Non-Linear Time Series Prediction by Systematic Data Exploration on a Massively Parallel Computer

Xiru Zhang
Kurt Thearling

SFI WORKING PAPER: 1994-07-045

SANTA FE INSTITUTE

Non-Linear Time-Series Prediction by Systematic Data Exploration on a Massively Parallel Computer

Xiru Zhang
PHZ Partners
One Cambridge Center
Cambridge MA 02142

Kurt Thearling
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142

xiru@phz.com kurt@think.com

Abstract

In this paper we describe the application of massively parallel processing (MPP) to the problem of time-series analysis. Time-series problems involve sequences of numbers (for example, the daily closing values of the stock market, EEG patterns of brainwave activity, or, as discussed in this paper, the temporal values from set of equations of motion). Often the problem of interest is the prediction of some future value(s) in the sequence using only past values. Taking advantage of the power of an MPP supercomputer, we describe techniques to perform exploratory data analysis on time-series problems in a quick and efficient manner.

1 Introduction

Problems concerning the analysis of a time-series have existed for a very long time. Ever since the creation of the first financial markets, people have tried (and usually failed) to predict the future. However, there are many time-series problems in science and engineering that are to some extent predictable (such as the occurrence of sun spots, or the failure rate of a product), and further improvement on their prediction accuracies is of great importance.

Predictive models used in practice can often be quite simple. In these cases the power of a massively parallel supercomputer is ususally unnecessary to when making predictions. However, the generation of the models often requires large amounts of computational power and memory, especially if one has to examine a large amount of historical data with no prior theory. It is this phase of the time-series analysis problem (sometimes called "exploratory data analysis") that the power of massively parallel computation can be efficiently leveraged. The increased memory and disk capacities of MPP supercomputers allow very large historical databases to be analyzed. We have also found that analysis algorithms are typically amenable to efficient parallel implementations which allow experiments to be quickly carried out. This is often of significance to a successful analysis project since it allows scientists to try out many different solutions on large training sets. In addition to evaluation of different algorithm parameters, radically different algorithms can be compared. It can also change the way people approach certain problems (to trade computer power for brain power).

All of our work described in this paper was done on the massively parallel Connection Machine CM-5, which can have up to 16,384 processing nodes (PN's), connected by a data network and a control network. Currently each PN is a SPARC processor which can be attached with 4 vector units capable of 160 MFLOPS. The CM-5 is a MIMD computer, but also has hardware support for SIMD operations, such as global synchronization and reduction. The data network has a "fat tree" topology with a random routing algorithm, and supports high speed arbitrary communications among PN's. Users interact with CM-5 through control processors, which are currently SUN-4's. The CM-5 has time-sharing operating system, allowing simultaneous use by multiple users. All high level data parallel programming languages supported by the CM-2 (*Lisp, C* and CMFortran) are also supported on the CM-5. Message passing is also supported from nodal MIMD programs.

In the remaining sections of this paper we describe approaches for mapping time-series analysis problems to MPP supercomputers and discuss some related issues in time-series analysis. We finish by describing our successes in using several algorithms on CM-5 for time-series prediction.

2 Prediction Techniques

2.1 Memory vs. Computation

We can see two extremes in the use of massively parallel computing power: memory-based algorithms and computation-based algorithms. The power and performance of memory-based algorithms is obtained by making use information contained in the large storage capacity (both RAM and disk) of MPP systems. Some of the common processing tasks that correspond to this extreme are Memory-based Reasoning, relational database operations, and text retrieval. Computation-based algorithms, on the other hand, get their power by making using of powerful number crunching capabilities of the parallel processors. Examples of this type of processing are numerical optimization, genetic algorithms, neural networks, and statistical modeling.

2.2 Memory Based Reasoning

Memory-based Reasoning (MBR) [15] is a form of K-Nearest Neighbor (KNN) classification technique [4]. It differs from traditional KNN algorithms in a qualitative, not quantitative, way. Most of previous applications of KNN made use of small (e.g., less than 10 megabytes) databases which were hand tuned to maximize performance. These applications were limited by the amount of data that could be used to quickly search for neighbors. By applying the KNN approach to much larger databases (hundreds/thousands of megabytes) and new methods of deriving matching metrics fully automatically, massively parallel computing transforms KNN into MBR. MBR applications rely on the ability to leverage the information contained in extremely large databases. Typical applications often involve hundreds of megabytes of data. In addition, data is often multidimensional, involving different types of information. There is little or no manual processing of the data (not even the removal of incorrect training examples) before it is used.

A parallel nearest neighbor search is used to look at all training examples simultaneously. Though distance metrics can be hand tuned to improve performance based on application specifics, simple measures can often produce very accurate results. Initial results can be quickly achieved since there are no models to create. Also, confidence levels can be generated using relative distances to matching and non-matching neighbors.

