built to directly predict y_{t+n_a}. *Iterated prediction* is a method in which a model is built to predict one step ahead. To predict more than one step ahead, earlier predictions are used as model inputs. In each step, the model estimates y_{t+i+1} from $\hat{x}_{t+i}$

and this is then used to estimate x_{t+i+1}. For example,

$$\begin{aligned} x_t &= \left[y_t, y_{t-1}, y_{t-2}, \ldots, y_{t-(n_d-1)}\right], \\ \hat{y}_{t+1} &= \hat{g}(x_t), \\ \hat{x}_{t+1} &= \left[\hat{y}_{t+1}, y_t, y_{t-1}, \ldots, y_{t-(n_d-2)}\right], \\ \hat{y}_{t+2} &= \hat{g}(\hat{x}_{t+1}), \\ \hat{x}_{t+2} &= \left[\hat{y}_{t+2}, \hat{y}_{t+1}, y_t, \ldots, y_{t-(n_d-3)}\right], \\ \hat{y}_{t+3} &= \hat{g}(\hat{x}_{t+2}), \\ &\vdots \end{aligned} \tag{7}$$

This process is iterated for n_a steps finally producing the prediction $\hat{y}_{t+n_a}$.

There has been much debate over which method is better. Direct prediction is questionable because a function that maps n_a steps into the future will usually be more complicated and thereby harder to model than one that predicts a single step into the future [14]. Iterated prediction is questionable because it does not take into account the accumulated errors in the input vector, $\hat{g}(x_{t+c}) \neq \hat{g}(\hat{x}_{t+c})$ [26]. Most researchers have found that iterated prediction is more accurate than direct prediction [10, 11, 13, 14]. The following sections describe a method of optimizing local models that use iterated prediction.

4.2 One-Step Cross-Validation Error

Having an accurate and efficient means of measuring the model accuracy is necessary for optimization to be effective. Local models have a distinct advantage over other methods in this regard because they can efficiently calculate the leave-one-out cross-validation error, C. This measure of model accuracy is calculated by taking a single point out of the data set, building a model using the remaining $n_p - 1$ points, and then using the model to estimate the output at the removed point. The process is repeated for n_c selected points in the data set and the average error is calculated. Mathematically, the average C is defined as

$$\mathrm{C} \triangleq \frac{1}{n_c} \sum_{i=1}^{n_c} p\left(y_{c(i)+1} - \hat{g}^{-}(x_{c(i)})\right), \tag{8}$$

where n_c controls the accuracy of the estimated error, $c(i)$ is the data set index of the ith cross-validation point, and $p(\cdot)$ is a user-specified penalty function [1].

[1] The squared error penalty function, $p(\varepsilon) = \varepsilon^2$, is the most common choice, but any smooth function could be used.

The local model output, $\hat{g}^-(x_{c(i)})$, is from a model constructed with the $c(i)$th point omitted from the data set and then used as the model input.

Local models can calculate the cross-validation error almost as efficiently as they can calculate the local model outputs. The number of nearest neighbors is increased by one and after the $k+2$ nearest neighbors have been found, the nearest neighbor, which is identical to the model input vector, is discarded and the model is constructed using the remaining $k+1$ neighbors.

4.3 Multi-Step Cross-Validation Error

In framing the time-series prediction problem as a nonlinear modeling problem, the natural error measure for local models is the cross-validation error (Equation 8). Since the error is estimated using points that are not included in each of the n_c training sets, this is an elegant measure of prediction error that penalizes overfitting and ensures good generalization. An extensive discussion of the merits and pitfalls of cross-validation and other measures of prediction error for linear model selection can be found in [6, 7].

Since C is an estimate of the error of predicting one-step ahead, it is often called the one-step cross-validation error (OSCVE). A crucial disadvantage of the OSCVE is that it does not take into account the effect of the errors in the input vector that occur with iterated prediction; the parameter values that minimize the OSCVE are generally not the same values that maximize the model performance for predicting n_a steps ahead [5, 27].

A more appropriate choice is an error measure that reflects the true cost for making iterated predictions where the *true cost* depends on the application. In many cases an average (possibly weighted) model accuracy over n_s-steps ahead is appropriate,

$$\overline{\mathrm{C}} \triangleq \frac{1}{n_c n_s} \sum_{i=1}^{n_c} \sum_{j=1}^{n_s} p\left(y_{c(i)+j+1} - \hat{g}^-(x_{c(i)+j})\right), \tag{9}$$

where n_s is the number of steps ahead over which the average error is measured. This error measure is called multi-step cross validation error. Minimization of this error is similar to backpropagation through time and real time recurrent learning, but here the leave-one-out cross-validation error is minimized rather than the average training set error [34, 35].

