NEURAL NETWORK LEARNING FOR TIME-SERIES PREDICTIONS USING CONSTRAINED FORMULATIONS

BY

MINGLUN QIAN

M.S. in Physics, University of Science and Technology of China, 1995
M.S. in Computer Science, University of Illinois at Urbana-Champaign, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

In this thesis, we propose a recurrent FIR neural network, develop a *constrained formulation* for neural network learning, study an efficient *violation guided backpropagation algorithm* for solving the constrained formulation based on the *theory of extended saddle points*, and apply neural network learning for predicting both noise-free time series and noisy time series. The recurrent FIR neural-network architecture combines a recurrent structure and a memory-based FIR structure in order to provide a more powerful modeling ability. The constrained formulation for neural network learning incorporates the error of each learning pattern as a constraint, a new cross-validation scheme that allows multiple validations sets to be considered in learning, and new constraints that can be expressed in a procedure form. The violation-guided back propagation algorithm first transforms the constrained formulation into an l_1-penalty function, and searches for a saddle point of the penalty function.

When using a constrained formulation along with violation guided backpropagation to neural network learning for near noiseless time-series benchmarks, we achieve much improved prediction performance as compared to that of previous work, while using less parameters. For noisy time-series, such as financial time series, we have studied systematically trade-offs between denoising and information preservation, and have proposed three preprocessing techniques for time-series with high-frequency noise. In particular, we have proposed a novel approach by first decomposing a noisy time series into different frequency channels and by preprocessing each channel adaptively according to its level of noise. We incorporate constraints on predicting low-pass data in the lag period when a low-pass filter is employed to denoise the band. The new constraints enable active training in the lag period that greatly improves the prediction accuracy in the lag period. Extensive prediction experiments on financial time series have been conducted to exploit the modeling ability of neural networks, and promising results have been obtained.

To my son Andrew, my wife, and my parents

Acknowledgments

This thesis would not have been possible without the generous help and consistent support of my thesis advisor, Prof. Benjamin Wah. I would like to express my appreciation for his valuable insights, encouragement, and thoughtful discussions. Prof. Wah not only helped me on the research project in the fields of neural networks and global search, but also taught me how to express and communicate in research. I sincerely believe I have benefit tremendously from his help and impact on me during my years under his supervision.

I would like to thank Prof. Narendra Ahuja, Prof. Jiawei Han, and Prof. Sylvian Ray, for taking their valuable time to be on my thesis committee and for providing many helpful comments and suggestions.

I would also like to thank Tao Wang, Zhe Wu, Xiao Su, Dong Lin, Yixin Chen, Hang Yu, and all other group members in our research group for providing constructive comments on my work and for providing a congenial environment for me to work in.

I am also indebted to my wife and my parents for their everlasting love, support, and understanding. They have always been my source of strength at times of difficulties.

Table of Contents

Chapter

1 Introduction . . . 1
1.1 Motivations of this research . . . 1
1.2 Problems addressed . . . 3
1.2.1 Chaotic time series . . . 6
1.2.2 Characteristics of time series . . . 8
1.2.3 Assumptions on time series studied in this thesis . . . 12
1.3 Metrics . . . 14
1.3.1 Metrics for properties . . . 14
1.3.2 General performance metrics . . . 17
1.3.3 Performance metrics for financial time series . . . 19
1.4 Proposed approaches of time-series predictions . . . 25
1.4.1 Preprocessing of financial time series . . . 25
1.4.2 Learning and prediction . . . 26
1.5 Benchmarks studied . . . 27
1.5.1 Noiseless or near noiseless time-series benchmarks . . . 27
1.5.2 Noisy time series: stock-market time series . . . 27
1.6 Contributions of this thesis . . . 29
1.7 Outline of this Thesis . . . 31

2 Previous Work . . . 33
2.1 Previous work on predicting chaotic time-series . . . 33

2.1.1 Linear models . . . 34
2.1.2 Models for nonlinear stationary time series . . . 40
2.2 Previous work on seasonal chaotic time series . . . 45
2.2.1 Seasonality detection . . . 45
2.2.2 Seasonality Elimination . . . 46
2.2.3 Prediction . . . 47
2.3 Previous work on piecewise chaotic time series . . . 47
2.3.1 Regime transition detection . . . 47
2.3.2 Models for piecewise chaotic time series . . . 48
2.4 Previous work on handling nonstationarity . . . 51
2.5 Previous work on handling noisy time series . . . 52
2.5.1 High frequency noise detection . . . 53
2.5.2 Preprocessing: denoising . . . 54
2.6 Artificial Neural Networks . . . 57
2.6.1 ANN architectures for time-series modeling . . . 58
2.6.2 Learning of traditional ANNs . . . 61
2.6.3 Remarks on existing work on ANNs . . . 64
2.7 Summary . . . 65

3 Proposed Preprocessing Approaches . . . 66
3.1 The need for preprocessing . . . 67
3.2 Traditional low-pass filtering . . . 74
3.2.1 Filter selection . . . 75
3.3 Wavelet decomposition . . . 93
3.3.1 Redundant wavelet decomposition . . . 93
3.3.2 Transformations on low-frequency channels . . . 100
3.3.3 Denoising on high-frequency channels . . . 103
3.3.4 Parameter selection . . . 114
3.4 Summary . . . 123

4 Neural Network Models And Learning Algorithms . . . 131
4.1 Overview . . . 132
4.2 Recurrent FIR Neural Networks . . . 134
4.2.1 Architecture . . . 135
4.2.2 Generalized epochwise backpropagation through time . . . 137
4.3 Constrained formulation for ANN learning . . . 140
4.3.1 Constrained formulation without cross-validation . . . 141
4.3.2 Cross-validation as additional constraints . . . 143
4.3.3 Constraints on predictions in the lag period in noisy time series . . . 146
4.4 Learning algorithm for constrained formulations . . . 148
4.4.1 Theory of Extended Saddle Points for Continuous Nonlinear Programming . . . 148
4.4.2 Violation-Guided Back-Propagation . . . 151
4.4.3 Relax-and-tighten (R&T) strategy . . . 155
4.4.4 Violation-guided backpropagation algorithm . . . 157
4.4.5 Parameters in VGBP . . . 159
4.5 Summary . . . 163

5 Experimental Results On Near Noiseless Chaotic Time Series . . . 164
5.1 Real world time-series . . . 164
5.1.1 Sunspots time series . . . 165
5.1.2 Laser intensity time series . . . 170
5.2 Artificial chaotic time series . . . 173
5.2.1 Mackey-Glass . . . 173
5.2.2 Henon map . . . 179
5.2.3 Lorenz attractor . . . 181
5.2.4 Ikeda attractor . . . 183
5.2.5 Summary of chaotic time-series benchmarks . . . 186

6 Predicting Stock-Market Time Series . . . 188
6.1 Experiments setup . . . 189
6.1.1 Predictors compared . . . 189
6.1.2 Performance measures . . . 192
6.2 Experimental results on preprocessing . . . 193
6.2.1 ANN Prediction on individual channels using channel-specific preprocessing . . . 194
6.2.2 Postprocessing . . . 204
6.2.3 Parameters used in channel-specific preprocessing . . . 205
6.3 Experimental results on stock market predictions . . . 206
6.4 Summary and Conclusions . . . 254

7 Conclusions And Future Work . . . 257

Bibliography . . . 259

Vita . . . 276

List of Tables

1.1 Data series for different starting points according to (1.1). 7

1.2 Properties of noiseless or near noiseless nonlinear time series studied in this thesis. 28

1.3 Stocks studied in this thesis. 28

3.1 Coefficients for the filters studied. All the filters are symmetric, and only the first $1 + \frac{T}{2}$ coefficients are listed for a T-tap filter. 80

3.2 Performance for low-pass filtered stock-price time series using different filters. ACF and P indicate autocorrelation and predictability. "Hit Ratios" shows ratios w.r.t. the daily low/high price ($\alpha_0/\alpha_+/\alpha_-$) defined by (3.1) in Section 3.2.1. "Buy/Sell" shows the ratios w.r.t. potential buy/Sell signals ($\beta_0/\beta_+/\beta_-$) defined by (3.2) in Section 3.2.1. 82

3.3 Combinations of wavelet mother filter for wavelet decomposition and low-pass filters used for channel H and LH. The notation listed in column 1 will be used in Table 3.4. The second column lists the mother filter used for the wavelet decomposition. The third column tells which low-pass filter is used for Channel H; the last column shows the low-pass filter used for Channel LH. Channel LL is left unchanged. 119

3.4 Performance of preprocessed stock-prices time series using different wavelet decompositions along with channel-specific low-pass filtering. The notations for the filters are explained in Table 3.3. ACF indicates autocorrelation. The column labelled by P records the predictability measures. "Hit Ratios" shows hit/over-hit/under-hit ratios w.r.t. the daily low/high price ($\alpha_0/\alpha_+/\alpha_-$) defined by (3.1) in Section 3.2.1. "Buy/Sell" shows the ratios of zero/positive/negative errors w.r.t. potential buy/Sell signals ($\beta_0/\beta_+/\beta_-$) defined by (3.2) in Section 3.2.1. 125

5.1 Average test performance by different formulations and cross-validation methods on *sunspot*. (Each entry in the second through the fifth columns represents a mean value with 95% confidence and ±10% of the value indicated. The last column shows the number of runs needed to achieve the required level of confidence. Bold numbers indicate the best results. Each run took approximately 24 to 26 seconds on a 450-MHz Pentium-III computer under Solaris 7. The ANN used has one input, two hidden units, one output, and 11 weights.) 167

5.2 Single-step test performance in nMSE on *sunspot* for AR(12) [167], WNet [174], SSNet [117], DRNN [8], COMM [167], ScaleNet [44], and VGBP. Boxed numbers indicate the best results. N/A stands for data not available. n represents the number of weights used. The second column shows the number of weights/free variables in each method.) 169

5.3 Single-step and iterative test performance in nMSE on *laser*. (The test set consists of patterns from 1001 to 1100. As a comparison, we also computed nMSE on single-step as well as iterative predictions for patterns 1001-1050 in the test set. Boxed numbers indicate the best results; N/A stands for data not available.) 171

5.4 Normalized mean square errors for single-step prediction errors for the MacKey-Glass time series (1500 steps) using different prediction models. 178

5.5 Normalized single-step prediction errors for the Hénon map time series by using different prediction models. The number of weights includes bias weights. Both AR and DRNN results are from [8], and FIR Network result is taken from [169]. 180

5.6 Normalized single-step prediction errors for Lorenz equations by using different prediction models. The results on autoregressive and DRNN are taken from [8], and the results on FIR Netowork are taken from [169]. . . 183

5.7 Normalized single-step prediction errors for the Ikeda map by using different prediction models. Results for Autoregressive and DRNN are taken from [8], and results for the FIR Netowork are taken from [169]. 185

5.8 Comparison of single-step-prediction performance in $nMSE$ using five methods: Carbon copy (C.C), a linear model (AR), FIR [169], DRNN [8], and VGBP. Carbon copy simply predicts the next time-series data to be the same as the proceeding data ($x(t+1) = x(t)$). The training (resp. testing) set indicates patterns used for learning (resp. testing). *Lorenz attractor* has two data streams labeled by x and z, respectively, whereas *Ikeda attractor* has two streams – real ($Re(x)$) and imaginary ($Im(x)$) parts of a plane wave. Boxed numbers indicate the best results. N/A stands for data not available. 187

6.1 Three representative traditional low-pass filters and their corresponding combinations of wavelets and low-pass filters used in channel-specific preprocessing. 191

6.2 Illustration of the simple trading strategy used in our study. Column labeled by "Tx" represents the buy/sell transaction, and $P_L/P_H/P_O/P_C$ stand for daily low/high/open/close prices. 193

6.3 Prediction performance on IBM stock-price time series using ANN models with different channel-specific preprocessing alternatives. The parameter combinations in the column labeled "Parameter" can be found in Table 6.1. LP T/f represents a T-tap low-pass filter with cut-off frequency f. The tuple (a, b) stands for a ratio of $a\%$ and average relative error of b, where relative error is defined as the prediction error divided by the actual price. The annual return is calculated using the simple strategy described in Section 6.1.2. 207

6.4 Prediction performance on the ten stock-price time series benchmarks for different predictors described in Section 6.1.1. 230

6.5 Annual return of portfolios based on three predictors (AR, CNN-LP, and CNN-CSP) and Buy-and-Hold strategy for the ten different stocks. All returns are based on portfolio values without leverage. The highest annual return achieved for each stock is highlighted. 253

6.6 Annual return by year for portfolios based on CNN-CSP. The numbers in the table are percentage numbers. 255

List of Figures

1.1 An example of real-world time series: IBM daily closing prices for 200 trading days starting from January 2, 1990 to October 15, 1990. 2

1.2 An example of a nonstationary periodic noisy time series. This time series has a long-term growing trend (nonstationarity), regular alternation of peaks and valleys (periodicity), and small local fluctuations (noise). . . . 9

1.3 Laser time series data. 10

1.4 Mackey-Glass(17) time series. 13

1.5 *Sunspots* time series. The dashed vertical line divides the training and the test sets. 13

1.6 Algorithm for predictability computation. 17

1.7 Illustration of the up/down-trend metric. The model predicts an up trend when the predicted price $P_t(t+h)$ is higher than the closing price of the current day, and a down trend when $P_t(t+h)$ is lower than the current closing price. The performance of the predicted trend is defined by (1.22). A $*$ in panel (a) – (e) indicates closing price $R_c(t)$; an o sign represents predicted price $P_t(t+h)$; and the bar stands for daily low/high prices. (a) and (b) predict an up trend; (c) and (d) predict a down trend; (e) shows a flat trend; and (f) shows the notations used in these graphs. 21

