Noisy Time Series Prediction using Symbolic Representation and Recurrent Neural Network Grammatical Inference

Steve Lawrence[1,*], Ah Chung Tsoi[2], C. Lee Giles[1,†‡]
{lawrence,giles}@research.nj.nec.com, Ah_Chung_Tsoi@uow.edu.au

[1] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
[2] Faculty of Informatics, University of Wollongong, NSW 2522 Australia

Technical Report
UMIACS-TR-96-27 and CS-TR-3625
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

Abstract

Financial forecasting is an example of a signal processing problem which is challenging due to small sample sizes, high noise, non-stationarity, and non-linearity. Neural networks have been very successful in a number of signal processing applications. We discuss fundamental limitations and inherent difficulties when using neural networks for the processing of high noise, small sample size signals. We introduce a new intelligent signal processing method which addresses the difficulties. The method uses conversion into a symbolic representation with a self-organizing map, and grammatical inference with recurrent neural networks. We apply the method to the prediction of daily foreign exchange rates, addressing difficulties with non-stationarity, overfitting, and unequal *a priori* class probabilities, and we find significant predictability in comprehensive experiments covering 5 different foreign exchange rates. The method correctly predicts the direction of change for the next day with an error rate of 47.1%. The error rate reduces to around 40% when rejecting examples where the system has low confidence in its prediction. The symbolic representation aids the extraction of symbolic knowledge from the recurrent neural networks in the form of deterministic finite state automata. These automata explain the operation of the system and are often relatively simple. Rules related to well known behavior such as trend following and mean reversal are extracted.

* http://www.neci.nj.nec.com/homepages/lawrence
† Lee Giles is also with the Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.
‡ http://www.neci.nj.nec.com/homepages/giles.html

December 16, 1997, 13:36

1 Introduction

1.1 Predicting Noisy Time Series Data

The prediction of future events from noisy time series data is commonly done using various forms of statistical models [20]. Typical neural network models are closely related to statistical models, and estimate Bayesian *a posteriori* probabilities when given an appropriately formulated problem [38]. Neural networks have been very successful in a number of pattern recognition applications. For noisy time series prediction, neural networks typically take a delay embedding of previous inputs[1] which is mapped into a prediction. However, high noise, high non-stationarity time series prediction is fundamentally difficult for these models:

1. The problem of learning from examples is fundamentally ill-posed, i.e. there are infinitely many models which fit the training data well, but few of these generalize well. In order to form a more accurate model, it is desirable to use as large a training set as possible. However, for the case of highly non-stationary data, increasing the size of the training set results in more data with statistics that are less relevant to the task at hand being used in the creation of the model.

2. The high noise and small datasets make the models prone to overfitting. Random correlations between the inputs and outputs can present great difficulty. The models typically do not explicitly address the temporal relationship of the inputs – e.g. they do not distinguish between those correlations which occur in temporal order, and those which do not.

Our first step is to use recurrent neural networks (RNNs). Recurrent neural networks employ feedback connections and have the potential to represent certain computational structures in a more parsimonious fashion [10]. RNNs address the temporal relationship of their inputs by maintaining an internal state. RNNs are biased towards learning patterns which occur in temporal order – i.e. they are less prone to learning random correlations which do not occur in temporal order.

However training RNNs tends to be difficult with high noise data, with a tendency for long-term dependencies to be neglected (experiments reported in [6] found a tendency for recurrent networks to take into account short-term dependencies but not long-term dependencies), and for the network to fall into a naive solution such as always predicting the most common output. We address this problem by converting the time series into a sequence of symbols using a self-organizing map (SOM). The problem then becomes one of grammatical inference – the prediction of a given quantity from a sequence of symbols. It is then possible to take advantage of the known capabilities of recurrent neural networks to learn grammars [50] in order to capture any predictability in the evolution of the series.

[1] For example, $x(t), x(t-1), x(t-2), \ldots, x(t-N+1)$ form the inputs for a delay embedding of the previous N values of a series [49, 41].

The use of a recurrent neural network is significant for two reasons: firstly, the temporal relationship of the series is explicitly modeled via internal states, and secondly, it is possible to extract rules from the trained recurrent networks in the form of deterministic finite state automata [17][2]. The symbolic conversion is significant for a number of reasons: the quantization effectively filters the data or reduces the noise, the RNN training becomes more effective, and the symbolic input facilitates the extraction of rules from the trained networks.

Financial time series data typically contains high noise and significant non-stationarity. In this paper, the noisy times series prediction problem considered is the prediction of foreign exchange rates. A brief overview of foreign exchange rates is presented in the next section.

1.2 Foreign Exchange Rates

The foreign exchange market is the world's largest market, with more than $US 1.3 trillion changing hands every day (`http://www.c-c.ch/forex.htm`). The most important currencies in the market are the US dollar (which acts as a reference currency), the Japanese Yen, the British Pound, the German Mark, and the Swiss Franc [32]. Foreign exchange rates exhibit very high noise, and significant non-stationarity. Many financial institutions evaluate prediction algorithms using the percentage of times that the algorithm predicts the right trend from today until some time in the future [32]. Hence, this paper considers the prediction of the direction of change in foreign exchange rates for the next business day.

The remainder of this paper is organized as follows: section 2 describes the data set we have used, section 3 discusses fundamental issues such as ill-posed problems, the curse of dimensionality, and vector space assumptions, and section 4 discusses the efficient market hypothesis and prediction models. Section 5 details the system we have used for prediction. Comprehensive test results are contained in section 6, and section 7 considers the extraction of symbolic data. Section 8 provides concluding remarks, and Appendix A provides full simulation details.

2 Exchange Rate Data

The data we have used is publically available (`http://www.cs.colorado.edu/~andreas/Time-Series/Data/Exchange.Rates.Daily`) and was first used by Weigend et al. [48]. The data consists of daily closing bids for five currencies (German Mark (DM), Japanese Yen, Swiss Franc, British Pound, and Canadian Dollar) with respect to the US Dollar and is from the Monetary Yearbook of the Chicago Mercantile Exchange. There are 3645 data points for each exchange rate covering the period September 3, 1973 to May 18, 1987. In contrast to Weigend et al., this work

[2]Rules can also be extracted from feedforward networks [21, 35, 44, 2, 39, 25], however the recurrent network approach and deterministic finite state automata extraction seem particularly suitable for a time series problem.

considers the prediction of all five exchange rates in the data and prediction for all days of the week instead of just Mondays.

3 Fundamental Issues

This section briefly discusses fundamental issues related to learning by example using techniques such as multi-layer perceptrons (MLPs): ill-posed problems, the curse of dimensionality, and vector space assumptions. These help form the motivation for our new method.

The problem of inferring an underlying probability distribution from a finite set of data is fundamentally an ill-posed problem because there are infinitely many solutions [26], i.e. there are infinitely many functions which fit the training data exactly but differ in other regions of the input space. The problem becomes well-posed only when additional constraints are used. For example, the constraint that a limited size MLP will be used might be imposed. In this case, there may be a unique solution which best fits the data from the range of solutions which the model can represent. The implicit assumption behind this constraint is that the underlying target function is "smooth". Smoothness constraints are commonly used by learning algorithms [14]. Without smoothness or some other constraint, there is no reason to expect a model to perform well on unseen inputs (i.e. generalize well).

Learning by example often operates in a vector space, i.e. a function mapping $\mathcal{R}^n$ to $\mathcal{R}^m$ is approximated. However, a problem with vector spaces is that the representation does not reflect any additional structure within the data. Measurements that are related (e.g. from neighboring pixels in an image or neighboring time steps in a time series) and measurements that do not have such a relationship are treated equally. These relationships can only be inferred in a statistical manner from the training examples.