$\overline{\mathrm{C}}$ has the disadvantage of requiring significantly more computation than the OSCVE. The user can tradeoff reduced accuracy of the estimated $\overline{\mathrm{C}}$ for reduced computation by specifying n_c, the number of cross-validation segments, and n_s, the number of steps ahead the $\overline{\mathrm{C}}$ is estimated.

4.4 Multi-Step Error Gradient

Many of the most efficient optimization algorithms require the gradient of an error function with respect to the model parameters. This section describes a derivation of the $\overline{\mathrm{C}}$ gradient with respect to the metric weights λ. In this section the model output is denoted as $\hat{g}_\lambda(\cdot)$ and the multi-step cross-validation error is denoted by $\overline{\mathrm{C}}_\lambda$ to explicitly show the dependence on the metric weights. Section 5 describes a method that uses the multi-step error gradient derived in this section to jointly optimize λ and k, the number of neighbors.

Both the model output $\hat{g}_\lambda(\cdot)$ and, if iterated prediction is used, the model input vector [2],

$$\hat{x}_t = \left[\hat{y}_t, \hat{y}_{t-1}, \ldots, y_{t-(n_d-1)}\right]^{\mathrm{T}} = \left[\hat{g}_\lambda(\hat{x}_{t-1}), \hat{g}_\lambda(\hat{x}_{t-2}), \ldots, y_{t-(n_d-1)}\right]^{\mathrm{T}},$$

depend on the metric weights λ. The gradient of $\overline{\mathrm{C}}$ is then given by

$$\nabla_\lambda \overline{\mathrm{C}}_\lambda = \frac{1}{n_c n_s} \sum_{i=1}^{n_c} \sum_{j=1}^{n_s} \nabla_\lambda p\left(y_{c(i)+j+1} - \hat{g}_\lambda^-(\hat{x}_{c(i)+j})\right). \tag{10}$$

If we define a new variable, $\varepsilon \triangleq y_{c(i)+j+1} - \hat{g}_\lambda^-(x_{c(i)+j})$, the gradient can be written compactly as

$$\nabla_\lambda \overline{\mathrm{C}}_\lambda = -\frac{1}{n_c n_s} \sum_{i=1}^{n_c} \sum_{j=1}^{n_s} \frac{dp(\varepsilon)}{d\varepsilon} \nabla_\lambda \hat{g}_\lambda^-(\hat{x}_{c(i)+j}). \tag{11}$$

The lth element of the gradient of the model output, $\nabla_\lambda \hat{g}_\lambda^-(\hat{x}_{c(i)+j})$, is

$$\frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+j})}{\partial \lambda_l} = \underbrace{\left.\frac{\partial \hat{g}_\lambda^-(u)}{\partial \lambda_l}\right|_{u=\hat{x}_{c(i)+j}}}_{①} + \sum_{m=1}^{\min(n_d, j-1)} \underbrace{\frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+j})}{\partial \hat{x}_{c(i)+j,m}}}_{②} \underbrace{\frac{\partial \hat{x}_{c(i)+j,m}}{\partial \lambda_l}}_{③}. \tag{12}$$

The first term, ①, accounts for the effect of λ_l on the model estimate $\hat{g}_\lambda(u)$ for a constant input, u. The second term, ②, is the partial derivative of the model output with respect to the mth element of the model input vector. Both of these derivatives are derived in Appendix A.

Since the estimated input vector is created from previous model outputs, $\hat{x}_{c(i)+j,m} = \hat{g}_\lambda^-(\hat{x}_{c(i)+j-m})$, the last term, ③, is the lth element of the gradient of the model output at earlier times. Thus, the gradient can be calculated recursively,

[2] Recall from Equation 7 that each element of x_t is estimated only when the true value is not known.

$$
\begin{aligned}
\frac{\partial \hat{g}_\lambda^-(x_{c(i)})}{\partial \lambda_l} &= \left.\frac{\partial \hat{g}_\lambda^-(u)}{\partial \lambda_l}\right|_{u=x_{c(i)}}, \\
\frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+1})}{\partial \lambda_l} &= \left.\frac{\partial \hat{g}_\lambda^-(u)}{\partial \lambda_l}\right|_{u=\hat{x}_{c(i)+1}} + \frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+1})}{\partial \hat{x}_{c(i)+1,1}} \frac{\partial \hat{g}_\lambda^-(x_{c(i)})}{\partial \lambda_l}, \\
\frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+2})}{\partial \lambda_l} &= \left.\frac{\partial \hat{g}_\lambda^-(u)}{\partial \lambda_l}\right|_{u=\hat{x}_{c(i)+2}} + \frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+2})}{\partial \hat{x}_{c(i)+2,1}} \frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+1})}{\partial \lambda_l} \\
&\quad + \frac{\partial \hat{g}_\lambda^-(\hat{x}_{c(i)+2})}{\partial \hat{x}_{c(i)+2,2}} \frac{\partial \hat{g}_\lambda^-(x_{c(i)})}{\partial \lambda_l}. \\
&\vdots
\end{aligned}
$$

This recursive approach enables the multi-step-ahead cross-validation error gradient in Equation 11 to be calculated efficiently.