1.8 Illustration of prediction errors $\varepsilon_t(t+h)$ w.r.t. the low/high price range. 22

1.9 Illustration of prediction errors w.r.t. potential buy signal when $P_t(t+h) < R_c(t)$. The bars represent the daily low/high prices, $*$ stands for the prediction $P_t(t+h)$, and o indicates the opening price. 24

1.10 Illustration of prediction errors w.r.t. potential sell signal when $P_t(t+h) > R_c(t)$. The bars represent the daily low/high prices, $*$ stands for the prediction $P_t(t+h)$, and o indicates the opening price. 24

2.1 A classification of linear and nonlinear time-series models. 35

2.2 Correlogram of the time series in Figure 1.2 that displays seasonality. The correlogram shows the autocorrelation peaks at lags of 10, 20, and 30, respectively. 46

2.3 A classification of time-series models for handling piecewise stationarity. . 49

2.4 A symmetric FIR filter with $2L$ taps. q^{-1} is a delay operator which transforms $R(t+i)$ to R(t+i-1). 55

2.5 An illustration of a filtering process on a time series of noisy IBM daily closing prices using a 20-tap symmetric low-pass FIR filter to de-noise the time series. Both the low-pass and high-pass data have a 10-day lag. The right two panels show the autocorrelation plots for both filtered time series. 55

2.6 Four techniques for handling edge effects in order to compensate for missing data in the lag period. Solid lines represent actual raw data, and dashed lines stand for extended data. 56

2.7 The average absolute errors diverge quickly when predicting missing low-pass data in the lag period of ten days. 57

2.8 Structure of a time-delayed neural network (TDNN). 59

2.9 Structure of a recurrent neural network (RNN). Here q^{-1} is the delay operator that delays $x(t)$ by one unit time. 59

2.10 *Sunspots* time series trained by backpropagation using an unconstrained formulation in (4.1). 63

2.11 *Laser* is a piecewise chaotic time-series that requires at least three validation sets: two for the regime change-over sections and another at the end of training. 64

3.1 Predictability of ten stock-price benchmarks (average price of daily low and high prices) at different prediction horizons. 69
3.2 Upper panel: autocorrelation of the average daily low and high prices for ten stocks at different prediction horizons. Lower panel: autocorrelation of the differenced series for the average daily low and high prices for the same stocks. 70
3.3 Averaged daily low and high prices for IBM stock 71
3.4 Frequency response of a 10-tap low-pass FIR filter (solid line) and the corresponding high-pass FIR filter (dashed line). 72
3.5 Low-pass filtered data (left panel) and high-pass filtered data (right panel) for the averaged daily low and high prices for the IBM stock shown in Figure 3.3. 73
3.6 Predictabilities and autocorrelations of the low-pass and high-pass data. Both low-pass and high-pass filters have 10 taps (i.e. 11 filter coefficients). 73
3.7 Spectra for the averaged daily low and high prices for the ten benchmark stocks in Table 1.3. The horizontal label for each graph is frequency and the vertical label for each graph is the magnitude for the frequency in the frequency space. 76
3.8 Frequency response of filters with cut-off frequency of $0.05, 0.10, 0.15, 0.20$, and 0.25, respectively. (a) 6-tap filters. (b) 8-tap filters. (c) 10-tap filters. (d) 12-tap filters. 79
3.9 Algorithm for redundant wavelet decomposition using a mother filter of $g(k)$. The algorithm decomposes $x(t)$ into $M+1$ bands: $w_1(t), w_2(t), \cdots, w_M(t)$ and $c_M(t)$ [111, 178, 179]. 95
3.10 Redundant wavelet decomposition using symmetric non-causal B_2-spline filters. It causes different lags in different channels. c_0 is the original signal to be decomposed. c_1, c_2 and c_3 are low-pass signals obtained using a B_2-spline filter convolved with the previous channel. The links between two consecutive channels indicate which three data elements in Channel l are used to obtain the new data element in Channel $l+1$. w_l is computed as $w_l(t) = c_{l+1}(t) - c_l(t)$. Because of the use of a non-causal filter, c_l has delay over c_{l-1}, and therefore, w_l also has delay over c_{l-1} or w_{l-1}. The lightly shaded segments in each bar indicates the number of lags incurred due to the use of symmetric mother filters, and the darkly shaded segment shows the prediction horizon. 97
3.11 Redundant wavelet decomposition using an asymmetric causal B_2-spline mother filter. It does not incur any lag for all channels. See Figure 3.10 for an explanation of the graph. 98
3.12 Redundant wavelet decomposition using B_3-spline causal mother filter. (a) Raw time series: average of daily low/high prices for IBM stock; (b) Channel H: $w_1(t)$; (c) Channel LH: $w_2(t)$; (d) Low-pass channel: $c_2(t)$. . . 99
3.13 Frequency response of the three bands obtained by wavelet decomposition using a B_2-, B_3-, B_4-, and B_6-spline mother filters in Panels (a), (b), (c), and (d) respectively. 101
3.14 Transforming a nonstationary series into a chaotic series by performing standardization on input patterns. 103
3.15 Signals before (left panels) and after (right panels) performing input pattern standardization for a) Citigroup, b) IBM, and c) Exxon-Mobil. . . . 104
3.16 Wavelet decomposition is performed first, then different low-pass filters can be applied to the decomposed bands according to the noise level in each band. 105

3.17 Wavelet decomposition and low-pass filtering on channels H and LH. The left panels plot the filter coefficients for each wavelet channel. The right panels plot the coefficients of the low-pass filters applied to the two high-frequency channels. The horizontal axes represent the indices for filters and the vertical axes are for filter coefficients. 106
3.18 Coefficients of the equivalent composite filter for the preprocessing process illustrated in Figure 3.17. 108
3.19 Magnitude of the frequency response for the composite filter. 109
3.20 Phase angle of the frequency response for the composite filter. 109
3.21 Coefficients of composite filter B4_63_43 and its corresponding optimal low-pass FIR filter with zero phase (upper panel) and their preprocessed IBM stock prices. 111
3.22 Coefficients of composite filter B4_82_62 and its corresponding optimal low-pass FIR filter with zero phase (upper panel) and their preprocessed IBM stock prices. 112
3.23 Coefficients of composite filter B6_81_61 and its corresponding optimal low-pass FIR filter with zero phase (upper panel) and their preprocessed IBM stock prices. 113
3.24 Range error in predicting the low-frequency channel of IBM stock price at horizon of one using different wavelet decomposition. The numbers $a/b/c$ in the legends indicate the number of predictions in/above/below the daily low-high price range. 117
3.25 The predicted values of the low-frequency channel of IBM stock prices at horizon of one using different wavelet decompositions. The numbers $a/b/c$ in the legends indicate the number of predictions in/above/below the daily low-high price range. 118
3.26 Frequency response of the composite filter compared with the 6 tap low-pass filter with a cut-off frequency of 0.15π. Upper panel: magnitude response. Lower panel: phase shift. 120

3.27 Frequency response of the composite filter compared with the 8 tap low-pass filter with a cut-off frequency of 0.10π. Upper panel: magnitude response. Lower panel: phase shift. 121
3.28 Frequency response of the composite filter compared with the 10 tap low-pass filter with a cut-off frequency of 0.05π. Upper panel: magnitude response. Lower panel: phase shift. 122

4.1 Structure of recurrent FIR neural network (RFIR) and FIR filter. Each thick line in RFIR represents a FIR filter. A single circle represents a regular node, a double circle stands for a recurrent node, and a small square is a bias node with constant input 1. A square with q^{-1} indicates a one-unit time delay. 136
4.2 Multiple validation sets in a training set. V_1,V_2 and V_3 are three validation sets. 144
4.3 Illustration of a constraint in the lag period for financial time-series predictions. The raw data (stock price) should center around the low-pass data as well as the ANN outputs. 146
4.4 An iterative learning procedure using a constrained formulation for ANN time-series prediction. The shaded box represents the routine to look for ESP_c. R&T stands for our *relax-and-tighten* strategy. 152
4.5 Decreases of the maximum violation over all training patterns for the MG17 time series when different initial violation tolerance τ's (broken lines) are used and when our relax-and-tighten (R&T) strategy (solid line) is used. 156
4.6 Procedure of violation-guided back-propagation algorithm. Refer to Section 4.4.4 for a detailed explanation for each line. $e_i(t)$ is error between the network output and the desired output defined in (4.4). 158
4.7 Progress of MSE defined in (4.1) for $T = 1$ and $T = \infty$ during the learning of an ANN to predict the MG17 time series. 160

4.8 Decreases of the maximum violation over all constraints for the MG(17) time series between using a fixed temperature ($T = 5$) and using a schedule of decreasing temperatures ($T_0 = 5$, $T_{i+1} = 0.25T_i$ and $T_{i+1} = 0.95T_i$ every 4000 evaluations, $T_\infty = 0.0003$, and η_0 was adjusted every 50 evaluations). 161
4.9 Robustness of VGBP with respect to α in predicting the MG-17, MG-30, and sunspots time series. 163

5.1 *Sunspots* time series. Dashed vertical lines divide the training and testing sets. 165
5.2 Single-step (a) and iterative prediction results of an ANN trained by VGBP for the *laser* time-series. The solid line plots the actual data and the dashed line represents the prediction data. 172
5.3 MacKey-Glass time series with (a) $\Delta = 17$ and (b) $\Delta = 30$. 174
5.4 A comparison of 100-step iterative predictions on two sets of *Mackey-Glass* time series. Solid lines represent the actual data; long dashed lines indicate the predicted data using VGBP; and short dashed lines are prediction results by running Wan's FIR-NN training algorithm in [169]. Our predictions (long dashed lines) are almost indistinguishable from the actual data for the most part of the two graphs. 177
5.5 Phase plot of Henon map: $x(t)$ versus $x(t-1)$. 179
5.6 Phase plot of Lorenz time-series: x versus z. (a) Actual time series, and (b) iteratively predicted time series. The parameters used to generate (b) are listed in Table 5.6. 182
5.7 Phase plot of the Ikeda map time series: real part $Re[z]$ versus imaginary part $Im[z]$. (a) Original Ikeda attractor phase plot. (b) Predicted Ikeda attractor phase plot. Parameters used to generate (b) are listed in Table 5.7.184

6.1 Prediction performance on Channel LL of IBM stock-price time series after 3-channel wavelet decomposition with a mother filter of B_3-spline filter. The relative error is the prediction error divided by the actual price. . . . 196

6.2 Predicted values for 3 lags and horizon 1 of Channel LH. 198
6.3 Prediction performance on combined Channels LH and LL of IBM stock-price time series after 3-channel wavelet decomposition with a B_3-spline mother filter. 200
6.4 Predicted values for 4 lags and horizon 1 of Channel H. 202
6.5 Prediction performance on all three channels combined for IBM stock-price time series after 3-channel wavelet decomposition with a B_3-spline mother filter. 203
6.6 Portfolio value (with initial value of \$1000000) grows with time. Each panel plots the evolution of portfolio values with time when a target filter and its corresponding channel-specific preprocessing are used. The upper, middle, and lower panels are for target filters of 6/0.15, 8/0.10, and 10, 0.05 respectively. 208
6.7 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for AMR Inc using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 210
6.8 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Citigroup using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 211
6.9 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for General Electric using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 212
6.10 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for IBM stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 213

6.11 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Mentor using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 214
6.12 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for New York Times using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 215
6.13 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Provident Financial Group using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 216
6.14 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Payless ShoeSource using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 217
6.15 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Exxon-Mobil using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 218
6.16 Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Yahoo using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 219
6.17 Prediction errors w.r.t. potential buy and sell signals for AMR Inc stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 220
6.18 Prediction errors w.r.t. potential buy and sell signals for Citigroup stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 221

6.19 Prediction errors w.r.t. potential buy and sell signals for General Electric stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 222
6.20 Prediction errors w.r.t. potential buy and sell signals for IBM stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 223
6.21 Prediction errors w.r.t. potential buy and sell signals for Mentor stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 224
6.22 Prediction errors w.r.t. potential buy and sell signals for New York Times stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 225
6.23 Prediction errors w.r.t. potential buy and sell signals for Provident Financial Group stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 226
6.24 Prediction errors w.r.t. potential buy and sell signals for Payless ShoeSource stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 227
6.25 Prediction errors w.r.t. potential buy and sell signals for Exxon-Mobil stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 228
6.26 Prediction errors w.r.t. potential buy and sell signals for Yahoo stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 229
6.27 Actual predictions for AMR Inc stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 233
6.28 Actual predictions for Citigroup stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 234

6.29 Actual predictions for General Electric stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 235
6.30 Actual predictions for IBM stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). . 236
6.31 Actual predictions for Mentor stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 237
6.32 Actual predictions for New York Times stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 238
6.33 Actual predictions for Provident Financial Group stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 239
6.34 Actual predictions for Payless ShoeSource stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 240
6.35 Actual predictions for Exxon-Mobil stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 241
6.36 Actual predictions for Yahoo stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel). 242
6.37 Last 200 predictions of Figure 6.35. 243
6.38 Last 200 predictions of Figure 6.35. 244
6.39 Last 200 predictions of Figure 6.35. 245
6.40 Portfolio value grows with time for trading AMR Inc stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $31.560 and $2.400, respectively. 247

6.41 Portfolio value grows with time for trading Citigroup stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $2.880 and $33.390, respectively. 248
6.42 Portfolio value grows with time for trading General Electric stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $5.285 and $23.965, respecitively. 248
6.43 Portfolio value grows with time for trading IBM stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $16.895 and $77.735, respectively. 249
6.44 Portfolio value grows with time for trading Mentor stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $4.200 and $17.375, respectively. 249
6.45 Portfolio value grows with time for trading New York Times stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $12.005 and $46.230, respectively. 250
6.46 Portfolio value grows with time for trading Provident Financial Group stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending day days of the investment duration are $10.245 and $28.925, respectively. 250
6.47 Portfolio value grows with time for trading Payless ShoeSource stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are $14.540 and $15.635, respectively. 251

6.48 Portfolio value grows with time for trading Exxon-Mobil stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$14.705 and \$33.175, respectively. 251
6.49 Portfolio value grows with time for trading Yahoo stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$2.015 and \$20.465, respectively. 252

Chapter 1

Introduction

1.1 Motivations of this research

A time series is an ordered sequence of observations made through time. Time series occur everywhere in our daily life. For example, they are observed in financial markets (such as stock prices, bond values, and foreign exchange rates), in marketing statistics (such as daily sales revenues, daily gasoline prices, and monthly demand and supply), in physical science (such as daily temperatures in a city, yearly sunspots in solar activity, and scientific experiments), in social science (such as community population and crime rates), and in many other areas. An example of real-world stock market time series (daily closing prices for IBM stock) is plotted in Figure 1.1.