Some previously successful applications of MBR include:

- Protein Structure Prediction [19]
- Optical Character Recognition [14]
- Cardiac Patient Viability Prediction [17]
- Census Data Classification [3]

Although these examples are not strictly time-series problems, they do illustrate the potential for the analysis and prediction of very large amounts of data. One aspect of these problems that separates them from most previous time-series analysis problems is the amount of data to be analyzed. Some applications are currently working with single databases on the order of tens of gigabytes, with the expectation that they will grow by a factor of ten in as little as two years. Though these problems may seem very different from time-series forecasting, they actually involve similar techniques.

An example of an MBR-like approach from the area of time-series analysis is the work of Farmer and Sidorowich [5, 6] (which they called a "local linear" method). In their work, Farmer and Sidorowich attempted to predict the behavior of a time-series generated by a chaotic dynamical system. Their training set consisted of a time-series of up to 10,000 sampled from a trajectory along the attractor. The time-series was then transformed into a reconstructed state space using a delay space embedding [11, 16]. In the delay space embedding, each point in the state space is a vector X composed of time-series values corresponding to a sequence of d delay lags: $x_1(t) = x(t), x_2(t) = x(t-\tau), ..., x_d(t) = x(t-(d-1)\tau)$. For a D dimensional attractor, d must be at least as large as D. Takens [16] showed that for a noise-free system of dimension D, it is sufficient to choose $d = 2D + 1$.

To forecast a time-series value, Farmer and Sidorowich first transformed the value into the state space representation. The nearest k ($> d$) neighbors

in the state space representation were then located. A local linear map was created for the k neighbors and applied to the value to be forecast. The result of the mapping was the predicted value. Although higher dimensional maps (quadratic, etc.) could be used, Farmer and Sidorowich did not find significant improvements over the linear map. Using this approach, they were able to forecast time-series values for a number of systems (Mackey-Glass differential equation, Rayleigh-Benard convection, and Taylor-Couette flow) much more accurately than standard forecasting techniques (global linear autoregressive).

As stated earlier, there have been several very large MBR applications. The following examples were both performed on a Connection Machine. Of particular importance is the large amount of data that was used to train the algorithms. It would have been very difficult (if not impossible) to make full use of such large data sets using a traditional computer architecture.

The first memory-based MPP algorithm involves optical character recognition [14]. Optical character recognition is the problem of taking a bit-mapped array of pixels and correctly classifying it into an alphanumeric character category. For pixel arrays of size 128 by 128, the problem has 16384 dimensions. Smith and his colleagues [14] used 300,000 training examples to provide a knowledge base for an MBR system. This corresponds to 614 megabytes of data. Using very simple Hamming distance metrics, classifying an unknown character could be performed with an average accuracy of 99%. The technique also allowed for the introduction of concept of confidence by allowing the system to refuse to classify unknown characters whose nearest neighbors fell below a threshold distance. When the confidence measure was introduced, the system achieved 99.9% accuracy at 93% coverage (i.e., the system was not able to confidently classify 7% of the data).

Another application of MBR to large databases involved the classification of Census Bureau data [3]. In this case the problem required classification of free-text responses to questions about a respondent's occupation. These responses needed to be classified by occupation category (504 types) and industry category (232 types). Approximately 130,000 training examples were used, corresponding to 15 megabytes of data. When compared to an existing expert system used by the Census Bureau, the MBR approach achieved a 57% accuracy improvement for the occupation classification. The MBR system also achieved a 10% improvement for the industry classification over the expert system.

2.2.1 Parallel search for nearest neighbors

As we have stated, it is necessary for MBR systems to locate the k-nearest neighbors for a point in the multidimensional state space. A number of distance metrics can be used, including Hamming, Euclidean, Manhattan, and a host of others. For serial computers, a K-D Tree representation [10] can effectively reduce search complexity for the nearest neighbors when there is structure in data. But when there is little or unknown structure in data, searching all data elements in parallel may be the most effective solution.

In experiments on a financial time-series (daily S&P 500 closing prices), the second author compared several K-D tree algorithms with simple parallel search examining portions of all data points. The experimental data contained 6067 points (approximately 20 years worth of daily time-series data) embedded into a five dimensional delay space. The distance metric for a pair of data points was simply Euclidean distance:

$$D = \sqrt{\sum_{i=1}^{n} (x'_i - x_i)^2}$$

where n is number of state space dimensions, x' is the point whose neighbors are being searched for, x is the current candidate neighbor, and x_i is the position of point x in dimension i. The total number of operations required to evaluate the distance between two points in the state space is n squaring operations, n subtraction operations, and $n-1$ addition operations. The square root operation is unnecessary since the ordering of the distances is the same as the ordering of the distances squared.