5 Local Averaging Optimization (LAO)

This section describes a method of optimizing the local averaging model parameters. The model is described by Equations 2, 3, and 6. The model parameters are the metric weights λ, the number of neighbors k, and the maximum embedding dimension n_d.

Smith described a method of optimizing k for each input using local linear least-squares models to minimize the one-step cross-validation error [31]. Birattari *et al.* generalized this method to use weighted least squares and reduced the computation by using recursive least squares [3]. Bontempi *et al.* described a more general method of adapting k to each input so as to minimize the multi-step cross-validation error of local linear models [4, 5]. For the Lueven time series prediction competition, I jointly optimized k globally and the exponentially weighted metric to minimize the multi-step cross-validation error [25, 27]. To my knowledge, this is the first description of an algorithm to jointly optimize the multi-step cross-validation error of the metric weights, λ, and the number of neighbors, k.

Gradient-based optimization algorithms can greatly improve the initial parameter values provided by the user, but this approach cannot be used to optimize integer-valued parameters, such as the number of neighbors, or other parameters for which the gradient cannot be calculated. To optimize these parameters, an algorithm that does not require the gradient must be used. One of the simplest of these algorithms is the cyclic coordinate method (CCM) which performs a global optimization of each parameter one at a time, and then repeats until convergence [2, pp. 283–5].

CCM's rate of convergence is much slower than gradient-based optimization methods. If the gradient can be calculated for only some of the model parameters, the rate of convergence can be increased by combining CCM with a gradient-based algorithm. Here a generalization of CCM is described that combines cyclic optimization with gradient-based optimization. During each cycle, k is found by an exhaustive search and the metric parameters are jointly optimized by a gradient-based optimization algorithm for a user-specified number of steps. This approach is substantially faster than using CCM to optimize each of the parameters one at a time. This generalized algorithm is described in detail below.

Generalized Cyclic Coordinate Method

1. Initialize the stopping criterion:
   $n_i := 0$.
2. Store the current value of $\overline{\mathrm{C}}$:
   $\overline{\mathrm{C}}_{\mathrm{prev}} := \overline{\mathrm{C}}(\lambda, k)$.
3. Perform an exhaustive search for k:
   $k := \underset{1 \le \alpha \le k_{\max}}{\operatorname{argmin}} \overline{\mathrm{C}}(\lambda, \alpha)$.
4. Perform gradient-based optimization for λ:
   For $i = 1$ to n_u,
   4.1 Calculate $\nabla_\lambda \overline{\mathrm{C}}(\lambda, k)$.
   4.2 Calculate a new direction of descent, g.
   4.3 Perform a line search:
   $\alpha := \underset{\alpha \ge 0}{\operatorname{argmin}} \overline{\mathrm{C}}(\lambda + \alpha g, k)$.
   4.4 Update the metric weights:
   $\lambda := \lambda + \alpha g$.
   4.5 Next i.
5. Update the count of iterations:
   $n_i := n_i + 1$.
6. If $n_i = n_{i,\max}$ or $\overline{\mathrm{C}}_{\mathrm{prev}}(\lambda, k) - \overline{\mathrm{C}}(\lambda, k) < \epsilon$, then the algorithm has reached the allowed number of iterations or converged. Exit function.
7. Goto 2.

Since each step in the loop can only decrease the cross-validation error, this method can only improve the model performance. Under very general conditions the algorithm is guaranteed to converge [2, p. 285].

The user-specified parameters of this optimization algorithm are the range of values of k considered, $k_{\max}$, the number of gradient-based updates per iteration, n_u, the maximum number of iterations, n_i, and the minimum reduction in $\overline{\mathrm{C}}$ to continue the optimization, ϵ. Each of these parameters enables the user to control the tradeoff between the amount of computation used for parameter optimization and the final accuracy of the model.

Table 1
List of the user-specified parameters for LAO with the Lorenz time series.

Parameter	Label	Value
n_y	Length of time series	3000
$n_{i,\max}$	Maximum number of iterations	100
n_d	Maximum embedding dimension	20
n_s	Number of steps ahead	1
n_u	Gradient updates per iteration	5
ϵ	Reduction of $\overline{\mathrm{C}}$ for convergence	0
$k_{\max}$	Maximum number of neighbors	15
$p(x)$	Penalty Function	x^2

All of the results reported in this paper used PARTAN, a conjugate gradient optimization algorithm, to calculate the directions of descent in Step 4.2 and the golden section method for the line search in Step 4.3 [2, 24].