The use of a recurrent neural network instead of an MLP with a window of time delayed inputs introduces another assumption by explicitly addressing the temporal relationship of the inputs via the maintenance of an internal state. This can be seen as similar to the use of hints [1] and should help to make the problem less ill-posed.

The *curse of dimensionality* refers to the exponential growth of hypervolume as a function of dimensionality [5]. Consider $\mathbf{x}_i \in \mathcal{R}^n$. The regression, $y = f(\mathbf{x})$ is a hypersurface in $\mathcal{R}^n$. If $f(\mathbf{x})$ is arbitrarily complex and unknown then dense samples are required to approximate the function accurately. However, it is hard to obtain dense samples in high dimensions[3]. This is the "curse of dimensionality" [5, 14, 15]. The relationship between the sampling density and the number of points required is $\approx N^{\frac{1}{n}}$ [15] where n is the dimensionality of the input space and N is the number of points. Thus, if N_1 is the number of points for a given sampling density in 1 dimension, then in order to keep the same density as the dimensionality is increased, the number of points must increase according to N_1^n.

It has been suggested that MLPs do not suffer from the curse of dimensionality [13, 4]. However, this is not true in general (although MLPs may cope better than other models). The apparent avoidance of the curse of dimensionality in [4] is due to the fact that the function spaces considered are more and more constrained as the input dimension increases [29]. Similarly, smoothness conditions must be satisfied for the results of [13].

The use of a recurrent neural network is important from the viewpoint of the curse of dimensionality because the RNN can take into account greater history of the input. Trying to take into account greater history with an MLP by increasing the number of delayed inputs results in an increase in the input dimension. This is problematic, given the small number of data points available. The use of a self-organizing map for the symbolic conversion is also important from the viewpoint of the curse of dimensionality. As will be seen later, the topographical order of the SOM allows encoding a one dimensional SOM into a single input for the RNN.

4 The Efficient Market Hypothesis

The Efficient Market Hypothesis (EMH) was developed in 1965 by E. Fama [11, 12] and found broad acceptance in the financial community [33, 45, 3]. The EMH, in its weak form, asserts that the price of an asset reflects all of the information that can be obtained from past prices of the asset, i.e. the movement of the price is unpredictable. The best prediction for a price is the current price and the actual prices follow what is called a random walk. The EMH is based on the assumption that all news is promptly incorporated in prices; since news is unpredictable (by definition), prices are

[3]Kolmogorov's theorem shows that any continuous function of n dimensions can be completely characterized by a one dimensional continuous function. Specifically, Kolmogorov's theorem [28, 29, 30, 15] states that for any continuous function:

$$f(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{2n+1} g_f \left(\sum_{i=1}^{n} \lambda_i Q_j(x_i) \right) \tag{1}$$

where $\{\lambda_i\}_1^n$ are universal constants that do not depend on f, $\{Q_j\}_1^{2n+1}$ are universal transformations which do not depend on f, and $g_f(u)$ is a continuous, one-dimensional function which totally characterizes $f(x_1, x_2, \ldots, x_n)$ (g_f is typically highly non-smooth). In other words, for any continuous function of n arguments, there is a one dimensional continuous function that completely characterizes the original function. As such, it can be seen that the problem is not so much the dimensionality, but the complexity of the function [15], i.e. the curse of dimensionality essentially says that in high dimensions, as fewer data points are available, the target function has to be simpler in order to learn it accurately from the given data.

unpredictable. Much effort has been expended trying to prove or disprove the EMH. Current opinion is that the theory has been disproved [42, 24], and much evidence suggests that the capital markets are not efficient [34].

If the EMH was true, then a financial series could be modeled as the addition of a noise component at each step:

$$x(k+1) = x(k) + \epsilon(k) \tag{2}$$

where $\epsilon(k)$ is a zero mean Gaussian variable with variance σ. The best estimation is:

$$\hat{x}(k+1) = x(k) \tag{3}$$

In other words, if the series is truly a random walk, then the best estimate for the next time period is equal to the current estimate. Now, if it is assumed that there is a predictable component of the series then it is possible to use:

$$x(k+1) = x(k) + f(x(k), x(k-1), \ldots, x(k-n+1)) + \epsilon(k) \tag{4}$$

where $\epsilon(k)$ is a zero mean Gaussian variable with variance σ, and $f(\cdot)$ is a non-linear function in its arguments. In this case, the best estimate is given by:

$$\hat{x}(k+1) = x(k) + f(x(k), x(k-1), \ldots, x(k-n+1)) \tag{5}$$

Prediction using this model is problematic as the series often contains a trend. For example, a neural network trained on section A in figure 1 has little chance of generalizing to the test data in section B, because the model was not trained with data in the range covered by that section.

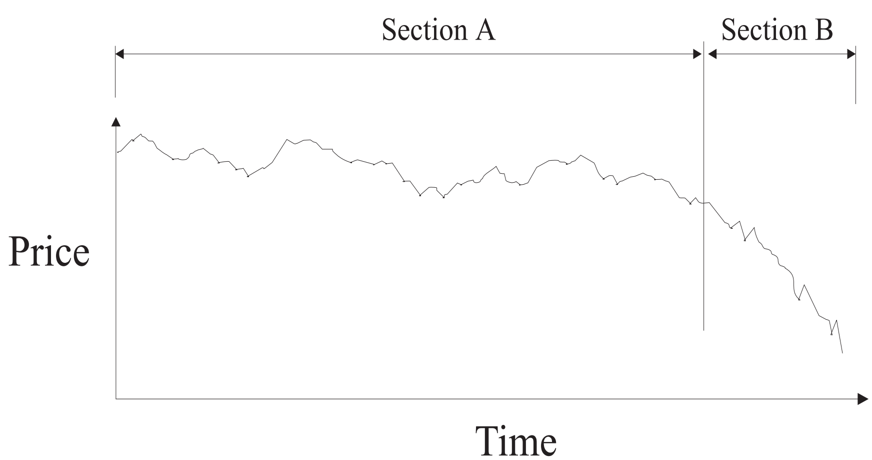

Figure 1. An example of the difficulty in using a model trained on a raw price series. Generalization to section B from the training data in section A is difficult because the model has not been trained with inputs in the range covered by section B.

A common solution to this is to use a model which is based on the first order differences (equation 7) instead of the raw time series [20]:

$$\delta(k+1) = f(\delta(k), \delta(k-1), \ldots, \delta(k-n+1)) + \nu(k) \tag{6}$$

where

$$\delta(k+1) \triangleq x(k+1) - x(k) \tag{7}$$

and $\nu(k)$ is a zero mean Gaussian variable with variance σ. In this case the best estimate is:

$$\hat{\delta}(k+1) = f(\delta(k), \delta(k-1), \ldots, \delta(k-n+1)) \tag{8}$$

This typically has the effect of reducing the non-stationarity of the series.

5 System Details

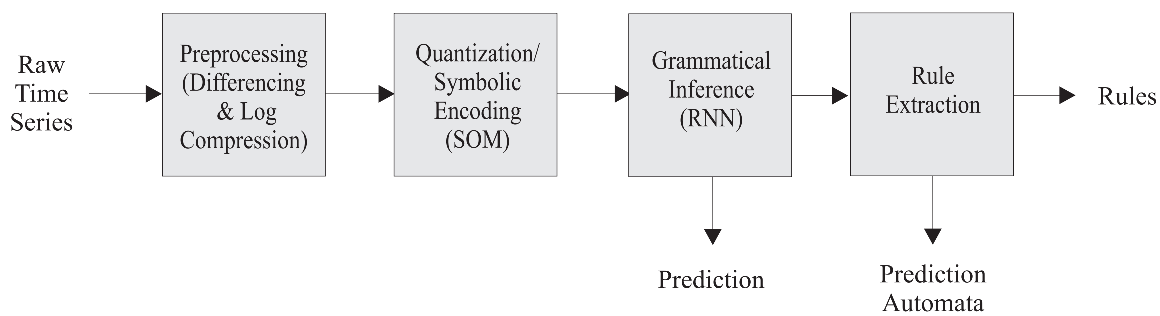

Figure 2. A high-level block diagram of the system used.