A time-series prediction problem entails the estimation of values of a certain number of future observations. It is important in many applications if one can predict a time series to a certain degree of accuracy. For example, if an investment firm is able to predict a certain financial time series slightly better than random guesses, then there is a good opportunity to profit. If a company can predict its monthly sales and customer demands accurately, it will be much easier to arrange its business plan efficiently. To predict a

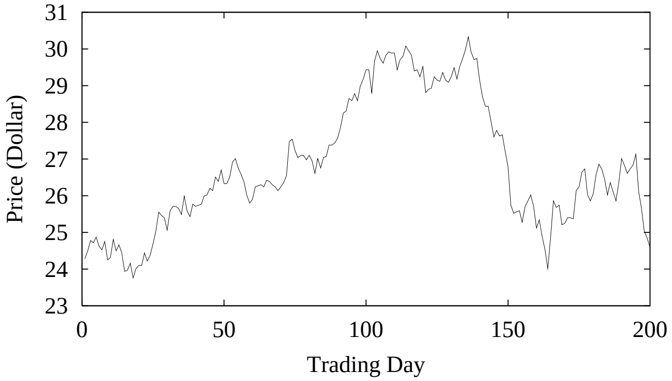


Figure 1.1: An example of real-world time series: IBM daily closing prices for 200 trading days starting from January 2, 1990 to October 15, 1990.

time series is to forecast the future. If one is able to forecast a time series accurately, he/she will always be in a better position in a competition or in a good position to plan for future events. Due to its great benefits, great efforts have been devoted in time-series analysis and forecasting in recent years.

Unfortunately, the underlying dynamics that governs a time series is generally unknown. This is where neural networks can help. A neural-network model is a nonlinear model that can model any continuous nonlinear function. Moreover, neural-network models do not impose any assumption, stochastic or deterministic, on the underlying dynamic process that generates the time series. Instead, neural network models detect the underlying mathematical function by "learning" from history and predict the future, based on information learned through history. As a result, neural-network models are

natural tools for modeling time series. Due to this reason, neural-network models have gained popularity in time-series analysis and prediction [173].

However, learning has always being a challenge in neural-network models. It is well-known that there are many local minima in the search space of a neural-network model in such a way that it is very hard to find good solutions in practice [162]. Although a variety of learning algorithms have been proposed, escaping from local minima effectively is largely unsolved [137]. One fundamental problem common to existing learning algorithms is that they all try to minimize the mean square errors of neural-network outputs; i.e., these algorithms only care about the overall average effect. In fact, the individual pattern's behavior is also very important. Consider the case when a search is stuck in a local minimum but not the global minimum. At that point, there must exist some patterns that still have nonzero errors. But the traditional mean-square-error formulation never identifies those patterns with nonzero errors and does not provide guidance on the search. Consequently, a new neural-network model that considers individual pattern's behavior would be very helpful.

Another motivation for this research is that neural networks have not been as popular in financial and economic time-series modeling as in many other applications [88, 89, 90]. One of the goals in this thesis is to provide a comprehensive study in this field by applying neural-network learning in financial stock-market time-series predictions. We also hope that our study will provide a stepping stone for future research in this area.

1.2 Problems addressed

The time-series prediction problem studied in this work is defined as follows. Given $R(t), t = 1, \cdots, t_0$, a sequence of time-series data from time $t = 1$ to the current time

$t = t_0$, a time-series prediction problem entails the prediction of future data $R(t_0 + h)$ with $h > 0$. Here h is called prediction horizon, and each element of this horizon is a prediction step. $R(t)$ can be expressed in the form of a vector or a scalar. When $R(t)$ is a scalar, the time series is called a uni-variate time series; otherwise, it is called a multi-variate time series. $R(t)$ can be either discrete or continuous.

Time series can be generated by many physical forces at the time. In practice, some critical domain-specific information may not be available or incomplete; the observed outputs can take a numerical or a symbolic format that is hard to be represented mathematically; and the underlying dynamics that governs the generating process may be deterministic or stochastic. Therefore, a lot of time series are not predictable based on information collected. In our work, we are only interested in time series that are deterministic and predictable.

In order to predict future data values, a time-series sequence must contain historical observations that are related to future observations [16]. If there is no historical data, one cannot build a sound model to predict the future. On the other hand, if most of the observations are independent (unrelated to future observations), then it is difficult to infer future data values from historical data, as they are uncorrelated. In this thesis, we only consider time series consisting of dependent observations.

To predict a time series, five steps are usually followed.

- **Step 1.** Collect and prepare the historical data of a time series to be modeled. This step has to be done carefully in order to prevent errors to propagate from one step to another. In this step, one may also need to consider the number of past data items to be used to fit a time-series model.

- **Step 2.** Select an appropriate time-series model. In this case, application-specific domain knowledge may greatly help make the decision on which model to use. For example, if one is certain that it is a linear time series, then a linear model is a better choice than a complicated nonlinear model. In this step, one has to identify the properties of the time series (e.g. linearity, stationarity, and noise) in order to choose an appropriate model.

- **Step 3.** Train the selected time-series model with historical data produced in Step 1 in order to find a good set of parameters for the model selected. This step, called model learning, is the most difficult step. The learning problem is normally formulated as an optimization problem and solved using optimization algorithms. The solution is a set of optimal or near optimal parameters for the model selected. More often than not, the optimization problem resulted from the time-series model is a complicated optimization problem, either due to its large size or nonlinearity, and often it is very hard to find optimal solutions or even good sub-optimal solutions.

- **Step 4.** After a model has been fit against the historical data, one would be interested to see if the model can generalize well before applying it to predict future data. This step is called cross-validation. Normally, the part of the historical data reserved for validation is not used in Step 3 to fit the model.

- **Step 5.** Apply the trained time-series model to predict future data values. This step is normally straightforward. Further actions (such as postprocessing) can be taken according to the prediction results in some applications.

Steps 3 and 4 are sometimes merged into a single step in which learning and cross-validation are applied simultaneously.

1.2.1 Chaotic time series

As stated before, we are only interested in time series that are deterministic and predictable. This kind of time series is generated by a deterministic dynamic system which can be described by a set of differential equations in general. There are three types of dynamic systems: catastrophic, stable, and chaotic [81]. A catastrophic process has a trajectory that is unbounded, such as the explosion of an atomic bomb or the Big Bang. A stable process has a trajectory that is periodic or quasi-periodic. There are plenty of examples of stable processes, such as planets rotating along a fixed orbit and trains moving on tracks. A chaotic process has a trajectory jumping between different sub-paths.

A chaotic process behaves very similar to a random process but is deterministic. Regarding the behavior of a chaotic process, consider the following equation:

$$x_{n+1} = 2x_n(1 - x_n) \tag{1.1}$$

with initial value $x_0 \in [0, 1]$. This equation gives successive data values by iterating itself from a given initial value x_0. It is easy to show that it has two fixed points $x = 0.5$ and $x = 0$; that is, once $x_k = 0.5$ (or $x_k = 0$), then $x_i = 0.5$ (or $x_i = 0$) for all $i > k$. Table 1.1 shows several different sequences of x's with different starting points. One can see that although both $x = 0.5$ and $x = 0$ are fixed points, only $x = 0.5$ is a stable fixed point, whereas $x = 0$ is an unstable fixed point. In other words, when the value of x is slightly different from $x = 0$, then the trajectory moves away from $x = 0$; whereas for all $x \in (0, 1)$, it converges towards 0.5. An unstable fixed point is called a *repellor*. We call

Table 1.1: Data series for different starting points according to (1.1).

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$\cdots$
-0.0100	-0.0202	-0.0412	-0.0858	-0.1864	-0.4423	-1.2757	-5.8066	$\cdots$
0.0100	0.0198	0.0388	0.0746	0.1381	0.2381	0.3628	0.4623	$\cdots$
0.4000	0.4800	0.4992	0.5000	0.5000	0.5000	0.5000	0.5000	$\cdots$
0.6000	0.4800	0.4992	0.5000	0.5000	0.5000	0.5000	0.5000	$\cdots$

a time series *chaotic* if there are more than one repellors presenting in the time-series process, and the observed values bounce back and forth between repellors [143]. The long-term mean and variance for a chaotic process are stable, although the mean and variance can change over a short time period.

According to Takens Theorem [146, 2, 1, 14], given a time series $[x(1), x(2), \cdots, x(T)]$ sampled from a chaotic system, the state of the underlying system can be reconstructed from windows $[x(t), x(t-\tau), \cdots, x(t-(m-1)\tau)], \forall t = ((m-1)\tau + 1), \cdots, T$, where m is called the **embedding dimension** and τ is called the **embedding delay**. According to Takens' theorem, one can express time-series value $x(t+1)$ as

$$x(t+1) = f(x(t), x(t-\tau), \cdots, x(t-(m-1)\tau)), \tag{1.2}$$

where h is the prediction horizon [146, 1, 14].

The study on chaotic systems are mainly in three areas, namely, *identification of chaotic behavior*, *modeling and prediction*, and *control* [78, 82]. In this thesis, we are only interested in *modeling and prediction*. In the first area, it concentrates on how to identify chaotic systems from stochastic ones, and provides estimates of embedding dimension m and embedding delay τ. τ can be determined by the autocorrelation function [2], and m can be determined by *False Nearest Neighbors* which can be found in [1, 3]. More work on finding the embedding delay and the embedding dimension can be found in [78, 82]. In the area of *control*, it involves the control of a chaotic system. To be more specific, one may take advantage of the chaotic behavior to obtain a "large" desired effect using a "small" control signal [30, 141]. Since (1.2) can be generalized to

$$x(t+1) = f(x(t), x(t-1), x(t-2), \cdots, x(t-n)), \tag{1.3}$$

we can leave all the work of figuring out τ, m, and function f to the chaotic time series model itself. When it is hard to find the embedding delay and the embedding dimension accurately, there is no much difference between modeling (1.2) and modeling (1.3). In our work, we simply skip the identification phase and directly model (1.3).

Chaotic time series is the center piece of this thesis, and the new time series model presented in this thesis is used to predict chaotic time series. In order to apply our approach on time series that is not chaotic, we propose a new preprocessing approach in order to convert it to one or multiple sets of (near) chaotic time series.

1.2.2 Characteristics of time series

Although a linear stationary noise-free time series is relatively easy to model and predict, a general time series is more difficult to predict because it may exhibit nonlinearity, non-stationarity, and possibly periodic behavior such as seasonality. More often than not, observations may be contaminated by noise. Figure 1.2 illustrates a noisy nonstationary periodic time series. The four main characteristics of a general time series are described as follows.

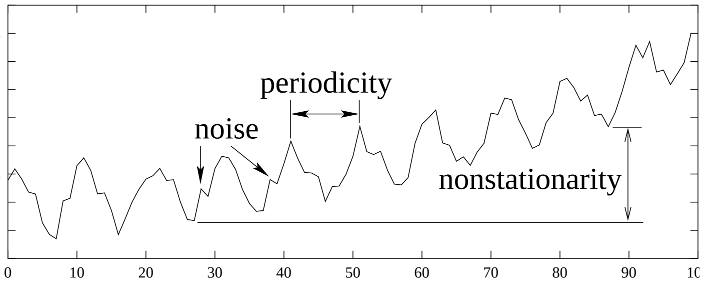


Figure 1.2: An example of a nonstationary periodic noisy time series. This time series has a long-term growing trend (nonstationarity), regular alternation of peaks and valleys (periodicity), and small local fluctuations (noise).

1.2.2.1 Nonlinearity

Consider $R(t_0+h)$ as a function of historical values:

$$R(t_0+h) = f(R(t_0), R(t_0-1), \cdots). \tag{1.4}$$

We say that a time series is linear if $f()$ is a linear function, and a nonlinear time-series otherwise. Linear time series have been well studied by Box and Jenkins using a class of linear stochastic processes [16], which led to the well-known ARIMA models. In this thesis, we are interested in developing general models that can represent nonlinear time series governed by an arbitrary continuous function.