6 Empirical Results

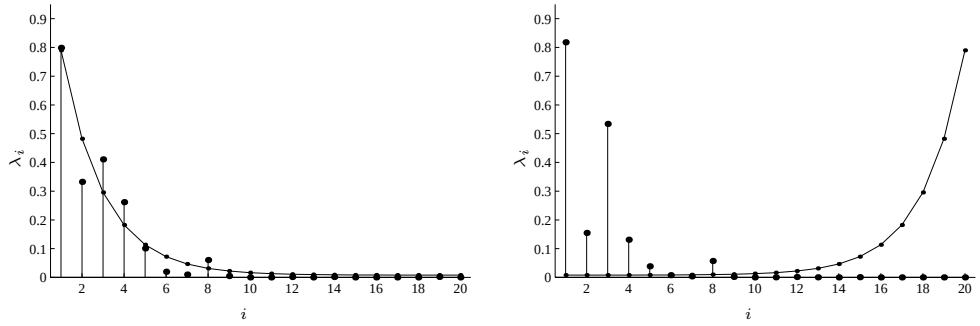
Although it is not possible to illustrate how much the optimization improves model performance compared to user-specified parameter values, the results in this section illustrate some of the benefits and limitations of local averaging optimization (LAO).

6.1 Metric Weights

To determine the sensitivity of LAO to the initial choice of the metric parameters, the algorithm was applied to the Lorenz [3] time series with four different initial metric weight vectors. The user-specified parameter values are listed in Table 1. Fig. 3 shows both the initial and optimized metric weights for each case.

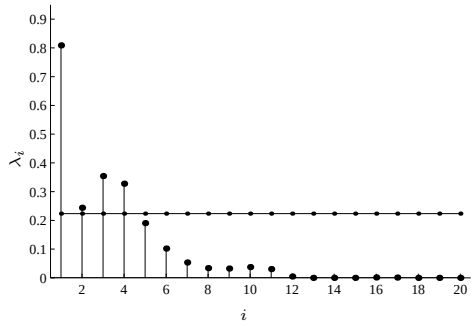
The optimized weights were nearly indistinguishable from one another, regardless of the initial weights. Similar results were obtained on many other time series. This indicates that for one-step ahead optimization, the algorithm is

[3] All of the data sets used in this work are available online at http://ece.pdx.edu/∼mcnames.

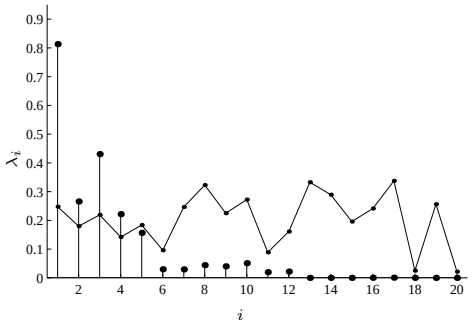


(a) Exponential Decay

(b) Exponential Growth



(c) Euclidean

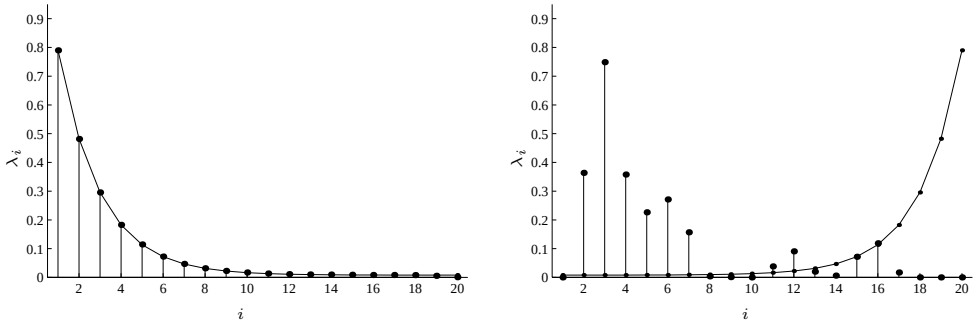


(d) Random

Fig. 3. The initial metric weights before optimization are shown by the line and the final metric weights after optimization are shown by the stem plot. The models were optimized to predict one step ahead ($n_s = 1$) on the Lorenz time series.

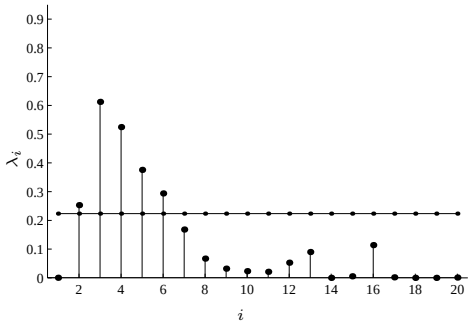
insensitive to local minima and tolerant of the user picking poor initial parameter values. These results also demonstrate that the best metric weights are similar to a decaying exponential, which has been assumed, but not demonstrated in previous work [13, 17, 27].

The models were optimized a second time to predict ten steps ahead ($n_s = 10$) with the same initial weight vectors. Fig. 4 shows the initial and optimized weights for each case. LAO was unable to improve the weights that were initialized as a decaying exponential in this case. This type of local minimum becomes more prevalent as the number of steps ahead, n_s, increases. In the other three cases, the optimized weights were insensitive to the initial values.

