Rule Inference for Financial Prediction using Recurrent Neural Networks, Proceedings of IEEE/IAFE Conference on Computational Intelligence for Financial Engineering (CIFEr), IEEE, Piscataway, NJ, 1997, pp. 253–259. Copyright IEEE.

Rule Inference for Financial Prediction using Recurrent Neural Networks

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

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

Abstract

This paper considers the prediction of noisy time series data, specifically, the prediction of foreign exchange rate data. A novel hybrid neural network algorithm for noisy time series prediction is presented which exhibits excellent performance on the problem. The method is motivated by consideration of how neural networks work, and by fundamental difficulties with random correlations when dealing with small sample sizes and high noise data. The method permits the inference and extraction of rules. One of the greatest complaints against neural networks is that it is hard to figure out exactly what they are doing – this work provides one answer for the internal workings of the network. Furthermore, these rules can be used to gain insight into both the real world system and the predictor. This paper focuses on noisy time series prediction and rule inference – use of the system in trading would typically involve the utilization of other financial indicators and domain knowledge.

1 Introduction

1.1 Prediction System

Neural networks are considered by many to provide state-of-the-art solutions to noisy time series prediction problems such as financial prediction [10]. When using neural networks to predict noisy time series data, a delay embedding [3] of previous temporal inputs is typically mapped into a prediction. Neural network models can be grouped into two classes based on whether or not they address the temporal relationship of the inputs by maintaining an internal state. For models like the standard multi-layer perceptron (MLP) neural network, the traditional vector space input encoding gives the model no hint as to the temporal relationship of the inputs. A short temporal pattern can be difficult to distinguish from patterns due to random correlations although these random patterns may not occur in temporal order. Recurrent neural network (RNN) models employ feedback connections and have the potential to represent certain computational structures in a more parsimonious fashion [2]. Such models are suited to the detection of temporal patterns. However, for noisy time series prediction, training can be difficult, and random correlations with the most recent data can make predictability based on earlier data difficult.

In the method used here, time series data is first converted 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 abilities of recurrent neural networks to learn deterministic grammatical rules which capture predictability in the evolution of the series. The symbolic encoding makes training the recurrent network easier, and aids in the extraction of symbolic knowledge. The use of a recurrent neural network is significant for two reasons: first, the model addresses the temporal relationship of the series by maintaining an internal state, and second, it is possible to extract more general rules from the trained recurrent networks which may be interpreted by humans. These rules are in the form of deterministic finite state automata. The use of an RNN can be seen as a bias against finding correlations which do not occur in temporal order.

---

[*] 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/lawrence`

1.2 Foreign Exchange

Foreign exchange rates exhibit very high noise, and significant non-stationarity. Here, we consider the prediction of the direction of change in the exchange rates for the next business day. A number of people have applied neural network technology to exchange rate prediction and trading, e.g. [13, 11]. This work focuses on our approach to noisy time series prediction and rule inference (we do not consider, for example, the use of external variables such as interest rates, or the use of a trading policy). The data used here is the same as the data used by Weigend et al. [13], and is from the Monetary Yearbook of the Chicago Mercantile Exchange and 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. The data covers the period September 3, 1973 to May 18, 1987 (there are 3645 data points), and is available from `http://www.cs.colorado.edu/~andreas/Time-Series/Data/Exchange.Rates.Daily`.

1.3 Efficient Market Hypothesis

The evolution of exchange rates has traditionally been thought to agree with the efficient market hypothesis (EMH), a theory which has found broad acceptance in the academic financial community [8]. 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. One argument for the EMH is that if any profit opportunities appear, they will be exploited immediately, and thereby disappear. Arguments against the EMH include the facts that not all information reaches all traders simultaneously, that there are significant differences in objectives and trading time frames, and that people with more sophisticated algorithms may be able to make more use of the available information.