When the K-D tree algorithm attempted to locate the five nearest neighbors for a test point (which was another point from the time-series that was removed from the training set), an average of 99.6% of the training set data points needed to be examined (i.e., the K-D tree traversal was able to prune only 0.5% of the search space). In a refinement of the K-D tree search algorithm, the search began in the leaf cell that the test data point mapped to. It was hoped that this would help by initializing the set of nearest neighbors to a good set of candidates and thereby allow the traversal of the tree be pruned subsequently in the search. This technique did improve the performance of the search but the improvement was not signification (an average of 99.5% of the training data points were examined).

Finally, the K-D tree search was replaced by a much simpler technique. In this approach, every data point was examined to see if it was one of the nearest neighbors. But instead of computing the entire distance from the test point, the distance was computed incrementally. The (square of the) distance corresponding to each dimension was added until all of the dimensions were included. If at any time the partial distance was greater than the the (square of) the furthest of the current set of K-nearest neighbors, that data point was discarded. So, although each of the data points in the state space were evaluated, only a fraction of the entire evaluation (n squaring operations, n subtraction operations, and $n-1$ addition operations) were performed. In experiments on the same data set, this technique performed only 45.6% of the possible operations (squaring, subtraction, and addition) necessary to locate neighbors from the training data.

This incremental search technique can be efficiently implemented in parallel using a local nearest neighbor heap for each processor and updating the entries incrementally. After the local neighbors are located, a global sort on the local neighbors can be used to find the global neighbors. In addition to the overall efficiency of this approach, the use of multiple processors can result in significant (near linear speedup) increases in the search performance.

2.3 Neural Networks

A variety of neural network approaches have been applied to the forecasting of nonlinear time-series. The work of Lapedes and Farber [8] as well as several of the papers included in [1] have demonstrated the ability of neural network models for time-series prediction. As will be described later, the application of MPP neural network algorithms to time-series analysis has been quite successful.

In this section we will review some of the work done in mapping neural network algorithms to massively parallel computers. For a more thorough description, see [13] and [9] and the references contained therein.

Neural network architectures are characterized by a collection of processing nodes connected by a series of weighted links. The relationship between the individual node's inputs and outputs is typically a nonlinear function (such as a sigmoid). By carefully choosing the weights for the links, a neural network can carry out complex mappings from global inputs to global outputs. The complicated issue in carrying this process out is in computing the

interconnection weights. Algorithms such as Backpropagation [12] are often used to perform this task.

To implement an algorithm on a massively parallel computer, we need first to examine what can be done in parallel in the algorithm. In Backpropagation neural net learning algorithm for example, (a) all computations for units at the same layer can be done in parallel, and (b) since the weight changes from different training examples are additive, a network can be trained on multiple examples at the same time (this is the so called "batch mode" training). We have implemented Backpropagation algorithm on the Connection Machine in several ways, all of which made use of one or both of the above parallelisms inherent in the algorithm. In all cases, a unit is assigned to a processor, thus the activation function can be computed locally within the processor. The difficult question is how the units at one layer send their outputs or errors to units at another layer through weighted links. Unfortunately, a discussion of this issue is beyond the scope of this paper, for details see [20]. In our recent implementation, a 128 processor CM-5 can perform 500 million weight updates per second for the Backpropagation algorithm.

3 Non-Linear Time-Series Prediction: An Example

This section describes our work on a time-series used in Santa Fe Time-Series Prediction and Analysis Competition (1991). Our Connection Machine neural network system gave the best prediction for this time-series in the competition [21].[1] This univariate time-series is synthetically generated, but no background information was provided during the competition. Thus it provided an objective judgement on the participating systems for this non-linear time-series.

In the absence of both specific information about the particular data set and general theories for non-linear time-series prediction, good performance has to come from careful analysis of the available data ("training data"). There are several important issues in time-series prediction:

[1] This paper describes our results both for the competition and further analysis afterwards. The goal of the competition was to predict the future values of the series. The answers were withheld until after the competition was closed.

- How large should the window size (dimension) be?
- What are the sampling lags for the past data?
- How can one predict multiple steps toward the future?

When generating a neural network model, one also has to decide:

- How many hidden units should be used? How many hidden layers?
- How large a training set is needed?
- How can one choose training parameters, such learning rate, training cycles, etc.?

For an MBR model, the following issues come up:

- How many neighbors should be used?
- How large a training set is needed?
- What sort of distance metric should be used?

Our answers to the these questions came from extensive exploration of the available data using a CM-5 massively parallel supercomputer.