1.2.2.2 Seasonality

A time series with dominant periodic components will exhibit regular periodic variations. Such behavior can often be found in annual electricity consumption, merchandise sales,

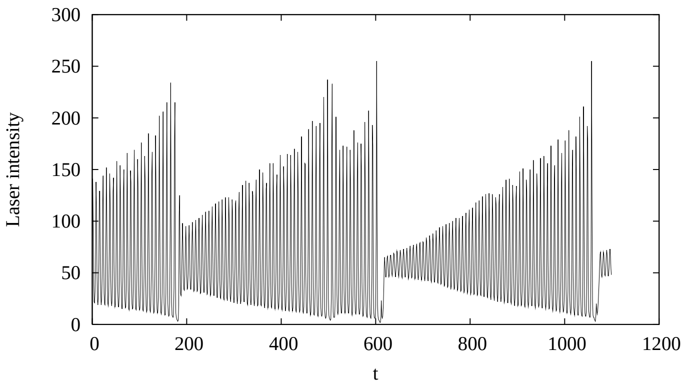


Figure 1.3: Laser time series data.

and many other events that fluctuate with respect to seasons. Periodicity is a relatively well-studied property and can be eliminated effectively by differencing techniques [16, 21]. The chaotic model is then applied to the differenced time series, assuming the differenced time series is chaotic.

1.2.2.3 Piecewise chaos

A time series is *piecewise chaotic* if it consists of several regimes in which each regime corresponds to a chaotic process, and the overall time series is a collection of multiple chaotic regimes. An example of such a time series is the laser time series shown in Figure 1.3. To apply a chaotic time-series model, one needs to identify regime shifts correctly before applying properly trained chaotic time-series models.

1.2.2.4 Nonstationarity

A time series is called stationary if its mean is constant and its auto-covariance function depends only on the lag (distance) [21, 22]; namely,

$$E[R(t)] = \mu \tag{1.5}$$

is constant over time t and

$$\mathrm{Cov}[R(t), R(t+\tau)] = \gamma(\tau) \tag{1.6}$$

is dependent only on lag τ. Here $E[X]$ is the expectation of random variable X, and the auto-covariance is defined as:

$$\gamma(\tau) = \mathrm{Cov}[R(t), R(t+\tau)] = E[(R(t)-\mu)(R(t+\tau)-\mu)]. \tag{1.7}$$

In practice, autocorrelation, which is autocovariance normalized by $\gamma(0)$, is often used:

$$\rho(\tau) = \gamma(\tau)/\gamma(0). \tag{1.8}$$

In general, nonstationarity is hard to model, as its future behavior may be unpredictable. In this thesis, we try to transform a nonstationary time series into a chaotic time series by using a new preprocessing approach.

1.2.2.5 Noise

Noise can be present in the entire or some parts of the frequency spectrum of a time series. Since random noise cannot be predicted, we perform denoising before fitting a time-series model. In this thesis we only study time series with high frequency noise. Typical examples of time series contaminated by high frequency noise are daily prices of instruments (stocks, bonds, options, *etc.*) in financial markets. Again, we try to transform noisy time series to one or multiple sets of chaotic time series in our work.

We review existing work on handling nonlinearity, nonstationarity, seasonality, and noise in time series in Chapter 2.

1.2.3 Assumptions on time series studied in this thesis

In this thesis, we only study time series that fall into following categories:

1. Noiseless or near noiseless nonlinear time series that is chaotic or near chaotic. Examples of such time series are the Mackey-Glass(17) time series (Figure 1.4) and the sunspots time series (Figure 1.5).

2. Noiseless or near noiseless piece-wise chaotic nonlinear time series, such as laser time series shown in Figure 1.3.

3. Financial time series that is nonstationary and noisy but can be transformed into one or multiple sets of near chaotic time series. The stock-market time series shown in Figure 1.1 is an example. This type of time series contains high-frequency noise and exhibits nonstationarity, which makes the prediction task very difficult.

4. We assume that a denoised financial time series is not a random walk and that the denoised future value is highly correlated to its historical values. In this thesis, we focus on exploring variations in the values of a financial time series alone and ignore other quantitative factors. Hence we are only interested in univariate models for financial time series.

5. Noise in financial time series is mainly high frequency noise. Predictions are feasible when the signal-to-noise ratio of the series is considerably high. Consequently, we don't study penny stocks whose signal-to-noise ratio is usually low.

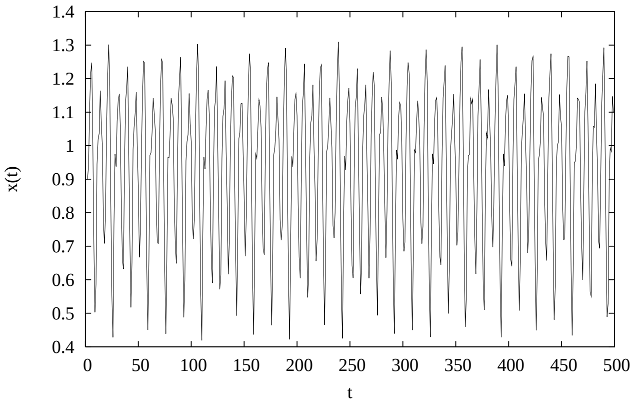


Figure 1.4: Mackey-Glass(17) time series.

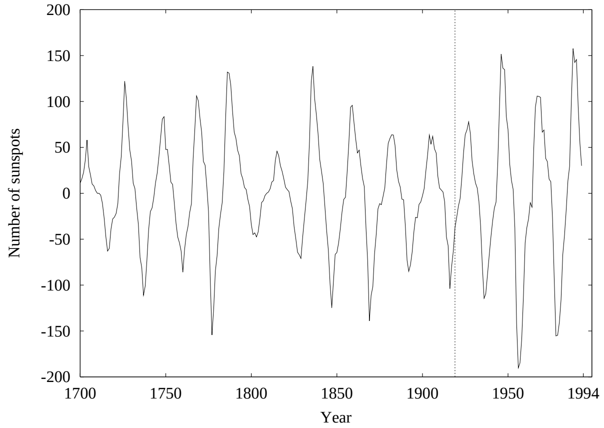


Figure 1.5: *Sunspots* time series. The dashed vertical line divides the training and the test sets.

1.3 Metrics

In this section, we present the metrics for measuring the behavior of time series studied in this thesis. We divide these metrics into three categories: property metrics, general performance metrics, and special performance metrics for financial time series.

1.3.1 Metrics for properties

Each time series has certain properties that somehow indicate how easy or difficult it can be predicted. Here we present two standardized measures: the autocorrelation measure and the predictability concept. Autocorrelation is widely used in time-series analysis, whereas predictability is a relatively new term [119, 120, 86].

1.3.1.1 Autocorrelation

Autocorrelation has been defined in (1.8). For convenience, we rewrite its definition in a single equation for time-series $X(t)$ as follows:

$$\rho(\tau) = \frac{E[(X(t) - \mu_X)(X(t-\tau) - \mu_X)]}{\sigma_X^2}, \text{ t=}, 0, 1, \cdots \tag{1.9}$$

where σ_X^2 is the variance of $X(t)$ [15, 21, 22].

1.3.1.2 Predictability

Before defining predictability, we review some basic concepts used in information theory [119, 120, 86].

Consider random variable X that takes N discrete values $\{x_1, x_2, \cdots, x_{N-1}, x_N\}$ with probability $\{p_1, p_2, \cdots, p_{N-1}, p_N\}$. The **entropy** of X is defined as [64, 86]:

$$H(X) = -\sum_{i=1}^{N} p_i \log_2 p_i. \tag{1.10}$$

The entropy of random variable X can be interpreted as a measure of its uncertainty; the larger the value $H(X)$ is, the more uncertain it will be to determine the next value of X.

The **joint entropy** [120, 86] between two discrete random variables X and Y defines the uncertainty of the joint pair of (X, Y). Assuming that random variable Y takes M discrete values $\{y_1, y_2, \cdots, y_{M-1}, y_M\}$, and that the joint probability distribution of $\{X, Y\}$ is $p(x_i, y_j)$ for all $i = 1, 2, \cdots, N$ and $j = 1, 2, \cdots, M$, then the joint entropy between X and Y is defined as:

$$H(X, Y) = -\sum_{i=1}^{N}\sum_{j=1}^{M} p(x_i, y_j) \log_2 p(x_i, y_j). \tag{1.11}$$

The **mutual information** [120, 86] between discrete random variables X and Y is defined as:

$$I(X;Y) = H(X) + H(Y) - H(X, Y), \tag{1.12}$$

where $I(X;Y)$ reflects the common information between X and Y. Here, the semicolon ";" notation is used as is in Palus's paper [120]. Mutual information $I(X;Y)$ achieves the minimum value of 0 when X and Y are independent, and achieves the maximum value of $H(X)$ when $X = Y$. Hence $I(X;Y)$ reflects the common information between X and Y and is, therefore, called mutual information.

We can now define the **predictability** [120, 86] of a time series for a horizon τ as:

$$P_\tau(X) = \frac{I(X(t); X(t+\tau))}{H(X)}, \tag{1.13}$$

where $H(X)$ is used to normalize the predictability in order for $P_\tau(X)$ to be between 0 and 1 [120].

Since the predictability defined above is for discrete random variables, we need to extend it to continuous time series. One approach is to discretize a continuous time series into a discrete one. Further, predictability can only be applied to stationary or near stationary time series because a nonstationary trend cannot be considered random. For example, a long-term nonlinear trend in a financial time series must first be removed by differencing before its predictability can be assessed. In this case, one is interested in predicting the differenced series rather than the original time series. Based on the above discussion, Figure 1.6 presents the algorithm for computing predictability.

In the algorithm presented in Figure 1.6, we use a method called box-counting [119] to compute (1.10) – (1.13). The reason to perform box-counting is to avoid over-quantization, since over-quantization leads to a small number of data item in some quantized boxes. Statistically, when the number of data items is too small, the estimation of its probability is meaningless. Box-counting requires the data length N and the quantization level Q to satisfy the following inequality [119, 120, 64, 86]:

$$Q^3 \leq N, \tag{1.14}$$

and the box boundaries to be determined in such a way that the number of data elements in each box is approximately the same.

Applying (1.10) – (1.13) to the IBM daily stock prices plotted in Figure 1.1, it results in an entropy of 1.791361 for the differenced series and a joint entropy (*resp*, mutual in-

Step 1.	Perform differencing if needed to remove long-term trend;
Step 2.	Find quantization boundaries such that the number of data falling into each quantization interval is approximately the same;
Step 3.	Compute $p(i)$ in (1.10) by counting the number of data items falling into each interval, and calculate entropy $H(x(t))$;
Step 4.	Compute $p(i,j)$ in (1.11) by counting the number of data pairs $(x(t), x(t+\tau))$ falling into each 2-D grid (i,j) formed by quantization boundaries, and calculate joint entropy $H(x(t), x(t+\tau))$;
Step 5.	Compute predictability P_τ according to (1.13)

Figure 1.6: Algorithm for predictability computation.

formation) of 3.468043 (*resp*, 0.114679) between the differenced series and the differenced series lagging by one day. Therefore, the predictability for the series is 0.064018. This predictability measure is extremely low and it indicates that the stock-price time series is hard to predict directly.

1.3.2 General performance metrics

The metrics described in this section can be used to measure the performance of a general time series. In this category, we introduce the normalized mean squared error and its variants, as well as the correlation between the predicted and the actual desired values.

1.3.2.1 Normalized mean square errors

The *normalized mean square error (nMSE)* is defined as follows:

$$nMSE = \frac{1}{\sigma^2 N}\sum_{t=t_0}^{t_1}(o(t) - d(t))^2, \tag{1.15}$$

where $o(t)$ and $d(t)$ are, respectively, the actual and desired outputs at time t; σ^2 is the variance of the targeted time series in period $[t_0, t_1]$; and $N = t_1 - t_0 + 1$ is the number of patterns tested.

Variants of $nMSE$ include the *mean square error* (MSE):

$$MSE = \frac{1}{N}\sum_{t=t_0}^{t_1}(o(t) - d(t))^2, \tag{1.16}$$

mean absolute error (MAE):

$$MAE = \frac{1}{N}\sum_{t=t_0}^{t_1}\left|o(t) - d(t)\right|, \tag{1.17}$$

and *relative mean absolute error* (RMAE):

$$RMAE = \frac{1}{N}\sum_{t_0}^{t_1}\frac{|(o(t) - d(t))|}{d(t)}. \tag{1.18}$$

Here, $RMAE$ is the average of the absolute difference between the desired data value and the actual data value normalized by the desired data value. It is applicable only to time series with positive values, such as financial time series.

One can see that the normalized mean squared error and its variants emphasize more the average behavior of the predictions rather than the behavior of individual patterns, as large errors in a few patterns may be scaled down significantly when N is large.

1.3.2.2 Correlation between predicted and actual data

Let μ_X be the mean of random variable X. Similar to (1.7) and (1.8), the covariance between the predicted value $o(t)$ and the target value $d(t)$ is:

$$\mathrm{Cov}(o(t), d(t)) = E[(o(t) - \mu_o)(d(t) - \mu_d)], \tag{1.19}$$

and the correlation between the predictions and the targets is:

$$\gamma(o(t), d(t)) = \frac{\mathrm{Cov}(o(t), d(t))}{\sigma_o \sigma_d}, \tag{1.20}$$

where σ_o and σ_d are, respectively, the standard deviations of $o(t)$ and $d(t)$. The correlation is always in the range $[-1, 1]$ and emphasizes the similarities between the predicted stream and the target stream. For example, if one multiplies $o(t)$ by a constant, then the correlation stays unchanged. Hence, if the prediction stream has exactly the same shape but with different mean or scale as compared to the target stream, then their correlation will be 1.

1.3.3 Performance metrics for financial time series

The performance metrics presented in this section are suitable only for evaluating financial time series. The metrics introduced in Section 1.3.3.1 are used for an overall coarse-level performance evaluation and are strategy-related, whereas the metrics in Sections 1.3.3.2 to 1.3.3.4 provide fine-level performance evaluation.