2 Prediction System

2.1 Overview

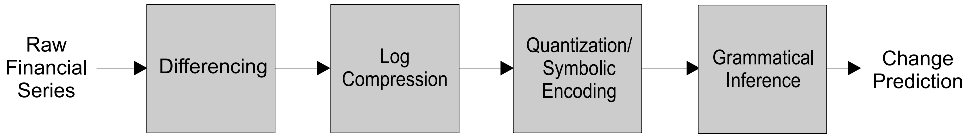


Figure 1. A high-level block diagram of the system we have used.

A high-level block diagram of the system we have used for noisy time series prediction is shown in figure 1. Training a model on a raw price series is problematic – e.g. the price range covered by the test data may be very different to the price range covered by the training data. As is common, we pass the raw series through first-order differencing and log compression stages. The differencing removes the long term trend and the log compression reduces the effect of outliers (often a result of exogenous shocks). We then use a delay embedding and quantization of the series. The quantization results in the series being converted into a sequence of symbols. A recurrent neural network is then used on the grammatical inference problem of mapping the symbol sequence into a prediction. The following section provides a mathematical description of the system and subsequent sections provide more information on the self-organizing map, the grammatical inference, and the techniques used to deal with non-stationarity, overfitting, and data bias.

2.2 Details

The raw time series values are $y(k)$, $k = 1, 2, \ldots, N$ where $y(k) \in \mathcal{R}$. These denote the daily closing prices of the financial time series. The first difference of the series, $y(k)$, is taken as follows:

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

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:

$$x(k) = \text{sign}(\delta(k))(\log(|\delta(k)| + 1)) \tag{2}$$

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 [7]:

$$\mathbf{X}(k, d_1) = (x(k), x(k-1), x(k-2), \ldots, x(k - d_1 + 1)) \tag{3}$$

where d_1 is the delay embedding dimension and is 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:

$$S(k) = g(\mathbf{X}(k, d_1)) \tag{4}$$

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 which is trained on the sequence of outputs from the SOM. For the Elman network:

$$\mathbf{O}(k+1) = \mathbf{C}^T \mathbf{z}_k + c_0 \tag{5}$$

and

$$\mathbf{z}_k = F_{n_h}(\mathbf{A}\mathbf{z}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{b}) \tag{6}$$

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:

$$\mathbf{u}_k = \begin{bmatrix} S(k) \\ S(k-1) \\ S(k-2) \\ \ldots \\ S(k - d_2 + 1) \end{bmatrix} \tag{7}$$

$\mathbf{A}$ 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, and the second output is trained to predict the probability of a negative change. Thus, for the complete system:

$$\mathbf{O}(k+1) = F_1(\delta(k), \delta(k-1), \delta(k-2), \ldots, \delta(k - d_1 + 1)) \tag{8}$$

which can be considered in terms of the original series:

$$\mathbf{O}(k+1) = F_2(y(k), y(k-1), y(k-2), \ldots y(k - d_1)) \tag{9}$$

2.3 Self-organizing Map

The self-organizing map, or SOM 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 $\mathcal{D}$. For more information on the SOM, see [5].

2.4 Grammatical Inference

Several recurrent neural network architectures have been used for grammatical inference [6]. It has been shown that a particular class of recurrent networks are at least Turing equivalent [12]. The recurrent neural network we have used for grammatical inference here is the Elman neural network. The Elman network has fully connected feedback among the hidden layer neurons. For the examples here, the networks had three input neurons, one hidden layer with five neurons, and two output neurons - one each for prediction of the probability of positive and negative changes. The set of symbols from the output of the SOM are linearly encoded into a single input for the Elman network. The linear encoding is justified by the topographical order of the symbols. Use of this encoding is important – if there was no order defined for the symbols then the linear encoding would not be appropriate and separate inputs should be used for each symbol. This is problematic due to the increased dimensionality of the resulting network (cf. the curse of dimensionality). Delayed inputs are used for the other two input neurons, aiding in the training process. The networks were trained with backpropagation through time [4] for a total of 500 updates. Weights in the network were updated after all patterns were presented (batch update), as opposed to stochastic update where weights are updated after every pattern presentation (stochastic update typically performs poorly when the data is very noisy). 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 [4]. Target outputs were -0.8 and 0.8 using the $\tanh$ output activation function and we used the quadratic cost function. A search then converge learning rate schedule was used [1] with an initial learning rate was 0.5.