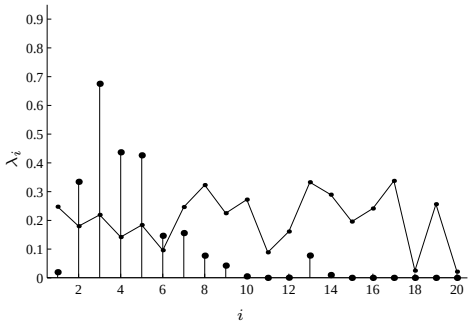


(a) Exponential Decay

(b) Exponential Growth



(c) Euclidean



(d) Random

Fig. 4. The initial metric weights before optimization are shown by the line and the final metric weights after optimization are shown by the stem plot. The models were optimized to predict ten steps ahead ($n_s = 10$) on the Lorenz time series.

6.2 Iterated Prediction Performance

Local averaging models were optimized to predict 1, 5, 10, 20, and 40 steps ahead on four different time series. Their performance was then measured on later sections of the time series not used for optimization. The user-specified parameters are listed in Table 1.

The performance was measured using the square root of the normalized mean squared error, $\mathrm{R}_j = \sqrt{\mathrm{N}_j}$, where N_j is defined as

$$\mathrm{N}_j \triangleq \frac{\frac{1}{n_t}\sum_{i=1}^{n_t}\left(y_{v(i)+j+1} - \hat{g}(\hat{x}_{v(i)+j})\right)^2}{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}.$$

In this equation y_t is the tth point in the time-series, $v(i)$ is the index of the first point in the ith test set, $\bar{y}$ is the sample mean of the time series, and $\hat{g}(\hat{x}_{v(i)+j})$ is the prediction of a model that has been iterated j times.

Table 2
Root normalized mean squared error for local averaging models optimized to predict 1, 5, 10, 20, and 40 steps ahead.

n_s	1	5	10	20	40
Euclidean	0.054	0.086	0.152	0.266	0.474
1 Step	0.026	0.049	0.097	0.293	0.570
5 Steps	0.029	0.040	0.075	0.232	0.449
10 Steps	0.033	0.042	0.069	0.193	0.441
20 Steps	0.057	0.069	0.118	0.234	0.444
40 Steps	0.055	0.067	0.113	0.230	0.440

R_j is a convenient measure of error because it is independent of the scale of the time series and $\mathrm{R}_j = 1$ can be interpreted as meaning the model prediction error is no better than predicting the sample mean $\bar{y}$, on average.

Tables 2 and 3 show the average R, $\overline{\mathrm{R}}_\ell = \frac{1}{\ell}\sum_{j=1}^{\ell} \mathrm{R}_j$, for $\ell = 1$, 5, 10, 20, and 40 steps ahead for the Lorenz and Mackey-Glass time series. The Euclidean weights were used to initialize the optimization algorithm in all cases.

For both data sets, the model optimized to predict one step ahead ($n_s = 1$) performed best on the test set for predicting one step ahead. However, the models optimized to predict multiple steps ahead ($n_s > 1$) did not always perform best at predicting n_s steps ahead on the test set. This is due to the presence of local minima in the multi-step cross-validation error surface.

These results indicate that although it is possible to optimize the models to predict multiple steps ahead, these models are sometimes not as accurate as models optimized to predict fewer steps ahead ($n_s < n_a$). However, models optimized to predict five or ten steps ahead often performed better on long term predictions than the models optimized to predict one step ahead. Similar results were obtained on other time series. This demonstrates that multi-step optimization produces more accurate long term predictions.

6.3 Direct Prediction Performance

Fraedrich and Rückert were the first to describe a method of optimizing a diagonal metric for local models [15]. This method, called adaptive analog forecasts (AAF), is more limited than local averaging optimization (LAO). AAF only works with $k = 1$ neighbor models, which are discontinuous and less general than local averaging models. Unlike LAO, AAF is incapable of jointly optimizing many model parameters, discrete and continuous-valued.