A high-level block diagram of the system used is shown in figure 2. The raw time series values are $y(k)$, $k = 1, 2, \ldots, N$ where $y(k) \in \mathcal{R}$. These denote the daily closing bids of the financial time series. The first difference of the series, $y(k)$, is taken as follows:

$$\delta(k) = y(k) - y(k-1) \tag{9}$$

This produces $\delta(k)$, $\delta(k) \in \mathcal{R}$, $k = 1, 2, \ldots, N-1$. In order to compress the dynamic range of the series and reduce the effect of outliers, a log transformation of the data is used:

$$x(k) = \text{sign}(\delta(k))(\log(|\delta(k)| + 1)) \tag{10}$$

resulting in $x(k), k = 1, 2, \ldots, N-1$, $x(k) \in \mathcal{R}$. As is usual, a delay embedding of this series is then considered [49]:

$$\mathbf{X}(k, d_1) = (x(k), x(k-1), x(k-2), \ldots, x(k-d_1+1)) \tag{11}$$

where d_1 is the delay embedding dimension and is equal to 1 or 2 for the experiments reported here. $\mathbf{X}(k, d_1)$ is a state vector. This delay embedding forms the input to the SOM. Hence, the SOM input is a vector of the last d_1 values of the log transformed differenced time series. The output of the SOM is the topographical location of the winning node. Each node represents one symbol in the resulting grammatical inference problem. A brief description of the self-organizing map is contained in the next section.

The SOM can be represented by the following equation:

$$S(k) = g(\mathbf{X}(k, d)) \tag{12}$$

where $S(k) \in [1, 2, 3, \ldots n_s]$, and n_s is the number of symbols (nodes) for the SOM. Each node in the SOM has been assigned an integer index ranging from 1 to the number of nodes.

An Elman recurrent neural network is then used[4] which is trained on the sequence of outputs from the SOM. The Elman network was chosen because it is suitable for the problem (a grammatical inference style problem) [10], and because it has been shown to perform well in comparison to other recurrent architectures (e.g. see [31]). For the Elman network:

$$\mathbf{O}(k+1) = \mathbf{C}^T \mathbf{z}_k + c_0 \tag{13}$$

and

$$\mathbf{z}_k = F_{n_h}(\mathbf{A}\mathbf{z}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{b}) \tag{14}$$

where $\mathbf{C}$ is a $n_h \times n_o$ vector representing the weights from the hidden layer to the output nodes, n_h is the number of hidden nodes, n_o is the number of output nodes, c_0 is a scalar, $\mathbf{z}_k$, $\mathbf{z}_k \in \mathcal{R}^{n_h}$, is an $n_h \times 1$ vector, denoting the outputs of the hidden layer neurons. $\mathbf{u}_k$ is a $d_2 \times 1$ vector as follows, where d_2 is the embedding dimension used for the input window of symbols that is presented to the SOM:

$$\mathbf{u}_k = \begin{bmatrix} S(k) \\ S(k-1) \\ S(k-2) \\ \ldots \\ S(k-d_2+1) \end{bmatrix} \tag{15}$$

$\mathbf{A}$ and $\mathbf{B}$ are matrices of appropriate dimensions which represent the feedback weights from the hidden nodes to the hidden nodes and the weights from the input layer to the hidden layer respectively. F_{n_h} is a $n_h \times 1$ vector containing the sigmoid functions. $\mathbf{b}$ is an $n_h \times 1$ vector, denoting the bias of each hidden layer neuron. $\mathbf{O}(k)$ is a $n_o \times 1$ vector containing the outputs of the network. n_o is 2 throughout this paper. The first output is trained to predict the probability of a positive change, the second output is trained to predict the probability of a negative change.

[4] Elman refers to the topology of the network – full backpropagation through time was used as opposed to the truncated version used by Elman.

Thus, for the complete system:

$$\mathbf{O}(k+1) = F_1(\delta(k), \delta(k-1), \delta(k-2), \delta(k-3), \delta(k-4)) \tag{16}$$

which can be considered in terms of the original series:

$$\mathbf{O}(k+1) = F_2(y(k), y(k-1), y(k-2), y(k-3), y(k-4), y(k-5)) \tag{17}$$

The following sections describe the self-organizing map, recurrent network grammatical inference, dealing with non-stationarity, controlling overfitting, *a priori* class probabilities, and a method for comparison with a random walk.

5.1 The Self-Organizing Map

5.1.1 Introduction

The self-organizing map, or SOM [27] is an unsupervised learning process which learns the distribution of a set of patterns without any class information. A pattern is projected from a possibly high dimensional input space $\mathcal{S}$ to a position in the map, a low dimensional display space $\mathcal{D}$. The display space $\mathcal{D}$ is often divided into a grid and each intersection of the grid is represented in the network by a neuron. The information is encoded as the location of an activated neuron. The SOM is unlike most classification or clustering techniques in that it attempts to preserve the topological ordering of the classes in the input space $\mathcal{S}$ in the resulting display space $\mathcal{D}$. In other words, for similarity as measured using a metric in the input space $\mathcal{S}$, the SOM attempts to preserve the similarity in the display space $\mathcal{D}$.

5.1.2 Algorithm

We give a brief description of the SOM algorithm, for more details see [27]. The SOM defines a mapping from an input space $\mathcal{R}^n$ onto a topologically ordered set of nodes, usually in a lower dimensional space $\mathcal{D}$. A reference vector, $m_i \equiv [\mu_{i1}, \mu_{i2}, ..., \mu_{in}]^T \in \mathcal{R}^n$, is assigned to each node in the SOM. Assume that there are M nodes. During training, each input, $x \in \mathcal{R}^n$, is compared to m_i, $i = 1, 2, \ldots, M$, obtaining the location of the closest match ($||x - m_c|| = \min_i\{||x - m_i||\}$). The input point is mapped to this location in the SOM. Nodes in the SOM are updated according to:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)] \tag{18}$$

where t is the time during learning and $h_{ci}(t)$ is the *neighborhood function*, a smoothing kernel which is maximum at m_c. Usually, $h_{ci}(t) = h(||r_c - r_i||, t)$, where r_c and r_i represent the locations of nodes in the SOM output space $\mathcal{D}$. r_c is the node with the closest weight vector to the input sample and

r_i ranges over all nodes. $h_{ci}(t)$ approaches 0 as $||r_c - r_i||$ increases and also as t approaches ∞. A widely applied neighborhood function is:

$$h_{ci} = \alpha(t) \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right) \tag{19}$$

where $\alpha(t)$ is a scalar valued learning rate and $\sigma(t)$ defines the width of the kernel. They are generally both monotonically decreasing with time [27]. The use of the neighborhood function means that nodes which are topographically close in the SOM structure are moved towards the input pattern along with the winning node. This creates a smoothing effect which leads to a global ordering of the map. Note that $\sigma(t)$ should not be reduced too far as the map will lose its topographical order if neighboring nodes are not updated along with the closest node. The SOM can be considered a non-linear projection of the probability density, $p(x)$, of the input patterns x onto a (typically) smaller output space [27].