3.1 The data set

The time-series was synthetically generated by numerically integrating the equations of motion for a damped, driven particle in the potential field:

$$V = a_4(x_1^2 + x_2^2 + x_3^2 + x_4^2)^2 - a_2\sqrt{x_1^2 + x_2^2} - a_1 x_1$$

The driving force is $A\sin(\omega t)$ in the x_3 direction, and the dissipation is $-\gamma \times velocity$. The value of a_1 has a small drift produced by integrating a Gaussian random variable, and the observable saved (i.e. the time-series) is

$$\sqrt{(x_1 - 0.3)^2 + (x_2 - 0.3)^2 + x_3^2 + x_4^2}$$

The equations of motion were integrated with a simple fixed-step 4th order Runge-Kutta routine. 100,000 data points were generated. For more details of this series, see [18]. As argued by the competition organizers, there are several advantages of using this time-series:

- Since it is synthetically generated, in principle everything about it is known. Thus we can check what kind of information and correlation can various prediction algorithms can discover. Also since no background information was provided initially, this is a "fair game" for all algorithms used.

- The time-series contains relatively high-dimensional dynamics. The configuration space has 9 degrees of freedom (4 position, 4 velocity, and 1 forcing time), which was picked to be large but still realistically accessible.

- A long time-series is available. To recognize the presence of such high-dimensional dynamics, large data sets are needed, and can obviously be easily generated here. This often is a problem with real world data.

We want to emphasize that in this paper we use this data set only as an example to demonstrate how to approach time-series prediction problem of this kind, i.e., large data sets with relatively little background information, and to show how supercomputers can not only reduce the time required to solve the problem, but may change the way one approaches such problems.

To simplify our discussion, we will focus on 5-step ahead prediction, i.e., given $x_t, x_{t-1}, ...x_{t-n}.....$, to predict x_{t+5}. The same techniques can be extended to other cases.

3.2 Simple Exploratory Data Analysis

Stationarity is an important property of time-series and was one of the first things we checked. Plots of the data, running means, and running standard deviations reveal that if the series is non-stationary it is only mildly so (see Figure 1). The persistence of the sample autocorrelation function (ACF) at long lags (see Figure 2), however, indicates that some periodicity or non-stationarity is likely to be present in the data (non-periodic and stationary time-series tend to have an exponential decay of ACF, see [2] for a detailed discussion on this subject). The irregularity and strength of the ACF (and partial ACF) also advocates the use of a reasonably large window of data for modeling, although there is no principled way of choosing the optimal window size from the ACF without knowing the correct functional form of the model.

Histograms of the data show two significant peaks and one minor peak, suggesting that the underlying dynamical system has at least three attractors[2] (see Figure 3).

3.3 Dimensional analysis of time-series data

An important aspect to time-series analysis is the choice of representation. As described earlier, a delay space embedding can be used to reconstruct a multidimensional representation. But an important question is "How many dimensions should be used in the representation?" For a time-series of unknown or unexplained origin, it is necessary to determine the dimension using one of a number of well-known algorithms from dynamical systems theory.

One of the commonly used algorithms is the "Grassberger/Procaccia" (G/P) correlation dimension algorithm [7]. In this algorithm, the goal is to observe the scaling of a correlation exponent as the radius of an d dimensional hypersphere is changed. As long as d is greater than the true correlation dimension, the exponent will scale with the dimension. This is usually determined by computing the dimension (sometimes using a sample of the time-series) for a small d. If the computed correlation dimension equals d, d is increased and the process is repeated until the correlation dimension stabilizes.

The exact calculation of the correlation is as follows. The sum

$$C(r) \;=\; \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\theta(\; r^2 \;-\; \sum_{k=1}^{d}(X_{i,k} \;-\; X_{j,k})^2 \;)$$

is called the correlation summation (where N is the number of data points in the time-series, $X_{i,k}$ is the k'th dimensional component for data point i, r is the correlation length, and $\theta()$ is the Heaviside function).

Grassberger and Procaccia have shown that for many time-series problems the correlation sum scales as a power function of r:

$$C(r) \;\propto\; r^{\nu}$$

where the exponent ν is very closely related to the actual correlation dimension D of the system. By plotting $C(r)$ versus r on a log-log graph, d is

[2] Actually the system has four attractors, but two were indistinguishable due to the observation function.

obtained by computing slope of the line (usually in the middle, with r neither too big nor too small - in those extremes $C(r)$ evaluates to constant values).