Table 3
Root normalized mean squared error for local averaging models optimized to predict 1, 5, 10, 20, and 40 steps ahead on the Mackey-Glass time series.

n_s	1	5	10	20	40
Euclidean	0.021	0.026	0.029	0.038	0.060
1 Step	0.012	0.020	0.023	0.032	0.052
5 Steps	0.018	0.018	0.019	0.027	0.044
10 Steps	0.019	0.019	0.019	0.027	0.043
20 Steps	0.019	0.019	0.019	0.027	0.042
40 Steps	0.035	0.037	0.038	0.045	0.062

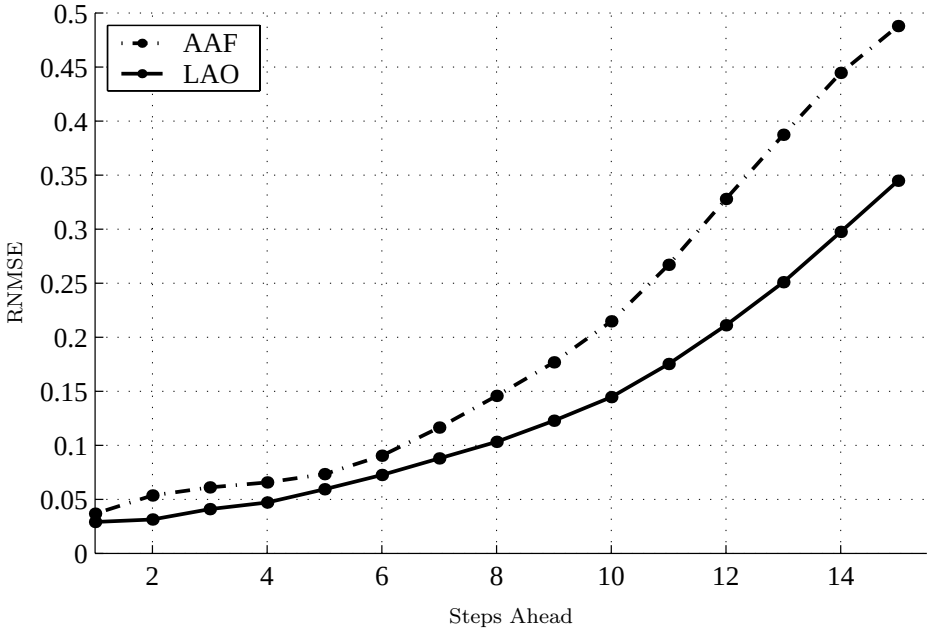


Fig. 5. The prediction error horizon of local averaging optimization (LAO) and adaptive analog forecasts (AAF) applied to the Lorenz time series.

LAO achieves this through the generalized cyclic coordinate method. AAF is also limited to direct prediction whereas the new method can be used for either direct or iterated prediction. In the latter case, LAO minimizes the multi-step cross-validation error.

To compare the performance of these two methods of prediction, both algorithms were optimized for one hundred iterations on the same time series segment and then applied to the same test segments. Both algorithms were applied to the Lorenz and Mackey-Glass time series. Figs. 5 and 6 show the prediction horizons for one to fifteen steps ahead. In all cases LAO performed better than AAF.

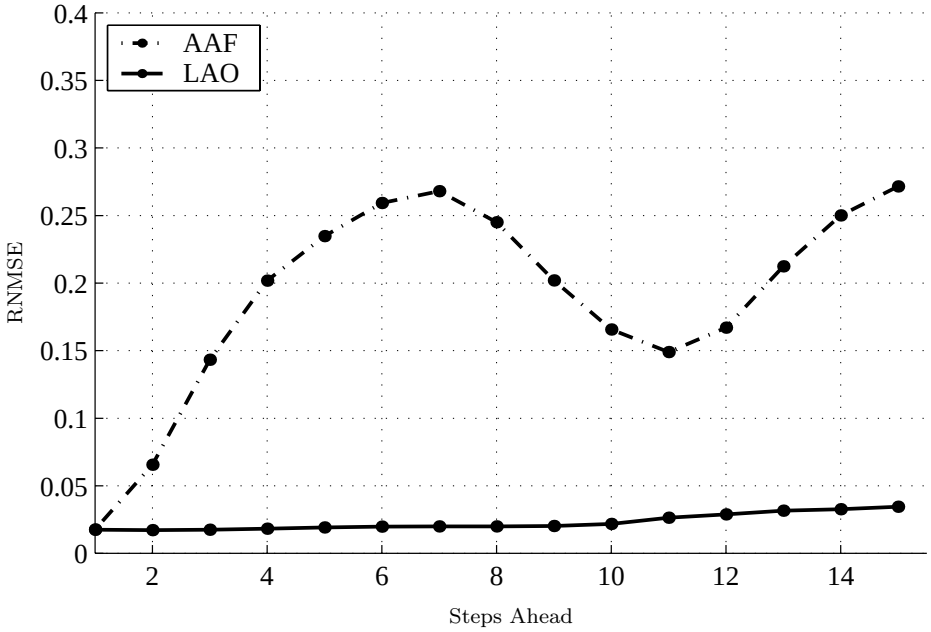


Fig. 6. The prediction error horizon of local averaging optimization (LAO) and adaptive analog forecasts (AAF) applied to the Mackey-Glass time series.

7 Conclusion

This paper describes a new method of local averaging optimization (LAO) for time series prediction. This relieves the user from the common practice of specifying critical parameter values and gives the user control of the computation used for optimization.