5.1.3 Remarks

The nodes in the display space $\mathcal{D}$ encode the information contained in the input space $\mathcal{R}^n$. Since there are M nodes in $\mathcal{D}$, this implies that the input pattern vectors $x \in \mathcal{R}^n$ are transformed to a set of M symbols, while preserving their original topological ordering in $\mathcal{R}^n$. Thus, if the original input patterns are highly noisy, the quantization into the set of M symbols while preserving the original topological ordering can be understood as a form of filtering. The amount of filtering is controlled by M. If M is large, this implies there is little reduction in the noise content of the resulting symbols. On the other hand, if M is small, this implies that there is a "heavy" filtering effect, resulting in only a small number of symbols.

5.2 Recurrent Network Prediction

In the past few years several recurrent neural network architectures have emerged which have been used for grammatical inference [7, 19, 16]. The induction of relatively simple grammars has been addressed often – e.g. [46, 47, 16] on learning Tomita grammars [43] . It has been shown that a particular class of recurrent networks are Turing equivalent [40].

The Elman neural network has feedback from each of the hidden nodes to all of the hidden nodes, as shown in figure 3. The set of M symbols from the output of the SOM are linearly encoded into a single input for the Elman network (e.g. if $M = 3$, the single input is either -1, 0, or 1). The linear encoding is important and is justified by the topographical order of the symbols. If no order was defined over the symbols then a separate input should be used for each symbol, resulting in a system with more dimensions and increased difficulty due to the curse of dimensionality. In order to facilitate the training of the recurrent network, an input window of 3 was used, i.e. the last three symbols are presented to 3 input neurons of the recurrent neural network.

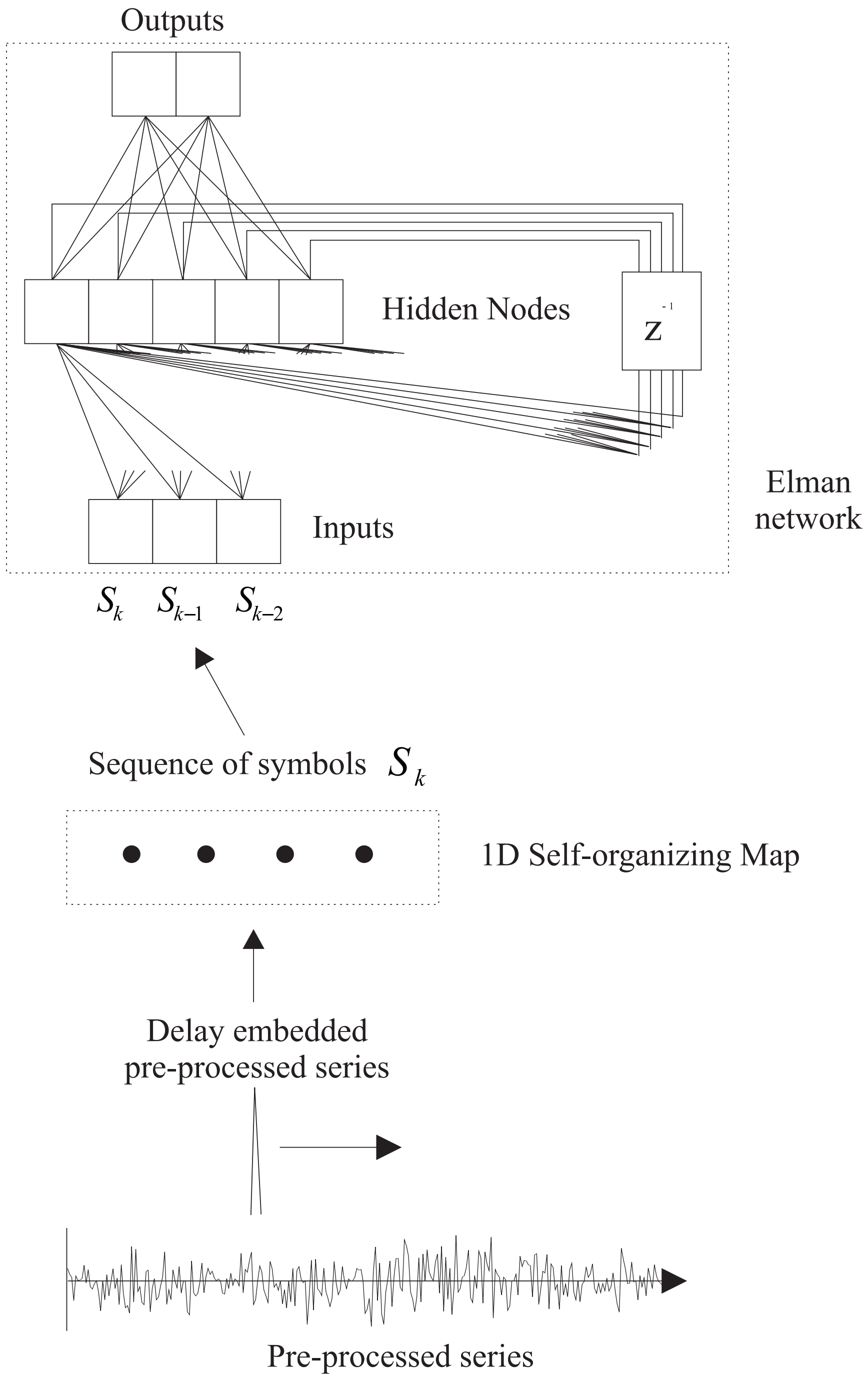

Figure 3. The pre-processed, delay embedded series is converted into symbols using a self-organizing map. An Elman neural network is trained on the sequence of symbols.

5.3 Dealing with Non-stationarity

The approach used to deal with the non-stationarity of the signal in this work is to build models based on a short time period only. This is an intuitively appealing approach because one would expect that any inefficiencies found in the market would typically not last for a long time. The difficulty with this approach is the reduction in the already small number of training data points. The size of the training set controls a noise vs. non-stationarity tradeoff [36]. If the training set is too small, the noise makes it harder to estimate the appropriate mapping. If the training set is too large, the non-stationarity of the data will mean more data with statistics that are less relevant for the task at hand will be used for creating the estimator. We ran tests on a segment of data separate from the main tests, from which we chose to use models trained on 100 days of data (see appendix A for more details). After training, each model is tested on the next 30 days of data. The entire training/test set window is moved forward 30 days and the process is repeated, as depicted in figure 4. In a real-life situation it would be desirable to test only for the next day, and advance the windows by steps of one day. However, 30 days is used here in order to reduce the amount of computation by a factor of 30, and enable a much larger number of tests to be performed (i.e. 30 times as many predictions are made from the same number of training runs), thereby increasing confidence in the results.