1.3.3.1 Annual/monthly return

The annual/monthly return is related to the trading strategy used. For example, the annual return of a stock is related to its buy-and-hold strategy. It measures how fast one can increase his/her asset under a certain strategy. Suppose $M(t_0)$ is the total asset in a portfolio at the beginning of the horizon t_0, and $M(t_1)$ is the total asset at the end of the horizon t_1, then the corresponding return over this period is defined as:

$$R_m = \left.\frac{M(t_1) - M(t_0)}{M(t_0)}\right|_{\mathcal{S}}, \tag{1.21}$$

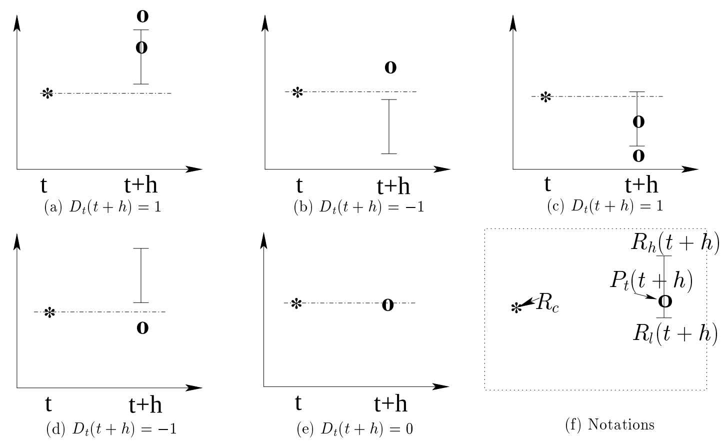
where $\mathcal{S}$ is the trading strategy used. This metrics is useful for evaluating performance over a fixed period, such as one year or one month. If the fixed period is one year (resp. one month), then the resulted return is the annual return (resp. monthly return).

1.3.3.2 Up/down trend

The daily stock price in day t can be summarized in four values, namely, daily low price $R_l(t)$, daily high price $R_h(t)$, daily opening price $R_o(t)$, and daily closing price $R_c(t)$. At day t, let $P_t(t+h)$ be the predicted price for trading day $t+h$. The metrics for up/down trend prediction is defined as:

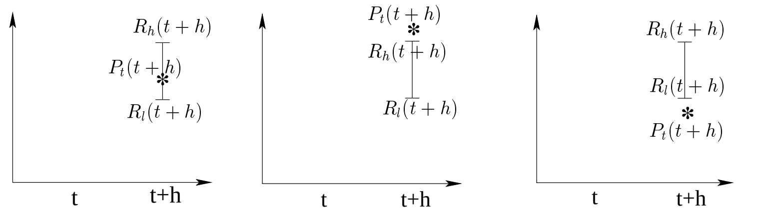
$$D_t(t+h) = \begin{cases} +1, & \text{if } P_t(t+h) > R_c(t) \text{ and } R_h(t+h) \geq R_c(t), \\ -1, & \text{if } P_t(t+h) > R_c(t) \text{ and } R_h(t+h) < R_c(t), \\ +1, & \text{if } P_t(t+h) < R_c(t) \text{ and } R_l(t+h) \leq R_c(t), \\ -1, & \text{if } P_t(t+h) < R_c(t) \text{ and } R_l(t+h) > R_c(t), \\ 0, & \text{if } P_t(t+h) = R_c(t). \end{cases} \tag{1.22}$$

Obviously, $D_t(t+h) = 1$ indicates a prediction in the correct direction (trend) for horizon h, and $D_t(t+h) = -1$ indicates a prediction in the wrong direction for horizon h. Figures 1.7(a)-(e) illustrate the five cases listed in (1.22) in the same order, and Figure 1.7f shows the notations used in these graphs.



(a) $D_t(t+h) = 1$ (b) $D_t(t+h) = -1$ (c) $D_t(t+h) = 1$ (d) $D_t(t+h) = -1$ (e) $D_t(t+h) = 0$ (f) Notations

Figure 1.7: Illustration of the up/down-trend metric. The model predicts an up trend when the predicted price $P_t(t+h)$ is higher than the closing price of the current day, and a down trend when $P_t(t+h)$ is lower than the current closing price. The performance of the predicted trend is defined by (1.22). A $*$ in panel (a) – (e) indicates closing price $R_c(t)$; an o sign represents predicted price $P_t(t+h)$; and the bar stands for daily low/high prices. (a) and (b) predict an up trend; (c) and (d) predict a down trend; (e) shows a flat trend; and (f) shows the notations used in these graphs.



(a) $\varepsilon_t(t+h) = 0$ (b) $\varepsilon_t(t+h) = P_t(t+h) - R_h(t+h)$ (b) $\varepsilon_t(t+h) = P_t(t+h) - R_l(t+h)$

Figure 1.8: Illustration of prediction errors $\varepsilon_t(t+h)$ w.r.t. the low/high price range.

1.3.3.3 Errors with respect to low/high price range

When a prediction for stock price at day $t+h$ is made, the predicted price consists of a single value but the daily price is a four tuple. In this section, we consider a prediction to be accurate when it falls between the daily low and high price range; otherwise, we measure the error by how far the prediction is from the low/high price range. Mathematically, the error with respect to the low/high price range is defined as follows:

$$\varepsilon_t(t+h) = \begin{cases} 0 & \text{if } R_l(t+h) \le P_t(t+h) \le R_h(t+h) \\ P_t(t+h) - R_h(t+h) & \text{if } P_t(t+h) > R_h(t+h) \\ P_t(t+h) - R_l(t+h) & \text{if } P_t(t+h) < R_l(t+h). \end{cases} \tag{1.23}$$

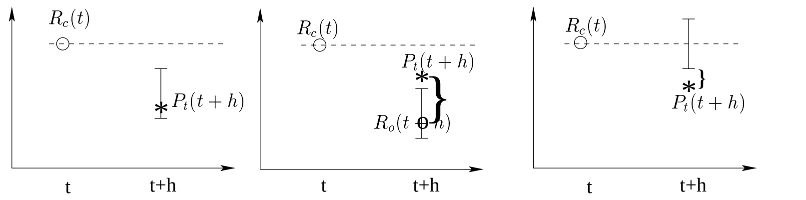
Figures 1.8(a)–(c) illustrate the three cases listed in (1.23). Panel (a) shows a perfect prediction without error, as the predicted value falls between the daily low and high price range at time $t+h$. Panel (b) shows the case that over-predicts the price at time $t+h$ and incurs a positive error of $P_t(t+h) - R_h(t+h)$. On the contrary, Panel (c) shows the case that under-predicts the price and leads to a negative error of $P_t(t+h) - R_l(t+h)$.

1.3.3.4 Errors with respect to potential buy/sell actions

At day t, if we predict that $P_t(t+h)$, the future price at day $t+h$, will be lower than the current daily closing price, then day $t+h$ is potentially a good day for buying. We set the target for a potential buy at $t+h$ to be $P_t(t+h)$. On the other hand, if we predict that the stock price after h days $P_t(t+h)$ will be higher than the current daily closing price $R_c(t)$, then we will consider to set the target for a potential sell at day $t+h$ at the predicted price $P_t(t+h)$. The prediction error with respect to the potential buy or sell is defined as follows:

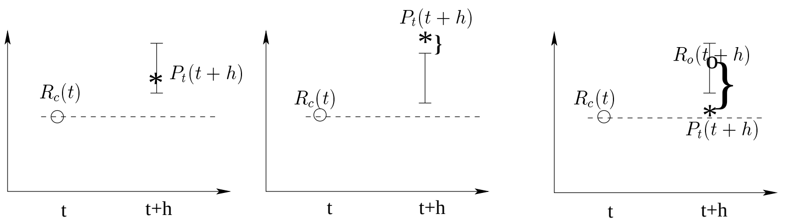
$$G_t(t+h) = \begin{cases} 0 & \text{if } R_l(t+h) \le P_t(t+h) \le R_h(t+h) \\ & \quad \text{and } \ P_t(t+h) < R_c(t) \\ P_t(t+h) - R_o(t+h) & \text{if } P_t(t+h) > R_h(t+h) \ \text{ and } \ P_t(t+h) < R_c(t) \\ P_t(t+h) - R_l(t+h) & \text{if } P_t(t+h) < R_l(t+h) \ \text{ and } \ P_t(t+h) < R_c(t) \\ 0 & \text{if } R_l(t+h) \le P_t(t+h) \le R_h(t+h) \\ & \quad \text{and } \ P_t(t+h) > R_c(t) \\ R_h(t+h) - P_t(t+h) & \text{if } P_t(t+h) > R_h(t+h) \ \text{ and } \ P_t(t+h) > R_c(t) \\ R_o(t+h) - P_t(t+h) & \text{if } P_t(t+h) < R_l(t+h) \ \text{ and } \ P_t(t+h) > R_c(t). \end{cases} \tag{1.24}$$

The error can be explained as follows. When $P_t(t+h) < R_c(t)$, if the predicted price $P_t(t+h)$ happens to fall within the daily low/high range for that day, then the stock can be bought at exactly the predicted price, and $G_t^{buy}(t+h) = 0$ (Figure 1.9a). If the daily high price at $t+h$ is below the predicted price at $t+h$, then we will be able to buy the stock at the opening price, which means that we use less money than expected to buy the stock, and that the error is positive (Figure 1.9b). Last, if the daily low price at $t+h$ is higher than the predicted price at $t+h$, then we will need to pay more than our targeted price to buy the stock, resulting in a negative prediction error of $P_t(t+h) - R_l(t+h)$ (Figure 1.9c).



(a) $G_t(t+h) = 0$ (b) $G_t(t+h) = P_t(t+h) - R_o(t+h)$ (c) $G_t(t+h) = P_t(t+h) - R_l(t+h)$

Figure 1.9: Illustration of prediction errors w.r.t. potential buy signal when $P_t(t+h) < R_c(t)$. The bars represent the daily low/high prices, $*$ stands for the prediction $P_t(t+h)$, and o indicates the opening price.



(a) $G_t(t+h) = 0$ (b) $G_t(t+h) = R_h(t+h) - P_t(t+h)$ (c) $G_t(t+h) = R_o(t+h) - P_t(t+h)$

Figure 1.10: Illustration of prediction errors w.r.t. potential sell signal when $P_t(t+h) > R_c(t)$. The bars represent the daily low/high prices, $*$ stands for the prediction $P_t(t+h)$, and o indicates the opening price.

Similarly, when $P_t(t+h) > R_c(t)$, the errors can be explained in Figure 1.10 as above. If the predicted price falls within the daily low/high range for $t+h$, then the stock can be sold at exactly the predicted price and $G_t^{sell}(t+h) = 0$ (Figure 1.10a). If the daily high price at $t+h$ is below the predicted price $P_t(t+h)$, then we will have to sell the stock for less money than our targeted price, and the prediction error at $t+h$ is negative (Figure 1.10b). Last, if the daily low price at $t+h$ is higher than the predicted price $P_t(t+h)$, then we can sell the stock at the opening price to achieve at least the targeted price, and the error is positive (Figure 1.10c). Our goal in predictions is to limit and minimize negative errors.

1.4 Proposed approaches of time-series predictions

In this section, we describe briefly the work in this thesis that consists of three main parts: preprocessing of time series, neural-network learning using a constrained formulation, and postprocessing for noisy stock time series.

1.4.1 Preprocessing of financial time series

Preprocessing is needed in financial time series in order to remove unpredictable noise. Traditional preprocessing approaches use low-pass filtering to reduce the noise. This approach has been well covered in the literature, and we will show its drawbacks in Chapters 2 and 3.

In our proposed approach, we decompose a time series into several channels using redundant wavelet decomposition. We then apply low-pass filtering to each sub-band when we detect that it has more noise than what can be tolerated in our predictor. The low-pass decomposed channel absorbs the majority of nonstationarity and is then transformed into a near chaotic time series. The low-pass filter for each high-frequency channel can be different according to the property of the individual channel.

1.4.2 Learning and prediction

We propose a new neural-network architecture that combines a recurrent structure and an explicit memory into one unified system. Our new system unifies many existing architectures, such as recurrent networks, FIR networks, time-delayed networks, and others. We also develop a unified epoch-wise backpropagation algorithm for the new architecture.

The key component in this research is the constrained formulation for neural-network learning. In this new learning approach, we propose to treat learning errors for individual patterns as additional constraints and increase the penalties of violated patterns during learning. In this way, we will be able to use an individual pattern's violation level to guide our search. We also propose to model cross-validation errors as constraints and allow multiple cross-validation sets, while at the same time, utilizing all historical data for learning. New constraints available in the future can be integrated easily in our proposed formulation.

In order to solve the constrained formulation, we have developed a new algorithm called violation-guided back-propagation (VGBP) which is based on the Theory of Extended Saddle Points [23, 164, 153, 155]. By transforming the constrained formulation into an l_1-penalty function, we search for extended saddle points of the penalty function. These extended saddle points have been proved to be equivalent to local optimal solutions of the constrained formulation. We generate approximate gradient directions of the penalty function in order to perform descents in the weight subspace of the penalty

function. Periodically, we modify the penalties in order to emphasize those patterns with large violations. To facilitate the convergence of VGBP learning, we also propose the relax-and-tighten strategy.

To address the lag effect caused by low-pass filtering discussed in Chapters 2 and 3, we propose a linked neural network in order to use the information in the lag period to provide active training.

1.5 Benchmarks studied

We list the properties of benchmarks on noiseless/near-noiseless time-series and noisy financial time series studied in this thesis.

1.5.1 Noiseless or near noiseless time-series benchmarks

Table 1.2 lists the noiseless or near noiseless time-series benchmarks studied in this work and their properties on nonlinearity, stationarity, and noise. We have selected these time series because all of them have been well studied in the literature [8, 168, 169, 173] and can be used as benchmarks for meaningful comparisons.