The time complexity for the calculation of the correlation dimension is $O(d \times N^2)$. For many times series problems, N can be quite large and thus it is often impossible to compute this for the entire series. But on a MPP, if the time-series data is replicated on each processor (which is reasonable given today's memory technologies that allow hundreds of megabytes of RAM per processor in a MPP), parallelizing the problem is quite simple. By partitioning the N^2 pairwise interactions among the parallel processors, it is possible to perform the calculation with only a single communication step at the end to combine the results. This allows for nearly linear speedup using a parallel algorithm. For a 100,000 element time-series and a 10 dimensional embedding, an optimized implementation of the algorithm requires approximately 500 billion floating point operations. By taking advantage of the vector units in the CM-5, we have been able to fully parallelize and vectorize the G/P correlation dimension algorithm. The current speed that has been obtained is approximately 57 MFLOPS per processing node. Using a 64 processor CM-5 (thus achieving an aggregate 3.6 GFLOP/second performance), the correlation dimension for the above example (100,000 data points, 10 dimensional embedding) was calculated for the entire dataset in just over two and a half minutes. Such a calculation would be significantly more expensive to perform on a workstation.

The results of the correlation dimension calculations for five different embedding lags (requiring a total of two and a half trillion floating point calculations) are shown in Figure 4. When we first attempted to calculate the dimension of the data set, both the topological dimension (i.e., the number of independent degrees of freedom available in the system) and the correlation dimension (which is guaranteed to be less than or equal to the topological dimension) were unknown. By incrementally increasing the embedding dimension, we determined that for most lag values the correlation dimension leveled off between 5.0 and 6.0. Table 1 lists the computed correlation dimension values (i.e., the slope of the curve in the linear region) for various lag values:

It should be noted that the computation of the correlation dimension is not an end it itself. The dimension simply provides a lower bound for the complexity of a predictive model. For the data set that we are considering in this paper, a good estimate of the lower bound would be six dimensions

Lag	d
1	$4.81 \pm .04$
2	$5.54 \pm .07$
3	$5.60 \pm .09$
4	$5.39 \pm .06$
5	$5.56 \pm .10$

Table 1: Computed correlation dimension (d) for increasing lag values for the experimental data set in a ten dimensional delay space embedding.

(this is consistent with the fact that there were nine degrees of freedom in the system - the correlation dimension is typically smaller than the topological dimension).

3.4 Prediction Strategies

3.4.1 Training and Test Data Sets

In this work the available data was divided into three sets:

- *Training Data Set* – used to train a prediction system; its size varies in different experiments in order to find the optimal size.
- *Cross Validation Set* – used to monitor the training, e.g, to decide whether the prediction system has overfitted the training set. Its size is fixed to 8K.
- *Test Set* – only used at the very end, to test the true performance of the system trained. It contains 8K data points.

3.4.2 Measures for Prediction Errors

Given a time-series $x_1, x_2, ...x_n$, the standard deviation

$$x_{sd} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - x_{mean})^2}$$

is essentially the "prediction error" if the mean of the time-series x_{mean} is always used as prediction. This is the best one can do if the series is a random walk. If we have a series of predictions, $\hat{x}_1, \hat{x}_2, ...\hat{x}_n$, then the error is defined as:

$$\hat{x}_{error} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}$$

In this work we used the ratio

$$Error = \frac{\hat{x}_{error}}{x_{sd}}$$

to measure the quality of the predictions. It basically shows the improvement over simply predicting the mean of the series.

3.4.3 Window Size and Sampling Lag

Given a time-series $x_t, x_{t-1}, ...x_{t-i}, ...$, in order to predict x_{t+n}, we had to decide how many previous data points were needed, which we call the *window size*. We also had to decide how to sample the past value. A constant lag sampling consists of:

$$x_t, x_{t-k}, x_{t-2k}, ...x_{t-ik}, ...$$

where k is called *sampling lag*. Both of these parameters were determined experimentally in this work.

3.4.4 Direct vs. Iterative Approaches

When predicting multiple steps into the future, we have several choices. One approach is to directly predict x_{t+n} from the sampled data, i.e., to construct a function F_n such that the prediction $\hat{x}_{t+n}$ is:

$$\hat{x}_{t+n} = F_n(x_t, x_{t-k}, x_{t-2k}, ...x_{t-ik}...)$$

Alternatively, we can construct function F_1 to make 1 step ahead prediction $\hat{x}_{t+1}$ first:

$$\hat{x}_{t+1} = F_1(x_t, x_{t-K}, x_{t-2K}, ...x_{t-NK})$$

and then to use $\hat{x}_{t+1}$ as if it were given, and iteratively apply function F_1 n times to predict x_{t+n} Still another approach is to start with F_1, but when moving one step forward, re-train it to get a new function F_2', that is:

$$\hat{x}_{t+1} = F_1(x_t, x_{t-k}, x_{t-2k}, ...x_{t-ik}...)$$
$$\hat{x}_{t+2} = F_2'(\hat{x}_{t+1}, x_{t-k+1}, x_{t-2k+1}, ...x_{t-ik+1}...)$$
$$.....$$

3.5 Neural Networks Approaches to Prediction

In the following discussion, all networks are fully connected, and the notion such as "$n_1 : n_2 : n_3 : n_4$" is used to denote a network with n_1 input units, two hidden layers of n_2 and n_3 units each and n_4 output units.