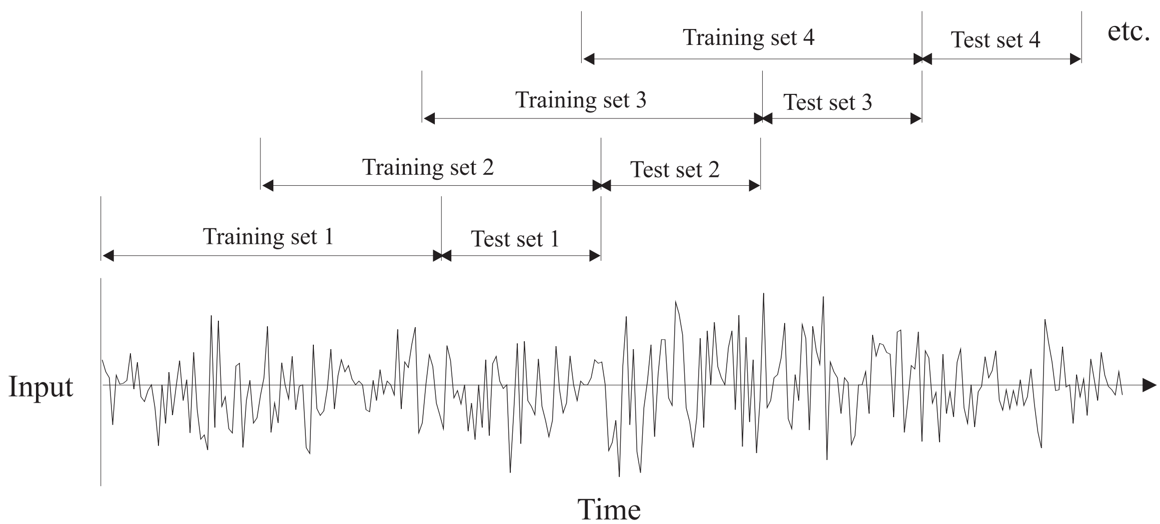

Figure 4. A depiction of the training and test sets used.

5.4 Controlling Overfitting

Highly noisy data has often been addressed using techniques such as weight decay, weight elimination and early stopping to control overfitting [48]. For this problem we use early stopping with a difference: the stopping point is chosen using multiple tests on a separate segment of data, rather than using a validation set in the normal manner. Tests showed that using a validation set in the normal manner seriously affected performance. This is not very surprising because there is a dilemma when choosing the validation set. If the validation set is chosen before the training data (in terms of the temporal order of the series), then the non-stationarity of the series means the validation set may be of little use for predicting performance on the test set. If the validation set is chosen after the training data, a problem occurs again with the non-stationarity – it is desirable to make the validation set as small as possible to alleviate this problem, however the set cannot be made too small because it needs to be a certain size in order to make the results statistically valid. Hence, we chose to control overfitting by setting the training time for an appropriate stopping point *a priori*. Simulations with a separate segment of data not contained in the main data (see appendix A for more details) were used to select the training time of 500 epochs.

We note that our use of a separate segment of data to select training set size and training time are not expected to be ideal solutions, e.g. the best choice for these parameters may change significantly over time due to the non-stationarity of the series.

5.5 *A Priori* Data Probabilities

For some financial data sets, it is possible to obtain good results by simply predicting the most common change in the training data (e.g. noticing that the change in prices is positive more often than it is negative in the training set and always predicting positive for the test set). Although predictability due to such data bias may be worthwhile, it can prevent the discovery of a better solution. Figure 5 shows a simplified depiction of the problem – if the naive solution, A, has a better mean squared error than a random solution then it may be hard to find a better solution, e.g. B. This may be particularly problematic for high noise data although the true nature of the error surface is not yet well understood.

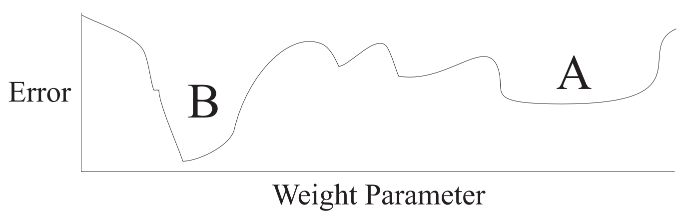

Figure 5. A depiction of a simplified error surface if a naive solution, A, performs well. A better solution, e.g. B, may be difficult to find.

Hence, in all results reported here the models have been prevented from acting on any such bias by ensuring that the number of training examples for the positive and negative cases is equal. The training set actually remains unchanged so that the temporal order is not disturbed but there is no training signal for the appropriate number of positive or negative examples starting from the beginning of the series. The number of positive and negative examples is typically approximately equal to begin with, therefore the use of this simple technique did not result in the loss of many data points.

5.6 Comparison with a Random Walk

The prediction corresponding to the random walk model is equal to the current price, i.e. no change. A percentage of times the model predicts the correct change (apart from no change) cannot be obtained for the random walk model. However, it is possible to use models with data that corresponds to a random walk and compare the results to those obtained with the real data. If equal performance is obtained on the random walk data as compared to the real data then no significant predictability has been found. These tests were performed by randomizing the classification of the test set. For each original test set five randomized versions were created. The randomized versions were created by repeatedly selecting two random positions in time and swapping the two classification values (the input values remain unchanged). An additional benefit of this technique is that if the model is showing predictability based on any data bias (as discussed in the previous section) then the results on the randomized test sets will also show this predictability, i.e. always predicting a positive change will result in the same performance on the original and the randomized test sets.

6 Experimental Results

We present results using the previously described system and alternative systems for comparison. In all cases, the Elman networks contained 5 hidden units and the SOM had one dimension. Further details of the individual simulations can be found in appendix A. Figure 6 and table 1 shows the average error rate as the number of nodes in the self-organizing map (SOM) was increased from 2 to 7. Results are shown for SOM embedding dimensions of 1 and 2. Results are shown using both the original data and the randomized data for comparison with a random walk. The results are averaged over the 5 exchange rates with 30 simulations per exchange rate and 30 test points per simulation, for a total of 4500 test points. The best performance was obtained with a SOM size of 7 nodes and an embedding dimension of 2 where the error rate was 47.1%. A t-test indicates that this result is significantly different from the null hypothesis of the randomized test sets at $p = 0.0076$ or 0.76% ($t = 2.67$) indicating that the result is significant. We also considered a null hypothesis corresponding to a model which makes completely random predictions (corresponding to a random walk) – the result is significantly different from this null hypothesis at $p = 0.0054$ or 0.54% ($t = 2.80$). Figure 7 and table 2 shows the average results for the SOM size of 7 on the individual exchange rates.

In order to gauge the effectiveness of the symbolic encoding and recurrent neural network, we investigated the following two systems:

1. The system as presented here without the symbolic encoding, i.e. the preprocessed data is entered directly into the recurrent neural network without the SOM stage.
2. The system as presented here with the recurrent network replaced by a standard MLP network.

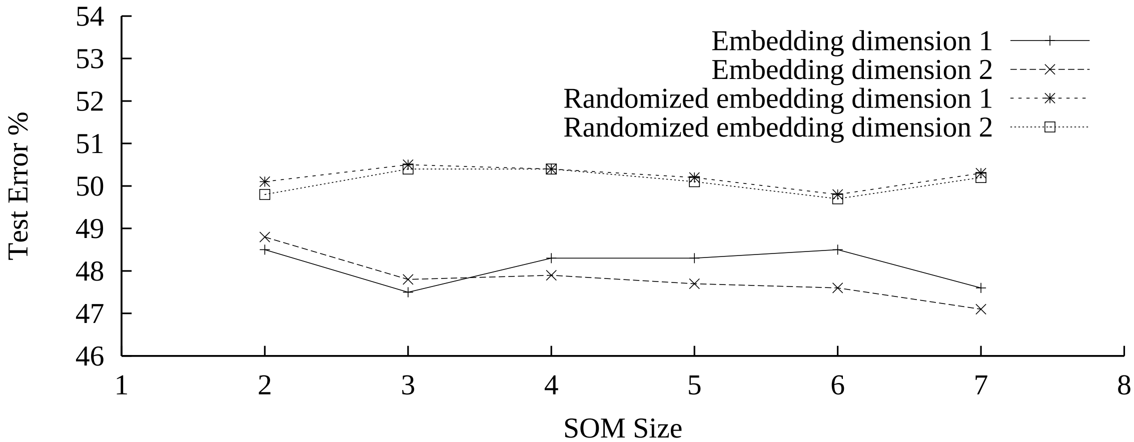

Figure 6. The error rate as the number of nodes in the SOM is increased for SOM embedding dimensions of 1 and 2. Each of the results represents the average over the 5 individual series and the 30 simulations per series, i.e. over 150 simulations which tested 30 points each, or 4500 individual classifications. The results for the randomized series are averaged over the same values with the 5 randomized series per simulation, for a total of 22500 classifications per result (all of the data available was not used due to limited computational resources).