1.5.2 Noisy time series: stock-market time series

The noisy time-series benchmarks studied in this thesis are all financial time series. Ten stocks are selected from a variety of economic sectors as shown in Table 1.3. The daily stock-price data were downloaded from the *Yahoo* Website [177]. We have selected the ten stocks in order to cover broadly different sectors, different capitalization, and different histories. We have also avoided penny stocks in order to ensure a high signal-to-noise ratio.

Table 1.2: Properties of noiseless or near noiseless nonlinear time series studied in this thesis.

Benchmark	Properties		
	Nonlinearity	Stationarity	Noise
Sunspots	Yes	Yes	Small amount
Laser	Yes	Piecewise chaotic	Very little
Mackey-Glass (17)	Yes	Yes	No
Mackey-Glass (30)	Yes	Yes	No
Henon	Yes	Yes	No
Lorentz	Yes	Yes	No
Ikeda	Yes	Yes	No

Table 1.3: Stocks studied in this thesis.

Symbol	Company	Sector	Duration	Price Range ($)
AMR	AMR Inc.	Transportation	01/02/91 to 03/28/03	[2.25, 89.94]
C	Citigroup	Financial	01/02/91 to 03/28/03	[1.43, 53.08]
GE	General Electric	Industrial	01/02/91 to 03/28/03	[3.37, 57.28]
IBM	Int'l Bus. Mach.	Info. Tech.	01/02/91 to 03/28/03	[10.10, 138.36]
MNTR	Mentor	Health Care	01/02/91 to 03/28/03	[1.89, 22.69]
NYT	New York Times	Consumer	01/02/91 to 03/28/03	[7.59, 52.55]
PFGI	Provident Fin. Group	Financial	01/08/92 to 03/28/03	[5.93, 48.45]
PSS	Payless ShoeSource	Consumer	04/18/96 to 03/28/03	[6.67, 26.07]
XOM	Exxon-Mobil	Energy	01/02/91 to 03/28/03	[12.24, 47.09]
YHOO	Yahoo	Info. Tech.	04/12/96 to 03/28/03	[1.29, 250.07]

1.6 Contributions of this thesis

The main contributions of this thesis are as follows.

- *The RFIR architecture and its learning algorithm [159, 158].* Our proposed RFIR architecture combines a recurrent and a memory-based FIR structure. The new architecture provides a more powerful model for time-series predictions because it generalizes traditional recurrent neural networks (RNN), nonlinear autoregressive networks with exogenous inputs (NARX), time-delayed neural networks (TDNN), and FIR neural networks. Our single architecture contains all those aforementioned structures as special cases and provides the flexibility of selecting a desired architecture according to problem-specific needs.

- *A new constrained formulation for neural-network learning [156, 157, 158].* To improve the quality during learning while minimizing normalized mean squared errors, we have incorporated a variety of constraints into our neural-network models in order to meet additional learning requirements. Our constrained formulation is flexible in including more than one necessary criteria in learning and can incorporate helpful prior knowledge. The constrained formulation not only can be used in neural-network learning for time-series predictions, but also can be used in neural-network learning for pattern classification [156].

- *A new cross-validation method on multiple validation sets [157, 158].* Our approach is more general than previous approaches that allow only one cross validation set at a time. It can handle piecewise chaotic time series effectively and does not require a certain portion of historical data to be reserved for validation. Hence, it provides the most effective learning by using all available patterns for learning.

- *A violation-guided back-propagation algorithm (VGBP) [158, 159].* Based on the theory of extended saddle points for continuous constrained optimization, we have developed an efficient algorithm that guides the search more effectively in our constrained formulation. Moreover, it provides fast convergence using a *relax-and-tighten* strategy, which exploits certain convergence behavior during the search process.

- *A Systematic study of edge effects in low-pass filtering of noisy time series [161].* We have identified some issues in existing approaches, such as flat extension and mirror extension, that impose some restricted assumptions on future data. We have further proposed new approaches to alleviate the issues.

- *An approach that incorporates constraints on predicted low-pass data in the lag period for financial time series prediction.* These new constraints enable active training in the lag period, which greatly improves our prediction accuracy in the lag period [161, 160].

- *Decomposition of noisy financial time series and channel-specific preprocessing to improve predictability.* The decomposed low-frequency channel exhibits mainly nonstationarity, and high-frequency channels are close to chaotic time series but contaminated by noise. A special transformation is applied to the low-frequency channel to convert it into a near chaotic time series. For high-frequency channels, since different channels may have a different level of noise, we have studied a channel-specific adaptive denoising method to remove high-frequency noise while incurring as short a lag as possible.

1.7 Outline of this Thesis

This thesis is organized as follows. In Chapter 2, we survey existing work on time-series models. We classify time-series models based on their assumptions used and point out the issues in existing models when applied to predict a general nonlinear noisy nonstationary time series. Among existing models, we are especially interested in neural-network models and provide a more detailed survey of existing neural-network models.

Chapter 3 addresses preprocessing for noisy time series. We first study trade-offs between denoising and information loss in traditional low-pass filtering approaches. We then present our new preprocessing approach that combines wavelet decomposition, transforming a nonstationary series into a near chaotic series, and channel-specific denoising.

In Chapter 4, we present our new recurrent FIR architecture (RFIR), along with our generalized epochwise backpropagation-through-time algorithm. We also show how this new architecture unifies a class of existing neural-network architectures. Based on this architecture, we introduce a new neural-network learning model with a unique constrained formulation. This is different from traditional models that use an unconstrained formulation. We introduce new constraints to improve learning under multiple objectives, which is not possible in a traditional unconstrained formulation with a single objective. To solve the constrained formulation, we have developed an efficient learning algorithm called violation-guided backpropagation, using information on violations from individual patterns to guide the search in order to find improved solutions. We further apply the relax-and-tighten strategy in order to accelerate search convergence. At the end of the chapter, we present the study on the effect of parameter selection in the learning algorithm.

In Chapter 5, we apply our constrained formulation and our violation-guided back-propagation algorithm to solve applications in noiseless or near noiseless nonlinear time series. Significant improvements over previous work have been achieved on all the benchmarks tested.

We dedicate Chapter 6 to address noisy financial time series. We compare the prediction performance between our new preprocessing approach and traditional low-pass preprocessing. We also compare the prediction performance between our ANN model and the traditional autoregressive model.

Finally, in Chapter 7, we conclude this thesis and point out future research directions.

Chapter 2

Previous Work

In this chapter, we review existing work on time-series predictions discussed in Chapter 1. We examine how different models are used in modeling time series of different characteristics. We also examine how certain time series can be transformed to another that can be solved by existing models. We further point out issues in existing models after each review.

2.1 Previous work on predicting chaotic time-series

As stated in Chapter 1, we are not interested in identifying of chaotic behavior (finding embedded dimension and embedded delay) or chaotic control. We are only interested in modeling and predicting general nonlinear chaotic time series.

A variety of time-series models have been proposed and studied in the last four decades. In this section, we review briefly some existing models and present their potential issues when applied to general nonlinear chaotic time series. Unless otherwise explicitly specified, we call a nonlinear chaotic time series simply as a nonlinear time series. We classify existing time-series models into linear and nonlinear [22] (Figure 2.1), based on whether certain linearity assumptions are used or not. Linear models are further divided into three sub-classes that include the ARMA model and its variants [16], exponential smoothing, and state space models [7, 22]. Last, nonlinear models are classified into assumption-based and learning-based models.

2.1.1 Linear models

Linear models work well for linear time series but may fail otherwise. Those models generally assume that the time series under consideration is generated by a linear process in which the value of a future observation is linearly dependent on some historical observations. There are three types of linear models that are widely used: ARMA models, exponential smoothing, and state-space models. One chooses to apply linear models to nonlinear time series because linear models are easy to use and understand, and their results are easy to interpret. Moreover, in many cases a linear process can be a good approximation to a nonlinear process in most local regions.

2.1.1.1 ARMA models and their variations

ARMA models [16, 17, 65] and their variations, such as autoregression (AR) and moving average (MA), are the most well-studied models in the literature. An ARMA model describes future data as a linear combination of some historical data and a random process as follows.

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \cdots + \alpha_p X_{t-p} + Z_t + \beta_1 Z_{t-1} + \cdots + \beta_q Z_{t-q}, \tag{2.1}$$

where X_t is the observed time-series process; Z_t is a purely random process with a mean of 0 and a variance of σ_Z^2; p is the order of autoregression; and q is the order of the moving average. When $p \neq 0$ and $q = 0$, the ARMA model becomes an autoregressive

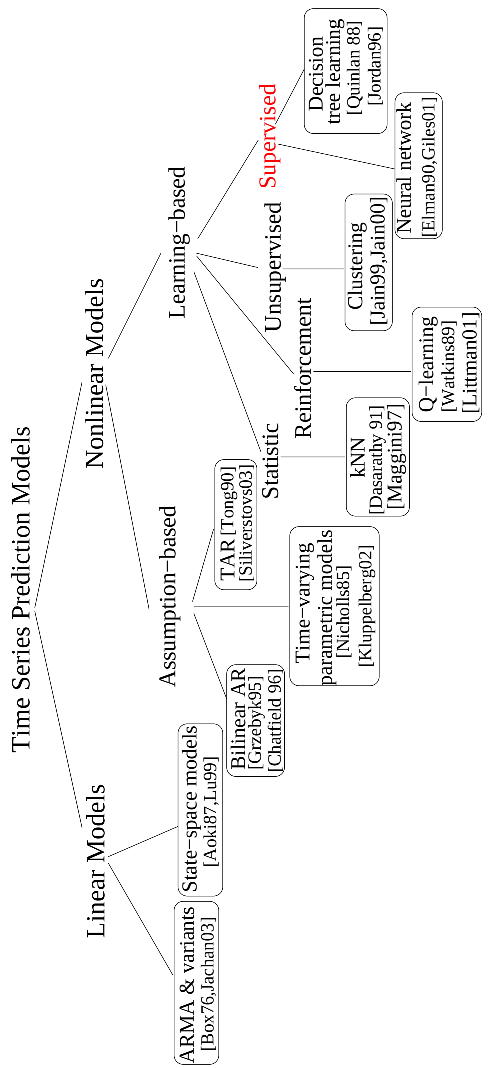


Figure 2.1: A classification of linear and nonlinear time-series models.

model ($AR(p)$); when $p = 0$ and $q \neq 0$, it becomes the moving average model ($MA(q)$). (2.1) is normally denoted as $ARMA(p, q)$.

Define backward shift operator B as:

$$B^k X_t = X_{t-k}, \qquad \text{for all } t. \tag{2.2}$$

In a linear model, (2.1) can be expressed as:

$$\phi(B)X_t = \theta(B)Z_t, \tag{2.3}$$

where

$$\phi(B) = 1 - \alpha_1 B - \alpha_2 B^2 - \cdots - \alpha_p B^p, \tag{2.4}$$

and

$$\theta(B) = 1 + \beta_1 B + \beta_2 B^2 + \cdots + \beta_q B^q. \tag{2.5}$$

For an AR process to be stationary, the roots of the equation

$$\phi(B) = 1 - \alpha_1 B - \alpha_2 B^2 - \cdots - \alpha_p B^p = 0 \tag{2.6}$$

must lie outside the unit circle, as shown by Box and Jenkins [15]. Further, for an MA process to be invertible (which ensures that there is a unique MA process for a given set of autocorrelation functions), the roots of the equation

$$\theta(B) = 1 + \beta_1 B + \beta_2 B^2 + \cdots + \beta_q B^q = 0 \tag{2.7}$$

must lie outside the unit circle as well [15].

To fit an $AR(p)$, one has to decide on the order of the model (namely, the value of p), and the algorithm to estimate the α's. Although there is no theoretical guideline on how to choose p, p is chosen empirically based on how fast the autocorrelations of the time series decrease towards zero [15, 21]. To estimate the α's, a least-square fit can be

applied. Many statistical software packages provide AR models, such as *TISEAN* [58] and *Financial time-series Toolbox* for *Matlab* [98, 97].

To fit an $MA(q)$ process, one also needs to consider the order of the model and the estimation algorithm. Unlike estimating p in $AR(p)$, the choice on q is relatively clear in $MA(q)$ because the theoretical ACF (autocorrelation function) of an $MA(q)$ process has a very clear "cut-off" at lag q [21, 22]. However, the estimation of the β's is much more difficult because it is based on minimizing $\sum Z_t^2$ [21]. In practice, an iterative optimization procedure may need to be employed in order to find the optimal β's [15, 21] when solving this quadratic optimization problem.

The estimation of parameters of an ARMA model is similar to that of a MA process in the sense that an iterative optimization procedure has to be applied in order to find the optimal α's and β's that minimize $\sum Z_t^2$ [126, 72].

In practice, most time series are not stationary, which means that ARMA models cannot be applied directly. However, some nonstationary time series can be transformed into stationarity ones by simple differencing:

$$Y_t = \nabla^d X_t = (1 - B)^d X_t. \tag{2.8}$$

Then the ARMA models can be applied to the differenced series Y_t as follows:

$$\phi(B)Y_t = \theta(B)Z_t, \tag{2.9}$$

or

$$\phi(B)(1 - B)^d X_t = \theta(B)Z_t. \tag{2.10}$$

Eq. (2.10) is called an *autoregressive integrated moving average process* (ARIMA) [15, 21]. The parameter estimation of ARIMA models are the same as that of ARMA models once differencing is performed.