One of the benefits of LAO is that the effective embedding dimension is automatically optimized to maximize model accuracy along with the metric weights. This is a different approach than most previous methods that seek a minimal embedding dimension without consideration of model accuracy.

LAO incorporates an efficient optimization algorithm that blends the cyclic coordinate method for discrete-valued parameters with efficient gradient-based optimization algorithms for continuous-valued parameters. This optimization can be applied to an iterated predictor through a recursive algorithm that efficient calculates the gradient of the multi-step-ahead cross-validation error.

Numerous empirical simulations were performed to determine how sensitive the algorithm is to shallow local minima. As long as the number of steps ahead is reasonable ($n_s \leq 10$), the optimized parameters are not sensitive to the initial values [4].

[4] This method of optimization has also been applied to local linear models to jointly

LAO was compared to adaptive analog forecasts (AAF), the only other algorithm that has been proposed for metric optimization of local models for time series prediction. LAO consistently generated more accurate predictions than AAF.

A Local Average Gradients

This appendix describes a derivation of the gradient of the local averaging model output with respect to the metric parameters and with respect to the model inputs. These gradients are used to calculate the gradient of the multi-step cross-validation error (Equation 9) with respect to the metric parameters, as expressed by Equation 12.

A.1 Metric Gradient

The expression for $\nabla_\lambda \hat{g}^-(x_t)$ can be obtained from Equation 2 as

$$\begin{aligned}\nabla_\lambda \hat{g}^-(x_t) &= \nabla_\lambda \frac{\sum_{i=1,i\neq t}^{n_p} w_i^2 y_{i+1}}{\sum_{i=1,i\neq t}^{n_p} w_i^2}, \\ &= \frac{1}{\sum_{i=1,i\neq t}^{n_p} w_i^2} \left(\sum_{i=1,i\neq t}^{n_p} y_i \nabla_\lambda w_i^2 - \hat{g}^-(x_t) \sum_{i=1,i\neq j}^{n_p} \nabla_\lambda w_i^2 \right).\end{aligned}$$

This reduces the problem to finding the gradient of the weighting function w_i^2 with respect to the metric parameters λ. Assuming the biweight function is used, this gradient can be obtained from Equations 3 and 6,

$$\nabla_\lambda w_i^2 = \nabla_\lambda \mathrm{B}(d_i^2, d_{k+1}^2) = \frac{\partial \mathrm{B}(d_i^2, d_{k+1}^2)}{\partial d_i^2} \nabla_\lambda d_i^2 + \frac{\partial \mathrm{B}(d_i^2, d_{k+1}^2)}{\partial d_{k+1}^2} \nabla_\lambda d_{k+1}^2.$$

The jth element of the gradient $\nabla_\lambda d_i^2$ can be obtained by calculating the partial derivative

$$\frac{\partial d_i^2}{\partial \lambda_j} = \frac{\partial}{\partial \lambda_j} \sum_{\ell=1}^{n_d} \lambda_\ell^2 (x_{t,\ell} - x_{i,\ell})^2 = 2\lambda_j (x_{t,j} - x_{i,j})^2.$$

These equations collectively describe how to calculate the gradient of the model output $g_\lambda^-(x_t)$ respect to the metric parameters λ.

optimize the ridge regression parameters [26]. The optimization is less effective on these models because local minima are more numerous.

A.2 Input Gradient

The steps for finding the input gradient are very similar to those for finding the metric gradient. The expression for $\nabla_{x_t}\hat{g}^-(x_t)$ can be obtained from Equation 2 as

$$\nabla_{x_t}\hat{g}^-_\lambda(x_t) = \frac{1}{\sum_{i=1,i\neq t}^{n_p} w_i^2}\left(\sum_{i=1,i\neq t}^{n_p} y_i \nabla_{x_t} w_i^2 - \hat{g}^-_\lambda(x_t)\sum_{i=1,i\neq t}^{n_p}\nabla_{x_t} w_i^2\right),$$

and the gradient of the metric weighting function w_i^2 can be obtained from Equations 3 and 6,

$$\nabla_{x_t} w_i^2 = \frac{\partial \mathrm{B}(d_i^2, d_{k+1}^2)}{\partial d_i^2}\nabla_{x_t} d_i^2 + \frac{\partial \mathrm{B}(d_i^2, d_{k+1}^2)}{\partial d_{k+1}^2}\nabla_{x_t} d_{k+1}^2.$$

The jth element of the gradient $\nabla_{x_t} d_i^2$ is then given by

$$\frac{\partial d_i^2}{\partial x_{t,j}} = \frac{\partial}{\partial x_{t,j}}\sum_{\ell=1}^{n_d}\lambda_\ell^2(x_{t,\ell} - x_{i,\ell})^2 = 2\lambda_j^2(x_{t,j} - x_{i,j}).$$

These equations collectively describe how to calculate the gradient of the model output with respect to the model inputs x_t.

References

[1] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1–5):11–73, February 1997.