3.5.1 Number Of Hidden Layers

First we compared networks of different numbers of hidden layers on the identical training and test sets. These networks have roughly the same number of weights (the number of free parameters). Thus, the difference in prediction errors must come from the architecture (the arrangement of hidden units into different hidden layers). Table 2 shows the result. Even though in principle a one-hidden layer network can approximate any continuous function, this result shows that in practice with limited resources a two-hidden layer network seems to have more approximation power than the one with one-hidden layer. For this problem, increasing the number of hidden layers to three did not improve the result.

3.5.2 Sampling Lags

To predict x_{t+5} from $x_t, x_{t-k}, x_{t-2k}, ...x_{t-ik}...$, We tested sampling lags $k = 1$ and $k = 5$. Table 3 shows the results. Short sampling lag produced better predictions for this time-series, which suggests that local correlations are the most important factors in the underlying model.

Layer Units	#Weights	Training Set Error	Test Set Error
10:10:1	110	0.6846	0.7319
10:7:5:1	110	0.6453	0.6915
10:6:5:4:1	114	0.6486	0.6943

Table 2: Three networks trained on 16K training data points, with 8K cross validation data points and 8K test data points. All networks were trained up to 10,000 cycles. The task was to predict x_{t+5} using $x_t, x_{t-1}, ...x_{t-9}$.

Layer Units	Sampling Lag	Training Set Error	Test Set Error
10:7:5:1	1	0.6453	0.6915
10:7:5:1	5	0.6794	0.7288

Table 3: Two networks trained on 16K training data, with 8K cross validation data and 8K test dat. Both networks were trained up to 10,000 cycles. The task is to predict x_{t+5} using $x_t, x_{t-k}, ...x_{t-9k}$. $k = 1$ and $k = 5$ are shown here.

3.5.3 Window Size

Window size as defined above depends in part on the underlying dimensionality of the time-series. There are certain techniques to estimate the dimension. And Takens [16] has proven a theorem that in the noise free case, N points are necessary, and $2N + 1$ points are sufficient in reconstructing (predicting) a time-series with underlying dimension of N. However, with the presence of noise (which is the case of the data used in this work), that may not hold.

Our results in table 4 shows that for our time-series problem, window sizes around 20 gave the best results. When determining the best window size in this experimental fashion, one has to be careful about one pitfall: when a larger window does not give better performance than a smaller window it could mean two things: (1) the larger window does not contain any more useful information (in fact it may contain more noise); (2) the function approximator used is not powerful enough to capture the information in the larger window. Within the frame of neural networks, one can increase the number of hidden units (thus the power of the network) for a fixed window size to see if the performance improves. An example in our work is window 15 and 20, they gave similar performances with two hidden layers with units 7 and 5 each. But when more hidden units were used, window 20 proved to be better. The performance of a network may not increase as the window size increase, it in principle can actually decrease because a larger window also contains more "noise," making it more difficult for the network to learn the true correlation.

3.5.4 Training Set Size

Intuitively larger training data set should carry more information. However, for time-series analysis, we have to concern about the issue of *non-stationarity.* That is, certain properties of the series may change over time, looking too far back to the past may find things that are no longer valid. Table 5 shows our results in determining the training set size.

3.5.5 Number Of Hidden Units for Neural Networks

To choose the correct number of hidden units is important: too few hidden units do not provide enough approximation power, whereas too many units not only require more computer time to train the network, but also has the

Input	Hidden	Training Set Error	Test Set Error
5	7:5	0.6902	0.7385
10	7:5	0.6453	0.6951
15	7:5	0.6336	0.6708
20	7:5	0.6339	0.6714
30	7:5	0.6400	0.6858
15	20:20	0.6293	0.6730
20	20:20	0.5997	**0.6468**
20	30:30	0.6129	0.6486
30	30:30	0.6397	0.6879

Table 4: Experiments on different window sizes. Data used: 16K training data, 8K cross validation data and 8K test data; 10,000 training cycles for all networks; 5 step ahead prediction.