2.1.1.2 Exponential smoothing

Exponential smoothing [18, 42] models estimate X_{t+1} as a linear combination of current observation X_t and the previous estimate for X_t. Denote the estimate for X_{t+1} as $\hat{X}_{t+1}$. Exponential smoothing can be expressed as:

$$\hat{X}_{t+1} = \alpha X_t + (1-\alpha)\hat{X}_t, \qquad 0 < \alpha \leq 1, \tag{2.11}$$

where α is the single parameter in the model. The reason it is called exponential smoothing is because (2.12) can be rewritten as:

$$\hat{X}_{t+1} = \alpha X_t + \alpha(1-\alpha)X_{t-1} + \alpha(1-\alpha)^2 X_{t-2} + \cdots \tag{2.12}$$

The error of the estimation is:

$$e_{t+1} = X_{t+1} - \hat{X}_{t+1}. \tag{2.13}$$

Estimating parameter α requires finding α that minimizes $\sum_{i \leq t} e_i^2$.

The exponential smoothing model described by (2.12) is very simple. However, it cannot be applied to time series exhibiting trends and seasonality. In the literature, a lot of extensions have been made in order to handle linear trends and seasonality, such as double exponential smoothing and the Holt-Winters forecasting procedure [42, 43, 20, 21, 19]. In these extensions, both local trends and seasonality are modeled using exponential smoothing in forms similar to (2.12). However all these ad hoc extensions require some linear property in both the local trend and seasonality and, therefore, are not suitable to deal with nonlinear trends, irregular seasonality, and other forms of nonlinearity.

2.1.1.3 State-space models

State-space models [7, 115, 53, 84] represent a time series using two sets of linear equations called *observation* equation and *transition* equation. For a univariate time series $\{X_t\}$, the observation equation is:

$$X_t = h_t^T \theta_t + n_t, \tag{2.14}$$

where θ_t and h_t are $m \times 1$ state vectors, and n_t denotes the observation error at time t. The state vector evolves over time and is depicted by the transition equation as follows:

$$\theta_t = G_t \theta_{t-1} + w_t, \tag{2.15}$$

where G_t is an $m \times m$ transition matrix, and w_t is an $m \times 1$ vector of transition errors. Since both (2.14) and (2.15) are linear, state-space models belong to the class of linear models.

In state-space models, both h_t and G_t are assumed to be known *a priori*. In practice, a variety of tools, including external knowledge, are used to help choose appropriate h_t and G_t [53, 21]. Another difficulty in using state-space model is that the error variance of n_t and w_t are not known *a priori*. This means that these parameters have to be updated constantly.

After h_t, G_t, and the variances for n_t and w_t are chosen, we need to estimate state vector θ_t. A Kalman filter is generally used for estimating the state vector [101, 102, 4, 7, 53] in two stages: the *prediction stage* and the *updating stage*. A Kalman filter first predicts the next state based on (2.15) with w_t set to zero. When the new data is available, it computes the prediction error and updates the state vector as well as other related parameters.

In short, linear models work well for many processes that exhibit linear properties, but do not work well for processes exhibiting nonlinear properties, such as financial-market time series. In those cases, linear models may fail to give accurate predictions when the linearity assumption does not hold [40].

2.1.2 Models for nonlinear stationary time series

Nonlinear models can be classified into assumption-based models and learning-based models. Assumption-based models assume that a time series can be described by a specific nonlinear function chosen *a priori*, and that historical data is used only to estimate the parameters in the selected nonlinear function. On the other hand, learning-based models do not assume any specific function. Instead, they employ certain training algorithms using historical data in order to learn the underlying dynamics through the training process. Historical data in learning-based models plays a role of teaching the model on how to respond to different inputs.

2.1.2.1 Assumption-based models

In assumption-based models, a specific form of nonlinear function is pre-selected for fitting the historical data, before parameter estimation takes place. Models falling into this class include *nonlinear autoregressive models* [116, 74, 21], *bilinear models* [49, 131, 127, 50], *threshold autoregressive models* (TAR) [150, 149, 148, 142], and the *ARCH* and *GARCH* models [13, 54, 147].

Nonlinear autoregressive models (abbreviated as NLAR) take a form of:

$$X_t = f(X_{t-1}, X_{t-2}, \cdots, X_{t-p}) + Z_t, \tag{2.16}$$

where $f()$ is a nonlinear function, p is the order of the model, and Z_t is a pure random process. The NLAR with an order of p is generally denoted as $NLAR(p)$. Obviously, $AR(p)$ is a special case of NLAR(p) in which $f()$ is a linear function. A more general form of NLAR incorporates Z_t inside the $f()$ function. Before using NLAR, one has to decide on the specific $f()$ to be used and may require domain-specific knowledge.

Bilinear models are considered to be nonlinear extensions of ARMA models. This class of models include product terms of lagged values of time-series $\{X_t\}$ and the random process $\{Z_t\}$. For example, a simple bilinear model can be:

$$X_t = \alpha X_{t-1} + \beta X_{t-1} Z_{t-1} + Z_t, \tag{2.17}$$

where both α and β are constants. Bilinear models can usually provide a good fit for data but generally do not give good long-term forecasts [127, 29, 22]. They are also "not particularly helpful in providing insight into the underlying generating mechanism," as other assumption-based nonlinear models do [21]. In practice, bilinear models are relatively less frequently used.

Threshold autoregressive models express a process in a piecewise linear AR model. For example,

$$X_t = \begin{cases} \alpha_1^1 X_{t-1} + \alpha_2^1 X_{t-2} + \cdots + \alpha_{p_1}^1 X_{t-p_1} & \text{if } X_{t-1} > r \\ \alpha_1^2 X_{t-1} + \alpha_2^2 X_{t-2} + \cdots + \alpha_{p_2}^2 X_{t-p_2} & \text{if } X_{t-1} \le r, \end{cases} \tag{2.18}$$

where $\{\alpha^1\} = [\alpha_1^1, \alpha_2^1, \cdots, \alpha_{p_1}^1]^T$ and $\{\alpha^2\} = [\alpha_1^2, \alpha_2^2, \cdots, \alpha_{p_1}^2]^T$ are constant vectors, and r is a threshold. Note that a transition from one linear AR model to another is not smooth. Later, smooth threshold autoregressive models (STAR) were introduced in order to provide a smooth and continuous transition between two linear AR models [148]. TAR and STAR models had some success in its early development stage, but were found to have little improvement in forecasting later. TAR models usually need great insights into the time-series process, such as the proper choice of r [149, 21].

ARCH models are formally called *autoregressive conditionally heteroscedastic models*. Unlike the three classes of nonlinear models discussed above, ARCH models are primarily used for modeling changes of variance (or volatility) of a series. The series modeled (denoted as $\{Y_t\}$) by ARCH models may be a differenced series or a series of residuals from an autoregressive model. An ARCH model can be depicted as follows:

$$Y_t = \sigma_t \epsilon_t, \tag{2.19}$$

where $\{\epsilon_t\}$ represents a sequence of identical and independent distributed (i.i.d.) random variables with zero mean and unit variance, and σ_t is the local conditional standard deviation of the process that takes the following form:

$$\sigma_t = \gamma + \delta_1 y_{t-1}^2 + \delta_2 y_{t-2}^2 + \cdots + \delta_p y_{t-p}^2. \tag{2.20}$$

Here y_{t-j} is the observed value of series Y_t at time $t-j$, p is the order of the ARCH model, and δ_i and γ are constant parameters similar to the parameters in $MA(p)$. Eq (2.20) is normally referred to as the $ARCH(p)$ model. ARCH models only depend on past values of the series. The *generalized autoregressive conditionally heteroscedastic* (GARCH) model of order p and q (denoted as $GARCH(p,q)$) is given by:

$$\sigma_t = \gamma + \delta_1 y_{t-1}^2 + \delta_2 y_{t-2}^2 + \cdots + \delta_p y_{t-p}^2 + \beta_1 \sigma_{t-1}^2 + \beta_q \sigma_{t-2}^2 + \cdots + \beta_q \sigma_{t-q}^2. \tag{2.21}$$

The $GARCH(p,q)$ model depends not only on previous values of the series $\{y_t\}$ but also on previous values of the variances $\{\sigma_t^2\}$. Here β_i are parameters similar to parameters in $AR(q)$. The ARCH/GARCH models are not focused on forecasting future values of a series; instead, they estimate changes of variance (volatility) of a series over time, which

sometimes is very important, especially in financial-market series (e.g., risk management in portfolio management). In general, choosing between different ARCH/GARCH models is difficult, especially when domain knowledge is absent. In our work, since we emphasize on forecasting a time series instead of the estimation of its variances, we do not study ARCH/GARCH models.

Assumption-based model are effective if one has *a priori* knowledge about the behavior of the time series under investigation. When domain knowledge is absent or when the domain is too complex to extract certain useful information, these models are very hard to use.

2.1.2.2 Learning-based models

Machine learning can handle nonlinear time series because it learns a nonlinear model without assuming the specific form of nonlinearity. Learning-based methods that can model a time series include statistic learning (such as k-nearest-neighbor (kNN) [26, 33, 27, 106, 87]), reinforcement learning (such as Q-learning [170, 172, 136, 71, 83]), unsupervised learning (such as clustering methods [37, 38, 118, 67, 110, 66]), and supervised learning (such as decision trees [128, 104, 69] and artificial neural networks (ANNs) [132, 133, 36, 57, 46]).

In a k-nearest neighbor method, the distances between historical data sequences with a length of k and the most recent data sequence of the same length are computed. The historical sequence with the shortest distance to the current sequence is considered a match, and future values are forecast by looking at the "future values" following the matched historical sequence. The main task in kNN involves the identification of the distance between vector $x = (x_1, x_2, \cdots, x_k)^T$ and vector $y = (y_1, y_2, \cdots, y_k)^T$, which is generally defined by the square root of the weighted squared differences:

$$d(x, y) = \sqrt{\sum_{i=1}^{k} w_i (y_i - x_i)^2}, \qquad (2.22)$$

where w_i's are the weights to be determined through learning.

In reinforcement learning, there are a set of actions and states. By penalizing or by rewarding each action under a state, reinforcement learning tries to maximize the rewards achieved in the given training set. It has been applied in robot planning and trading rule/strategy development in financial markets.

Clustering methods group time-series data into clusters. Data patterns inside a cluster are considered to be similar, and data patterns in different clusters are considered to be dissimilar. After clustering, data inside a cluster can be trained using an appropriate model (including linear models) to simplify learning. Clustering methods are useful in modeling epoch-wise chaotic time series discussed later.

Artificial neural networks are mainly supervised learning models that are widely applied in time-series modeling. Since they are the basis for our proposed model, we review them separately in Section 2.6.

Learning-based methods learn a model of the expected outputs when given the input and the current state (if any) of the model. In general, a learning-based model for a given learning set uses a single nonlinear objective and does not use individual patterns to help escape from local optima [140, 156]. Since machine learning problem is a complex nonlinear optimization problem with many local minima, any search algorithm may get stuck in sub-optimal solutions. With a single objective in traditional machine learning approaches, one has no way to control how well individual patterns can be trained, and there is no way to incorporate domain knowledge in learning. In this research, we propose

a constrained formulation for ANN learning that adds constraints on individual patterns and that uses violated constraints to help guide learning. Such formulations are general and can be applied to other learning methods.

2.2 Previous work on seasonal chaotic time series

Many time series, especially economic time series, exhibit seasonality. When seasonality is present in a time series, it is always helpful to remove the seasonality before modeling the time series. Here we review existing work on seasonality detection and seasonality elimination. We only consider additive seasonality, as multiplicative seasonality can be easily transformed into additive seasonality by a logarithmic transformation.

2.2.1 Seasonality detection

In practice, correlograms and spectral plots can be employed to detect seasonality. A correlogram is a plot of autocorrelation of a sequence of time series [22, 41]. If a seasonality with period L is present in a time series, then there will be a large jump of autocorrelation at lag L in the correlogram. For example, Figure 2.2 gives the correlogram of the time series in Figure 1.2. Peaks are clearly displayed at lags of 10, 20, and 30. This correlogram strongly indicates that there is a seasonality with period of 10 embedded in the time series. A spectrum can also be used to detect seasonality [96]. A spectrum shows how much each frequency contributes to the whole time series. The frequency that corresponds to the period of seasonality in the time domain will stand out from its neighboring frequencies in the spectrum.

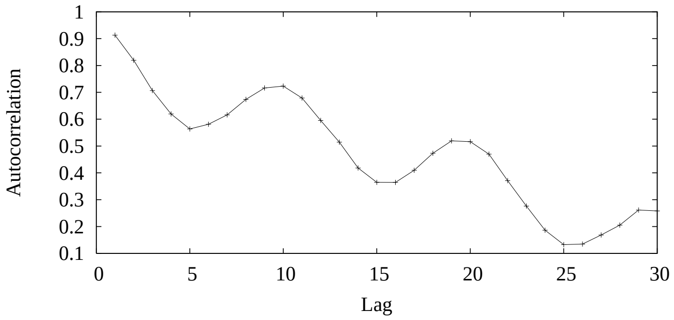


Figure 2.2: Correlogram of the time series in Figure 1.2 that displays seasonality. The correlogram shows the autocorrelation peaks at lags of 10, 20, and 30, respectively.

2.2.2 Seasonality Elimination

When the period of seasonality is detected, the simplest way to remove the seasonality is to difference the original time series [15, 91, 92] at a distance of the detected periodicity. Suppose the period of seasonality detected is L. In their *seasonal autoregressive moving average* model (SARMA), Box and Jenkins applied L-order differencing on the original time series X_t first before the ARMA model was applied. Define the L-order differencing operation as follows:

$$\Delta_L X_t = X_t - X_{t-L}. \tag{2.23}$$

The ARMA model expressed in (2.3) can be extended to the *seasonal autoregressive moving average* (SARMA) model as follows:

$$\phi(B)\Delta_L X_t = \theta(B)Z_t. \tag{2.24}$$

The following procedure can be applied to decompose a time series into its seasonal and nonseasonal components [91, 92]. First the time series of the moving average with a window of L is applied to the original series. In the resulting moving averages, the seasonal component is removed. Then the seasonal component can be obtained by subtracting the series of moving averages from the original series.