[2] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley & Sons, Inc., second edition, 1993.

[3] Mauro Birattari, Gianluca Bontempi, and Hugues Bersini. Lazy learning meets the recursive least squares. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems*, pages 375–381. MIT Press, 1999.

[4] G. Bontempi, M. Birattari, and H. Bersini. Lazy learning for iterated time-series prediction. In *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*, pages 62–68, Katholieke Universiteit Leuven, Belgium, July 1998.

[5] G. Bontempi, M. Birattari, and H. Bersini. Local learning for iterated time-series prediction. In *Machine Learning: Proceedings of the Sixteenth International Conference*, pages 32–38, San Francisco, CA, 1999.

[6] Leo Breiman. Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6):2350–2383, 1996.

[7] Leo Breiman and Philip Spector. Submodel selection and evaluation in regression. the x-random case. *International Statistical Review*, 60(3):391–319, 1992.

[8] Th. Buzug and G. Pfister. Comparison of algorithms calculating optimal embedding parameters for delay time coordinates. *Physica D*, 58:127–137, 1992.

[9] Liangyue Cao. Practical method for determining the minimum embedding dimension of a scalar time series. *Physica D*, 110:43–50, 1997.

[10] Martin Casdagli. Nonlinear prediction of chaotic time series. *Physica D*, 35:335–356, 1989.

[11] Martin Casdagli, Deirdre Des Jardins, Stephen Eubank, J. Doyne Farmer, John Gibson, and James Theiler. Nonlinear modeling of chaotic time series: Theory and applications. In Jong Hyun Kim and John Stringer, editors, *Applied Chaos*, pages 335–380. John Wiley & Sons, Inc., 1992.

[12] Martin C. Casdagli and Andreas S. Weigend. Exploring the continuum between deterministic and stochastic modeling. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Time Series Prediction*, Santa Fe Institute Studies in the Sciences of Complexity, pages 347–366. Addison-Wesley, 1994.

[13] J. D. Farmer and John J. Sidorowich. Exploiting chaos to predict the future and reduce noise. In Yee Chung Lee, editor, *Evolution, Learning and Cognition*, pages 277–330. World Scientific, 1988.

[14] J. Doyne Farmer and John J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848, August 1987.

[15] Klaus Fraedrich and Bernd Rückert. Metric adaption for analog forecasting. *Physica A*, 253:379–393, 1998.

[16] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[17] P. García, J. Jiménez, A. Marcano, and F. Moleiro. Local optimal metrics and nonlinear modeling of chaotic time series. *Physical Review Letters*, 76(9):1449–1452, February 1996.

[18] Cyril Goutte and Jan Larsen. Adaptive metric kernel regression. *Journal of VLSI Signal Processing*, 26:155–167, 2000.

[19] Trevor Hastie and Robert Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):607–616, June 1996.

[20] Ioannis Katsavounidis, C.-C. Jay Kuo, and Zhen Zhang. Fast tree-structured nearest neighbor encoding for vector quantization. *IEEE Transactions on Image Processing*, 5(2):398–404, February 1996.

[21] D. Kugiumtzis. State space reconstruction parameters in the analysis of chaotic time series — the role of the time window length. *Physica D*, 95:13–28, 1996.

[22] D. Kugiumtzis. Assessing different norms in nonlinear analysis of noisy time series. *Physica D*, 105:62–78, 1997.

[23] David G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, January 1995.

[24] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, second edition, 1984.

[25] James McNames. A nearest trajectory strategy for time series prediction. In *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*, pages 112–128, Katholieke Universiteit Leuven, Belgium, July 1998.

[26] James McNames. *Innovations in Local Modeling for Time Series Prediction*. PhD thesis, Stanford University, 1999.

[27] James McNames, Johan Suykens, and Joos Vandewalle. Winning entry of the K.U. Leuven time-series prediction competition. *International Journal of Bifurcation and Chaos*, 9(8):1485–1500, August 1999.

[28] Daniel B. Murray. Forecasting a chaotic time series using an improved metric for embedding space. *Physica D*, 68:318–325, 1993.

[29] Tim Sauer, James A. Yorke, and Martin Casdagli. Embedology. *Journal of Statistical Physics*, 65(3):579–616, 1991.

[30] Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[31] Leonard A. Smith. Local optimal prediction: exploiting strangeness and the variation of sensitivity to initial condition. In *Philosophical Transactions of the Royal Society*, volume 348 of *A*, pages 371–381, 1994.

[32] F. Takens. Detecting strange attractors in turbulence. In D. A. Rand and L. S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 336–381. Springer-Verlag, 1981.

[33] Naoki Tanaka, Hiroshi Okamoto, and Masayoshi Naito. An optimal metric for predicting chaotic time series. *Japanese Journal of Applied Physics*, 34(1):388–394, January 1995.

[34] P. J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–60, October 1990.

[35] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–80, 1989.