2.5 Non-stationarity

The approach we use to handle non-stationarity is to build models based on short time periods only. There is a noise vs. non-stationarity tradeoff as the size of the training set is varied. If the training set is too small, noise makes it harder to find true patterns in the data. If the training set is too large, the non-stationarity of the data means that more data with statistics that are less relevant for the task at hand is used when creating the estimator. Based on preliminary experiments (using a separate segment of data that was not part of the subsequent training and test data sets), we chose to train models using 100 days of data in order to balance the noise/non-stationarity tradeoff. After training, we test each model on the next 30 days of data. The entire training/test set window is moved forward 30 days and the process is repeated. We use test sets of 30 days instead of one day (i.e. a new model every day) in order to reduce the amount of computation by a factor of 30, and enable us to perform a much larger number of tests, thereby increasing the significance of our results.

2.6 Overfitting

Tests showed that using a validation set for this problem seriously affected performance. This is not surprising because there is a dilemma when choosing the validation set. If the validation set is located 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 located after the training data, we again have a problem with the non-stationarity - we would like to make the validation set as small as possible to alleviate this problem, however it cannot be made too small because it needs to be a certain size in order for the results to be statistically valid and therefore useful for controlling the bias/variance tradeoff. For the problem considered, a validation set would have to be large because of the high degree of noise. With this in mind, we control overfitting by setting the number of hidden nodes and training time for an appropriate stopping point *a priori*. This stopping point was determined from experiments with a separate segment of data that was not part of the subsequent training and test data sets.

2.7 Data Bias

In some cases, better than average results can be obtained 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 is uninteresting in the present context because it can obscure any results due to the model learning short term predictability of the dynamic evolution of the series. Such bias can also present problems to model training algorithms because the lower mean squared error of the simple solution (predict always positive or always negative) can prevent gradient descent algorithms from finding any better minima. In our system we have prevented the models 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 a number of positive or negative examples near the beginning of the series).

2.8 Random Walk Comparison

The prediction corresponding to the random walk model is equal to the current price, i.e. no change. We cannot obtain a percentage of times the model predicts the correct change for the random walk model because the model predicts no change. However, we can use our models with data that corresponds to a random walk and compare the results to those obtained with the real data. If we obtain equal performance on the random walk data as compared to the real data, then we have not found any predictability. We performed these tests by randomizing the classifications in the test set. For each original test set we created five randomized versions. 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 (the original and randomized test sets contain identical numbers of positive and negative examples).

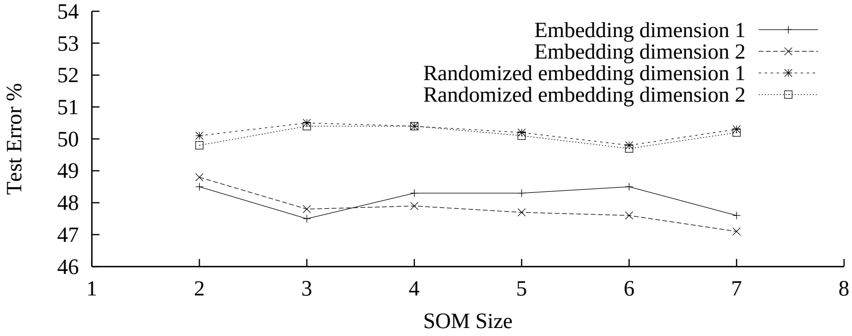
3 Results

Figure 2 shows the average error rate as the number of nodes in the SOM was increased from 2 to 7 for SOM embedding dimensions of 1 and 2. Each of the results represents the average over the 5 individual series and 30 simulations per series, i.e. over 150 simulations which tested 30 points each, or a total of 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. 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%. Experiments were also performed where the RNN was replaced with a standard multi-layer perceptron (MLP) neural network - performance in this case was very poor.