Layers	Training Data	Training Set Error	Test Set Error
20:7:5:1	8K	0.6773	0.6982
20:7:5:1	16K	0.6339	0.6714
20:7:5:1	32K	0.6765	0.6972

Table 5: Experiments on different training data set sizes. Data used: 8K cross validation data and 8K test data; 10,000 training cycles for all networks; 5 step ahead prediction.

Input	Hidden	Training Set Error	Test Set Error
20	7:5	0.6339	0.6714
20	10:10	0.6202	0.6609
20	20:20	0.5997	0.6468
20	30:30	0.6129	0.6486

Table 6: Experiments on different hidden units. Data used: 16K training data, 8K cross validation data and 8K test data; 10,000 training cycles for all networks; 5 step ahead prediction.

danger of overfitting the training data. That is why we had a cross-validation set available to monitor the training process. Table 6 shows our results in determining the optimal number of hidden units. Two hidden layers of 20 units each produced the best prediction in our experiments.

This results indeed shows that too many hidden units did not help.

3.5.6 Direct, Iterative and "Re-training" Approaches

Table 7 shows the results of three different approaches for predicting multiple steps ahead of a time-series. *Direct approach* uses $x_t, x_{t-1}, ...x_{t-19}$ as inputs to the neural network and trains the net to predict x_{t+5} directly. *Iterative approach* uses the same inputs to train a network to predict x_{t+1} first, and then uses the same network to predict x_{t+2}, taking the first prediction as one of the inputs: $\hat{x}_{t+1}, x_t, x_{t-1}, ...x_{t-18}$, and to iterate this procedure until x_{t+5} is reached. *Re-training* approach is similar to iterative approach, except that at each step of iteration, instead of using the same original network, it retrains a new network, i.e., there are 5 networks for predicting x_{t+1} to x_{t+5}.

The re-training approach actually produced better prediction than iterative approach not only for x_{t+5}, but for every step along the way. See Table 8.

Approach	Network	Training Set Error	Test Set Error
iterative	20:20:20:1	0.7951	0.8260
direct	20:20:20:1	0.5997	0.6468
re-training	20:20:20:1	0.5882	0.6271

Table 7: Experiments on different strategies for multiple steps ahead prediction. Data used: 16K training data, 8K cross validation data and 8K test data; 10,000 training cycles for all networks; 5 step ahead prediction.

Approach	Step 1	Step 2	Step 3	Step 4	Step 5
iterative	0.1390	0.3313	0.5276	0.6951	0.8260
re-training	0.1390	0.3133	0.4340	0.5440	0.6271

Table 8: Comparison between iterative and re-training approaches for 1 to 5 step ahead predictions. Only the prediction errors on the test set are shown in this table. Data used: 16K training data, 8K cross validation data and 8K test data; 10,000 training cycles for all networks.

3.6 Memory Based Approaches to Prediction

As described earlier, Farmer and Sidorowich popularized the use of memory based approaches for time-series prediction. By finding "neighbors" in a multidimensional phase space, these neighbors can be used to create a predictive mapping function which takes the current point as an input. Previous research has shown that use of simple linear maps often provide quite satisfactory performance [5].

Assuming that a d dimensional embedding has already been chosen, the only remaining variable in the procedure is to determine the number of neighbors to be used to create the map. If the number of neighbors M is chosen to be the total number of training data points, the algorithm reduces to a global linear fit. By choosing M to be relatively small, the degree of locality can be varied. In the work presented here, a linear least-squares regression algorithm is used to create a mapping over a d dimensional space using M data points (i.e., the neighbors). As long as $M \geq d$, the mapping is guaranteed to be computable.

In Table 9, the results of using an MBR based prediction strategy are shown. In all cases, the number of neighbors M was chosen to be a function of d ($M = 2 \times d$). As can be seen, the performance of this technique was reasonably effective, when compared to the neural network approach. The best error rate observed by MBR was .684 compared with a value of .627 for the neural network model.

One thing to notice is that when the lag is increased beyond one, the performance actually degrades when the dimension is increased. This is consistent with our experience with the data which showed that there was little predictive information from points distanced greater than thirty time-steps apart. For a model with a dimension of ten and a lag or three, the maximum distance in the embedding is thirty time steps. One would expect that this model would not perform as well as a model whose window of data points was within a smaller time step radius. In our experiments, the best results were obtained when the dimension of the delay-space representation was set to twenty, with a lag of one. For both higher and lower dimensional embeddings, the performance was slightly degraded.

Since the power of an MBR approach to prediction is in the training data, one would think that increasing the training set size might improve performance (in the examples discussed earlier in this paper, that had often

Dimension	Lag = 1	Lag = 3	Lag = 5
5	0.915689	0.818361	0.837462
10	0.766500	0.821324	0.950947
15	0.704076	0.925727	1.119455
20	0.684870	1.035484	1.204141
25	0.715877	N/A	N/A

Table 9: Error values for memory based prediction with 16K training data and 8K test data; 5 step ahead prediction; number of neighbors = $2 \times d$.