	SOM embedding dimension: 1					
SOM size	2	3	4	5	6	7
Error	48.5	47.5	48.3	48.3	48.5	47.6
Randomized error	50.1	50.5	50.4	50.2	49.8	50.3

	SOM embedding dimension: 2					
SOM size	2	3	4	5	6	7
Error	48.8	47.8	47.9	47.7	47.6	47.1
Randomized error	49.8	50.4	50.4	50.1	49.7	50.2

Table 1. The results as shown in figure 6.

Exchange Rate	British Pound	Canadian Dollar	German Mark	Japanese Yen	Swiss Frank
Embedding dimension 1	48.5	46.9	46	47.7	48.9
Embedding dimension 2	48.1	46.8	46.9	46.3	47.4
Randomized embedding dimension 1	50.1	50.5	50.4	49.8	50.7
Randomized embedding dimension 2	50.2	50.6	50.5	49.7	49.8

Table 2. Results as shown in figure 7.

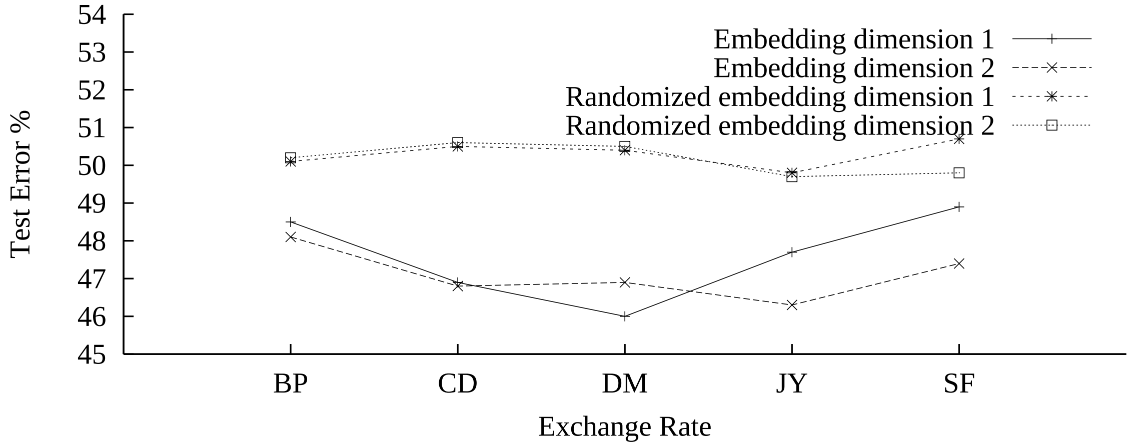

Figure 7. Average results for the five exchange rates using the real test data and the randomized test data. The SOM size is 7 and the results for SOM embedding dimensions of 1 and 2 are shown.

Tables 3 and 4 show the results for systems 1 and 2 above. The performance of these systems can be seen to be in-between the performance of the null hypothesis of a random walk and the performance of the hybrid system, indicating that the hybrid system can result in significant improvement.

Error	49.5
Randomized error	50.1

Table 3. The results for the system without the symbolic encoding, i.e. the preprocessed data is entered directly into the recurrent neural network without the SOM stage.

	SOM embedding dimension: 1					
SOM size	2	3	4	5	6	7
Error	49.1	49.4	49.3	49.7	49.9	49.8
Randomized error	49.7	50.1	49.9	50.3	50.0	49.9

Table 4. The results for the system using an MLP instead of the RNN.

6.1 Reject Performance

We used the following equation in order to calculate a confidence measure for every prediction: $\gamma = y_{max}(y_{max} - y_{min})$ where y_{max} is the maximum output, and y_{min} is the minimum output (for outputs which have been transformed using the *softmax* transformation: $y_i = \frac{\exp(u_i)}{\sum_{j=1}^{k} \exp(u_j)}$ where u_i are the original outputs, y_i are the transformed outputs, and k is the number of outputs). The confidence measure γ gives an indication of how confident the network model is for each classification. Thus, for a fixed value of γ, the prediction obtained by a particular algorithm can be divided into two sets, one set $\mathcal{B}$ for those below the threshold, and another set $\mathcal{A}$ for those above the threshold. We can then reject those points which are in set $\mathcal{B}$, i.e. they are not used in the computation of the classification accuracy.

Figure 8 shows the classification performance of the system (percentage of correct sign predictions) as the confidence threshold is increased and more examples are rejected for classification. Figure 9 shows the same graph for the randomized test sets. It can be seen that a significant increase in performance (down to approximately 40% error) can be obtained by only considering the 5-15% of examples with the highest confidence. This benefit can not be seen for the randomized test sets. It should be noted that by the time 95% of test points are rejected, the total number of remaining test points is only 225.

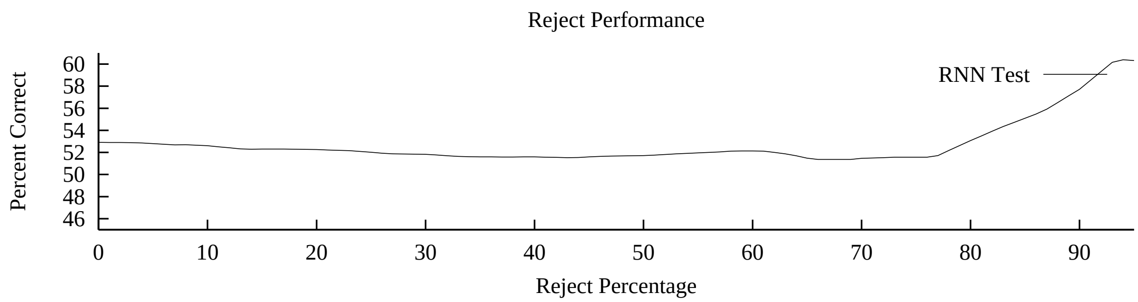

Figure 8. Classification performance as the confidence threshold is increased. This graph is the average of the 150 individual results.

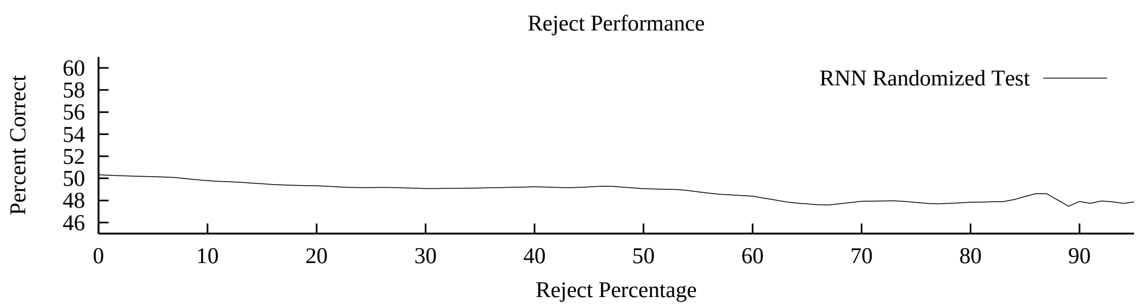

Figure 9. Classification performance as the confidence threshold is increased for the randomized test sets. This graph is the average of the individual randomized test sets.