A confidence measure was calculated for every prediction based on the magnitude of the winning output and the difference between the winning output and the other output (for outputs transformed using the *softmax* transformation [4]). We assigned a confidence threshold which was used to reject predictions with confidence values below the threshold. We found that the classification error could be reduced to around 40% by rejecting all but the 10-20% of examples with the highest confidence. The randomized test sets did not show this benefit.

4 Rule Extraction

Understanding of the operation of a neural network can be gained by the extraction of rules. The ordered triple of a discrete Markov process ($\{$state; input $\rightarrow$ next state$\}$) can be extracted from an RNN 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 [9]. A simple algorithm can then be used to assign states to the clusters and insert transitions between the clusters on the relevant input symbols. Figure 3 shows sample automata extracted from the trained networks. The automata 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 of these 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. State 1 is the starting state of the DFAs. 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) can be seen as a variations on (a) where there are exceptions to the simple rules of (a) – 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) and (b), (c) and (d) exhibit behavior related to trend following algorithms, e.g. in



	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

Figure 2. 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).

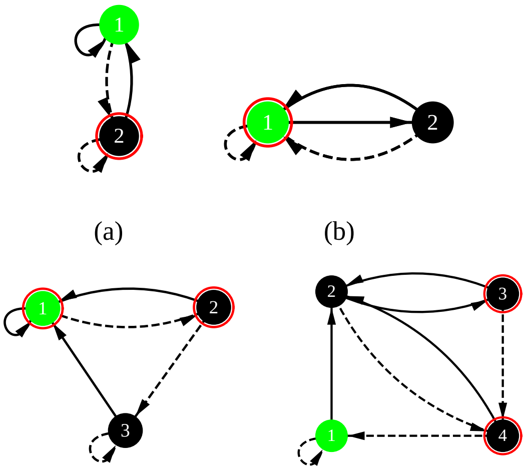
(c) a positive change corresponds to a positive prediction, and a negative change corresponds to a negative prediction except for the first negative change after a positive change. These automata correspond to the simple case where the SOM size is 2 and the SOM embedding dimension is 1. More complex and better performing automata can be extracted as the SOM size and embedding dimension are increased.

5 Summary

The hybrid neural network approach presented here addresses fundamental problems with the prediction of a noisy time series. The method has found significant predictability in foreign exchange rate data. With an extensive set of 4500 predictions covering 5 different foreign exchange rates, the direction of change for the next day has been predicted with a 47.1% error rate. Furthermore, the method permits rules related to well known behavior such as trend following and mean-reversal to be extracted from the trained predictors. These results for the extraction of rules are encouraging and show that meaningful symbolic data can be extracted from a noisy time series. Such knowledge may be very useful to a user of this methodology.

Acknowledgments

We would like to thank D. Farmer, A. Weigend and C. Omlin for helpful comments.



(a) (b)

(c) (d)

Figure 3. Sample deterministic finite state automata (DFA) extracted from trained financial prediction networks using a quantization level of 5.

References

[1] C. Darken and J.E. Moody. Towards faster stochastic gradient search. In *Neural Information Processing Systems 4*, pages 1009–1016. Morgan Kaufmann, 1992.

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

[3] D. Farmer and J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59:845–848, 1987.

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

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

[6] Stefan C. Kremer. *A Theory of Grammatical Induction in the Connectionist Paradigm*. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, 1996.

[7] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.

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

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

[10] A. Refenes, editor. *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995.

[11] A. Refenes and A. Zaidi. Managing exchange-rate prediction strategies with neural networks. In A. Refenes, editor, *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995.

[12] H.T. Siegelmann. Computation beyond the Turing limit. *Science*, 268:545–548, 1995.

[13] 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.