Dimension	Lag = 1
5	0.880733
10	0.775553
15	0.697443
20	0.695274

Table 10: Results for memory based prediction with 32K training data and 8K test data; 5 step ahead prediction; number of neighbors = $2 \times d$.

been the case). Table 10 contains the results when running the MBR algorithm on a training set twice the size as previously described. Although the performance, for lower dimensional embeddings, starts off better, the performance plateaus at a slightly higher value. This is also consistent with our experience with this data set. We found that the time-series was relatively non-stationary and that data points located even further in the past than in the original training set provided little predictive information.

4 Summary

We have described several techniques to analyze time-series problems using massively parallel computers. By leveraging the power available in such a computer system, the generation and comparison of different predictive

models can be efficient carried out. We discussed two of these approaches (neural networks and memory based reasoning) as they were applied to an existing time-series problem. Both techniques provided effective solutions.

5 Acknowledgements

The authors would like to thank David Waltz, Jim Hutchinson, Tommy Poggio, and the Santa Fe Institute for their support of this research.

References

[1] M. Casdagli and S. Eubank, *Nonlinear Modeling and Forecasting*, Reading, MA: Addison-Wesley, 1992.

[2] C. Chatfield, *The Analysis of Time-Series*, Chapman and Hall, 1989.

[3] R.H. Creecy, B.M. Masand, S.J. Smith, and D.L. Waltz, "Trading MIPS and Memory for Knowledge Engineering: Automatic Classification of Census Returns on a Massively Parallel Supercomputer," *Communications of the ACM*, August 1992.

[4] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, CA: IEEE Computer Society Press, 1991.

[5] J.D. Farmer and J.J. Sidorowich, "Predicting Chaotic Time-Series," *Physical Review Letters*, 59(8):845, 1987.

[6] J.D. Farmer and J.J. Sidorowich, "Exploiting Chaos to Predict the Future and Reduce Noise," in *Evolution, Learning, and Cognition*, edited by Y.D. Lee. Singapore: World Scientific, 1988.

[7] P. Grassberger and I. Procaccia, "Characterization of Strange Attractors," *Physical Review Letters*, 50(5):346, 1983.

[8] A. Lapedes and R. Farber, "Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling," Los Alamos National Laboratory Technical Report LA-UR-87-2662, July 1987.

[9] T. Nordström and B. Svensson, "Using and Designing Massively Parallel Computers for Artificial Neural Networks," *Journal of Parallel and Distributed Computing*, 14:260, 1992.

[10] S. Omohundro, "Efficient Algorithms With Neural Network Behavior," *Complex Systems*, 1:273, 1987.

[11] N.H. Packard, J.P. Crutchfield, J.D. Farmer, and R.S. Shaw, "Geometry from a Time-Series," *Physical Review Letters*, 45:9, 1980.

[12] D.E. Rummelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, edited by J. McClelland and D.E. Rummelhart, Cambridge, MA: MIT Press, 1986.

[13] A. Singer, "Implementations of Artificial Neural Networks on the Connection Machine, *Parallel Computing*, 14:305, 1990.

[14] S. Smith, et al., in preparation.

[15] C. Stanfill and D. L. Waltz. "Toward Memory-based Reasoning." *Communications of the ACM*, 29:1213, 1986.'

[16] F. Takens, "Detecting Strange Attractors in Fluid Turbulence," in *Dynamical Systems and Turbulence*, editem by D. Rand and L.-S. Young, Berlin: Springer-Verlag, 1981.

[17] D.L. Waltz, "Applications of the Connection Machine," *IEEE Computer* 20(1):85, 1987.

[18] *Time Series Prediction: Forecasting the Future and Understanding the Past*, edited by A. Weigend and N. Gershenfeld, Reading, MA: Addison-Wesley, 1994.

[19] X. Zhang, D.L. Waltz, J.P. Mesirov, "Protein Structure Prediction by Memory-Based Reasoning," *Proceedings of the Case-Based Reasoning Workshop*, Pensacola Beach, FL, May 1989, pp. 1-5.

[20] X. Zhang, "Backpropagation Algorithm On the Connection Machine Systems," *Proceedings of International Joint Conference On Neural Networks*, Beijing, 1992, pp. III.210 - III.215.

[21] X. Zhang and J. Hutchinson, "Practical Issues in Nonlinear Time-Series Prediction," in *Predicting the Future and Understanding the Past*, edited by A. Weigend and N. Gershenfeld, Reading, MA: Addison-Wesley, 1993.