7 Automata Extraction

A common complaint with neural networks is that the models are difficult to interpret – i.e. it is not clear how the models arrive at the prediction or classification of a given input pattern [39]. A number of people have considered the extraction of symbolic knowledge from trained neural networks for both feedforward and recurrent neural networks [9, 18, 37]. For recurrent networks, the ordered triple of a discrete Markov process ($\{$state; input $\rightarrow$ next state$\}$) can be extracted and used to form an equivalent deterministic finite state automata (DFA). This can be done by clustering the activation values of the recurrent state neurons [37]. The automata extracted with this process can only recognize regular grammars[5].

The algorithm used for automata extraction is the same as that described in [16]. The algorithm is based on the observation that the activations of the recurrent state neurons in a trained network tend to cluster. The output of each of the N state neurons is divided into q intervals of equal size, yielding q^N partitions in the space of outputs of the state neurons. Starting from an initial state, the input symbols are considered in order. If an input symbol causes a transition to a new partition in the activation space then a new DFA state is created with a corresponding transition on the appropriate symbol. In the event that different activation state vectors belong to the same partition, the state vector which first reached the partition is chosen as the new initial state for subsequent symbols. The extraction would be computationally infeasible if all q^N partitions had to be visited, but the clusters corresponding to DFA states often cover a small percentage of the input space.

Sample deterministic finite state automata (DFA) extracted from trained financial prediction networks using a quantization level of 5 can be seen in figure 10. The DFAs have been minimized using standard minimization techniques [23]. The DFAs were extracted from networks where the SOM embedding dimension was 1 and the SOM size was 2. As expected, the DFAs extracted in this case are relatively simple. In all cases, the SOM symbols end up representing positive and negative changes in the series – transitions marked with a solid line correspond to positive changes and transitions marked with a dotted line correspond to negative changes. For all DFAs, state 1 is the starting state. Double circled states correspond to prediction of a positive change, and states without the double circle correspond to prediction of a negative change. (a) can be seen as corresponding to mean-reverting behavior – i.e. if the last change in the series was negative then the DFA predicts that the next change will be positive and vice versa. (b) and (c) can be seen as variations on (a) where there are exceptions to the simple rules of (a), e.g. in (b) a negative change corresponds to a positive prediction, however after the input changes from negative to positive, the prediction alternates with successive positive changes. In contrast with (a)-(c), (d)-(f) exhibit behavior related to trend following algorithms, e.g. in (d) a positive change corresponds to a positive prediction, and a negative change corresponds to a negative prediction except for the first negative change after a positive change. Figure 11 shows a sample DFA for a SOM size of 3 – more complex behavior can be seen in this case.

[5] A regular grammar G is a 4-tuple $G = \{S, N, T, P\}$ where S is the start symbol, N and T are non-terminal and terminal symbols, respectively, and P represents productions of the form $A \rightarrow a$ or $A \rightarrow aB$ where $A, B \in N$ and $a \in T$.

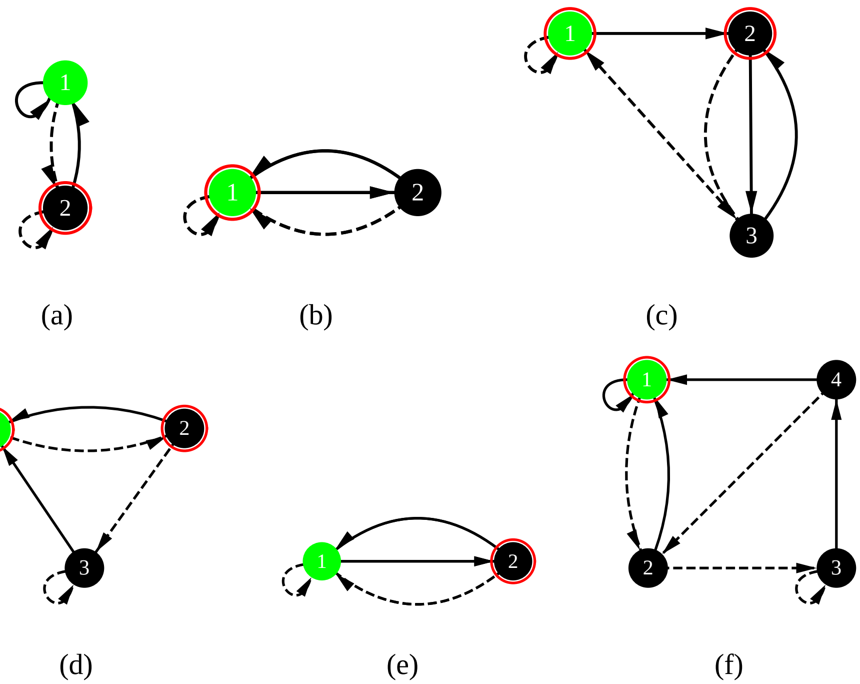

Figure 10. Sample deterministic finite state automata (DFA) extracted from trained financial prediction networks using a quantization level of 5.

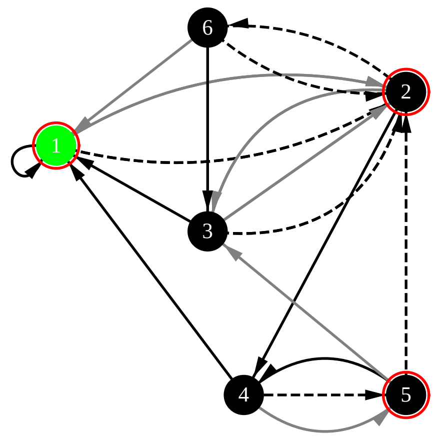

Figure 11. A sample DFA extracted from a network where the SOM size was 3 (and the SOM embedding dimension was 1). The SOM symbols represent a positive (solid lines) or negative (dotted lines) change, or a change close to zero (gray line). In this case, the rules are more complex than the simple rules in the previous figure.

8 Conclusions

Traditional statistical methods have difficulty with high noise, high non-stationarity time series prediction. The intelligent signal processing method presented here uses a combination of symbolic processing and recurrent neural networks to deal with fundamental difficulties. The use of a recurrent neural network is significant for two reasons: firstly, the temporal relationship of the series is explicitly modeled via internal states, and secondly, it is possible to extract rules from the trained recurrent networks in the form of deterministic finite state automata. The symbolic conversion is significant for a number of reasons: the quantization effectively filters the data or reduces the noise, the RNN training becomes more effective, and the symbolic input facilitates the extraction of rules from the trained networks. Considering foreign exchange rate prediction, we performed a comprehensive set of simulations covering 5 different foreign exchange rates, which showed that significant predictability does exist with a 47.1% error rate in the prediction of the direction of the change. Additionally, the error rate reduced to around 40% when rejecting examples where the system had low confidence in its prediction. The method facilitates the extraction of rules which explain the operation of the system. Results for the problem considered here are positive and show that symbolic knowledge which has meaning to humans can be extracted from noisy time series data. Such information can be very useful and important to the user of the technique.

Appendix A: Simulation Details

For the experiments reported in this paper, $n_h = 5$ and $n_o = 2$. The RNN learning rate was linearly reduced over the training period (see [8] regarding learning rate schedules) from an initial value of 0.5. All inputs were normalized to zero mean and unit variance. All nodes included a bias input which was part of the optimization process. Weights were initialized as shown in Haykin [22]. Target outputs were -0.8 and 0.8 using the $\tanh$ output activation function and we used the quadratic cost function. In order to initialize the delays in the network, the RNN was run without training for 50 time steps prior to the start of the datasets. The number of training examples for the positive and negative cases was made equal by not training on the appropriate number of positive or negative examples starting from the beginning of the training set (i.e. the temporal order of the series is not disturbed but there is no training signal for either a number of positive or a number of negative examples starting at the beginning of the series). The SOM was trained for 10,000 iterations using the following learning rate schedule $0.5 \times (1 - n_i/n_{ti})$ where n_i is the current iteration number and n_{ti} is the total number of iterations (10,000). The neighborhood size was altered during training using floor$(1+((2/3)n_s-1)(1-n_i/n_{ti})+0.5)$. The first differenced data was split as follows: Points 1210 to 1959 (plus the size of the validation set in the validation set experiments) in each of the exchange rates were used for the selection of the training dataset size, the training time, and experimentation with the use of a validation set. Points 10 to 1059 in each of the exchange rates were used for the main tests. The data for each successive simulation was offset by the size of the test set, 30 points, as shown in figure 4. The number of points in the main tests from each exchange rate is equal to 1,050 which is 50 (used to initialize delays) + 100 (training set size) + 30 $\times$ 30 (30 test sets of size 30). For the selection of the training dataset size and the training time the number of points was 750 for each exchange rate (50 + 100 + 20 test sets of size 30). For the experiments using a validation set in each simulation, the number of points was 750 + the size of the validation set for each of the exchange rates.

Acknowledgments

We would like to thank Doyne Farmer, Andreas Weigend and Christian Omlin for helpful comments.

References

[1] Y.S. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6:192, 1990.

[2] J.A. Alexander and M.C. Mozer. Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 609–616. The MIT Press, 1995.

[3] Martin Anthony and Norman L. Biggs. A computational learning theory view of economic forecasting with neural nets. In A. Refenes, editor, *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995.

[4] A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

[5] R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.

[6] Y. Bengio, editor. *Neural Networks for Speech and Sequence Recognition*. Thomson, 1996.

[7] A. Cleeremans, D. Servan-Schreiber, and J.L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381, 1989.

[8] C. Darken and J.E. Moody. Note on learning rate schedules for stochastic optimization. In R.P. Lippmann, J.E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 832–838. Morgan Kaufmann, San Mateo, CA, 1991.

[9] J.S. Denker, D. Schwartz, B. Wittner, S.A. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.

[10] J.L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2/3):195–226, 1991.

[11] E.F. Fama. The behaviour of stock market prices. *Journal of Business*, January:34–105, 1965.

[12] E.F. Fama. Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, May:383–417, 1970.

[13] András Faragó and Gábor Lugosi. Strong universal consistency of neural network classifiers. *IEEE Transactions on Information Theory*, 39(4):1146–1151, 1993.

[14] J.H. Friedman. An overview of predictive learning and function approximation. In V. Cherkassky, J.H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks, Theory and Pattern Recognition Applications*, volume 136 of *NATO ASI Series F*, pages 1–61. Springer, 1994.

[15] J.H. Friedman. Introduction to computational learning and statistical prediction. Tutorial Presented at Neural Information Processing Systems, Denver, CO, 1995.

[16] C. Lee Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[17] C. Lee Giles, C.B. Miller, D. Chen, G.Z. Sun, H.H. Chen, and Y.C. Lee. Extracting and learning an unknown grammar with recurrent neural networks. In J.E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 317–324, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[18] C. Lee Giles and C.W. Omlin. Extraction, insertion and refinement of symbolic rules in dynamically-driven recurrent neural networks. *Connection Science*, 5(3,4):307–337, 1993. Special Issue on Architectures for Integrating Symbolic and Neural Processes.

[19] C. Lee Giles, G.Z. Sun, H.H. Chen, Y.C. Lee, and D. Chen. Higher order recurrent networks & grammatical inference. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 380–387, San Mateo, CA, 1990. Morgan Kaufmann Publishers.

[20] C.W.J. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, San Diego, second edition, 1986.

[21] Yoichi Hayashi. A neural expert system with automated extraction of fuzzy if-then rules. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 578–584. Morgan Kaufmann Publishers, 1991.

[22] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Macmillan, New York, NY, 1994.

[23] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, Reading, MA, 1979.

[24] L. Ingber. Statistical mechanics of nonlinear nonequilibrium financial markets: Applications to optimized trading. *Mathematical Computer Modelling*, page in press, 1996.

[25] M. Ishikawa. Rule extraction by successive regularization. In *International Conference on Neural Networks*, pages 1139–1143. IEEE Press, 1996.

[26] M.I. Jordan. Neural networks. In A. Tucker, editor, *CRC Handbook of Computer Science*, page in press. CRC Press, Boca Raton, FL, 1996.

[27] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Germany, 1995.

[28] A.N. Kolmogorov. On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition. *Dokl*, 114:679–681, 1957.

[29] V. Kůrková. Kolmogorov's theorem is relevant. *Neural Computation*, 3(4):617–622, 1991.

[30] V. Kůrková. Kolmogorov's theorem. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 501–502. MIT Press, Cambridge, Massachusetts, 1995.

[31] Steve Lawrence, C. Lee Giles, and Sandiway Fong. Can recurrent neural networks learn natural language grammars? In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1853–1858. IEEE Press, Piscataway, NJ, 1996.

[32] Jean Y. Lequarré. Foreign currency dealing: A brief introduction (data set C). In A.S. Weigend and N.A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, 1993.

[33] B.G. Malkiel. *Efficient Market Hypothesis*. Macmillan, London, 1987.

[34] B.G. Malkiel. *A Random Walk Down Wall Street*. Norton, New York, NY, 1996.

[35] C. McMillan, M.C. Mozer, and P. Smolensky. Rule induction through integrated symbolic and subsymbolic processing. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 969–976. Morgan Kaufmann Publishers, 1992.

[36] J.E. Moody. Economic forecasting: Challenges and neural network solutions. In *Proceedings of the International Symposium on Artificial Neural Networks*. Hsinchu, Taiwan, 1995.

[37] C.W. Omlin and C.L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

[38] D.W. Ruck, S.K. Rogers, K. Kabrisky, M.E. Oxley, and B.W. Suter. The multilayer perceptron as an approximation to an optimal Bayes estimator. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[39] R. Setiono and H. Liu. Symbolic representation of neural networks. *Computer*, 29(3):71–77, 1996.

[40] H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

[41] 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 366–381, Berlin, 1981. Springer-Verlag.

[42] S.J. Taylor, editor. *Modelling Financial Time Series*. J. Wiley & Sons, Chichester, 1994.

[43] M. Tomita. Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 105–108, Ann Arbor, MI, 1982.

[44] G. Towell and J. Shavlik. The extraction of refined rules from knowledge based neural networks. *Machine Learning*, 131:71–101, 1993.

[45] George Tsibouris and Matthew Zeidenberg. Testing the efficient markets hypothesis with gradient descent algorithms. In A. Refenes, editor, *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995.

[46] R.L. Watrous and G.M. Kuhn. Induction of finite state languages using second-order recurrent networks. In J.E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 309–316, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[47] R.L. Watrous and G.M. Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406, 1992.

[48] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart. Predicting sunspots and exchange rates with connectionist networks. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity, Proceedings Vol. XII*, pages 395–432. Addison-Wesley, 1992.

[49] G.U. Yule. On a method of investigating periodicities in disturbed series with special reference to Wolfer's sunspot numbers. *Philosophical Transactions Royal Society London Series A*, 226:267–298, 1927.

[50] Z. Zeng, R.M. Goodman, and P. Smyth. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330, 1994.