2.2.3 Prediction

After modeling a preprocessed time series, one can predict the future values of the preprocessed series by training a prediction model. The prediction of the original time series can be obtained by adding the predicted values for the preprocessed (differenced) series to the original time-series values L steps earlier.

2.3 Previous work on piecewise chaotic time series

One may be able to find many models for chaotic time series, but there is relatively few work on piecewise chaotic time series. The main difficulty is that it is hard to figure out which chaotic behavior the time series is exhibiting.

2.3.1 Regime transition detection

The detection of regime transition is an approximation at best since there are no clear boundaries between regimes. Similar to the detection of seasonality, the detection of a regime switch is mainly done graphically and, therefore, very subjective. In practice, one detects visually whether different segments behave differently in a time series. If the plot shows plausible transitions between segments, then a moving mean and variance are computed to see if there are regions where the mean and the variance behave significantly differently from those before and after.

2.3.2 Models for piecewise chaotic time series

Time series with piecewise chaotic regimes has been studied extensively. There are two categories of models for piecewise chaotic time series. Models in the first category try to identify regime transition first, before applying an appropriate model specific for that regime. Models in the second category perform transition identification and model learning together. We classify models for piecewise chaotic time series in Figure 2.3. In the classification, the branch for machine learning is the same as that in Figure 2.1, as most existing approaches, except for Wan [169], just leave the learning of regime shifts to a learning-based model without an explicit mechanism to handle these shifts.

2.3.2.1 Regime-identification based models

Models using this approach include regime switching models [32, 109, 75, 45] and hidden Markov models [130, 129, 76, 70].

In a regime-switching model, more than one time-series models are trained beforehand using historical data. At prediction time, some strategies are used to determine which pre-trained model should be used, or a new model should be trained in case all pre-trained models do not work well. These strategies are based on either statistical or performance measures, where a statistical measure attempts to identify changes in the data distribution by certain statistical techniques, such as the χ^2 test and the K-S tests [32]; and a performance measure evaluates the recent prediction accuracy for dif-

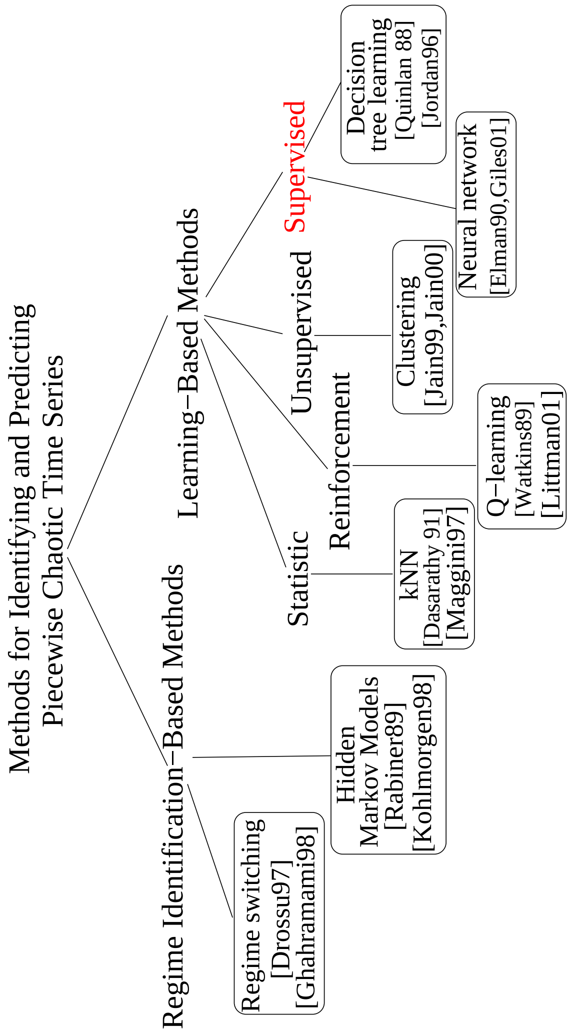


Figure 2.3: A classification of time-series models for handling piecewise stationarity.

ferent models. If all existing models in use do not give satisfactory performance, then a new model is launched [32, 109, 75].

Hidden Markov models are actually a special class of regime switching models. Since these models are given heavy consideration in the literature, we list them as separate models from other regime switching models. In hidden Markov models, n hidden Markov models are selected; the probability of transition from the current model to another model is computed; and the model that maximizes the probability of the observed sequence is selected [76, 118]. Clustering methods can be employed to determine the number of hidden Markov models to be used. These regime-identification based models are limited because they do not work well unless the change-over points can be identified correctly. Moreover, the prediction of change-over points may be as hard as the prediction problem itself.

2.3.2.2 Learning-based models

Without separating the entire learning process into regime identification and intra-regime modeling, machine learning try to learn regime transitions by reserving patterns in each regime change to be verified in a cross-validation set. However, traditional learning approaches using a single objective may have difficulties in handling cross validations for multiple regime changes. This happens because the single objective containing the sum of the errors in all the cross-validation sets does not provide guidance in learning, and the model has no indication on which transition regions are not sufficiently trained. To address this issue, a constrained formulation can be used to constrain the error in each validation set to be satisfied during learning. We show such an approach in Chapter

5.1.2 when ANN learning is applied to predict the laser time series and its successful prediction of regime changes in testing.

The previous work on learning-based models has already been covered in Section 2.1.2.2 and is not repeated here.

2.4 Previous work on handling nonstationarity

Almost all methods of time-series analysis, linear or nonlinear, require some kind of stationarity. In most literature, it emphasizes on how stationarity can be established; and if nonstationarity is detected, "often the time series was discarded as unsuitable for a detailed analysis, or it was split into segments that were short enough to be regarded as stationary" [138] or it was differenced once to twice to remove trends [96].

The differencing methods assume that a linear trend is mainly the cause of nonstationarity, which is not the case for general nonstationary time series. Also, when performing predictions on the original nonstationary time series, one needs to undo differencing. Undoing differencing may introduce severe instability as the prediction accuracy on a long horizon relies on the prediction accuracy on its component horizons [96].

Another type of methods is to split the original time series into local stationary segments [126, 127, 152] before performing time-series analysis. To identify the nonstationarity, one first estimates certain parameters (such as mean and variance) using different segments (windows) of the time series under study. If the observed variations on the estimated parameters are found to be significant, then the time series within that segment is considered to be nonstationary. In this case, the local stationary window should be smaller than the window used. To apply a stationary time-series model, one has to study the time series within each local stationary window. An implicit assumption used in the methods based on local stationarity is that the time series studied must exhibit local stationary. The assumption may not be true in general nonstationary time series. Also, for models based on local stationarity to work well, one has to be able to identify local stationary windows accurately. Further, the pattern at the end of current local stationary window has to either appear in previous local stationary windows (which are included in the training data) or appear in current local stationary window. On the other hand, one should not include too many irrelevant local stationary windows in the training set, which may distract learning. Because of those requirements, one has to conduct additional work to make sure appropriate local stationary windows are included in the training set.

In general, the general nonstationarity problem cannot be solved without knowing mathematic equations governing the nonstationarity [96]. In our work, we are only interested in nonstationarity presented in financial time series. For such time series, it is possible to apply a transformation-based method to convert a nonstationary low-pass time series into a chaotic time series in Chapter 3.

2.5 Previous work on handling noisy time series

Random noise is uncorrelated, has zero mean, and is not predictable due to its uncorrelated nature. As its presence in a time series distracts a model from learning useful and clean information, especially when the signal-to-noise ratio is relatively low, it is necessary to eliminate such noise before learning. Although noise can be present in different frequency channels, we are only interested in time series with high frequency noise in this thesis.

2.5.1 High frequency noise detection

To detect high frequency noise, one can first decompose signals into a low-frequency channel and a high-frequency channel using FFT. The cut-off frequency for the high-frequency channel is the parameter to be found in the following detection procedure [95, 96].

Step 1 Select a low enough cut-off frequency f_0 for time series $\{X\}$.

Step 2 Obtain frequency space coefficients $\{Y\}$.

Step 3 Perform *Kolmogorov-Smirnov* (KS) test to determine whether the coefficients corresponding to frequencies higher than f_0 are evenly distributed in frequency band $[f_0, 1]$.

Step 4 If KS test succeeds, then there is high frequency noise above cut-off frequency f_0; otherwise, increase f_0 and go back to Step 2. If f_0 is close enough to 1, then there is no high frequency noise.

In practice, it is up to a user to determine at which level a KS test is to be accepted or rejected. We can apply this procedure to the financial time series studied in this thesis. Figure 3.7 plots spectra for ten stock-price time series studied in our work. It is clear that the spectra have relatively small magnitudes and are evenly distributed when frequency $f > 0.15$. So we can consider frequency above 0.15 mainly noise. For frequencies between 0.05 and 0.15, it all depends on the confidence level used in the procedure in order to determine if they are evenly distributed. Because of those observations, we will use filters with a cut-off frequency between 0.05 and 0.15 for financial time series preprocessing in Chapter 3.

2.5.2 Preprocessing: denoising

2.5.2.1 Low-pass filtering

In the literature, de-noising is usually done by low-pass filtering or wavelet transforms [96, 179]. Figure 2.4 illustrates the use of a symmetric FIR filter to generate de-noised data $S(t)$:

$$S(t) = \sum_{j=-L}^{L} R(t+j)g(j), \tag{2.25}$$

where $g(j)$ is the j^{th} filter coefficient, $2\,L$ is the number of filter taps, and $R(t)$ is the raw data in the noisy time-series.

A symmetric FIR filter is a non-causal filter because its current filtered output depends on future inputs. For example, the filtered output of a $2L$-tap symmetric filter ends at $t_0 - L$ because it depends on raw data that ends at t_0. Such dependencies on future data lead to lags (sometimes called edge effects) in the filtered data. Figure 2.5 shows a 10-day lag in both the low-pass and high-pass data of the closing stock prices of IBM, when filtered by a 20-tap symmetric FIR filter.

2.5.2.2 Handling edge effects (flat/mirror/zero-padding extension)

An edge effect is not a unique artifact of non-causal filters but also occurs when causal filters are used. Although the outputs of causal filters do not depend on future inputs, they reflect a delayed behavior of the original time series and amount to a lag similar to that in non-causal filters.

To overcome edge effects in a time series, a predictor has to first predict missing filtered data in the lag period before predicting into the future. In a time series with

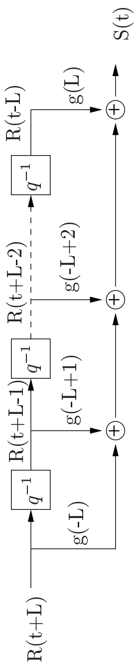


Figure 2.4: A symmetric FIR filter with $2L$ taps. q^{-1} is a delay operator which transforms $R(t+i)$ to R(t+i-1).

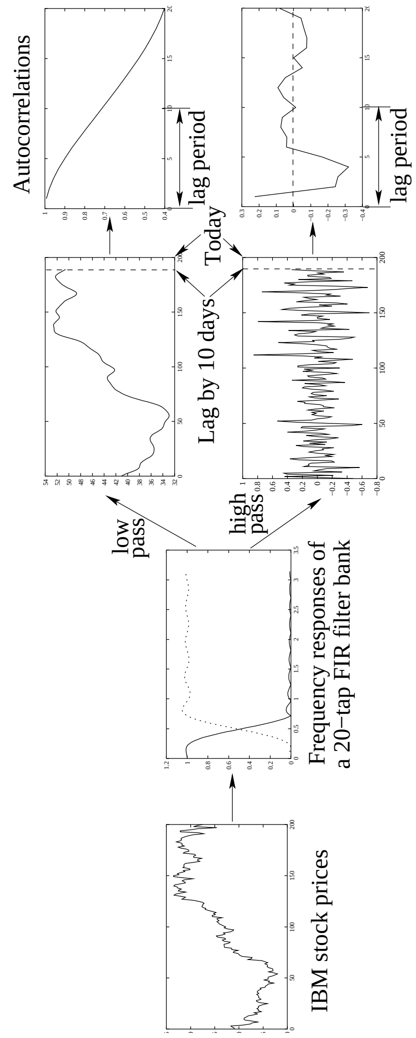


Figure 2.5: An illustration of a filtering process on a time series of noisy IBM daily closing prices using a 20-tap symmetric low-pass FIR filter to de-noise the time series. Both the low-pass and high-pass data have a 10-day lag. The right two panels show the autocorrelation plots for both filtered time series.

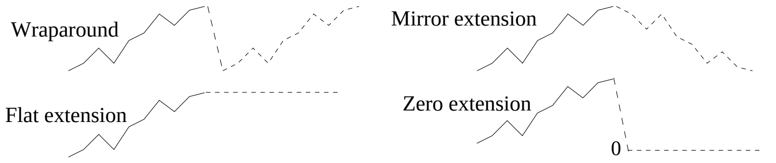


Figure 2.6: Four techniques for handling edge effects in order to compensate for missing data in the lag period. Solid lines represent actual raw data, and dashed lines stand for extended data.

high-frequency random noise, such predictions will be limited to those of low-pass data, as the auto-correlations between high-frequency samples at distances longer than the lag period will be low (Figure 2.5). As a result, we focus on the predictions of low-pass data in this chapter.

Existing approaches on predicting missing low-pass data in a lag period typically impose some assumptions on the future raw data. Figure 2.6 shows four example approaches [96, 111, 178]. Here, a flat extension assumes that future raw data $R(t_0 + j)$, $j = 1, 2, \ldots$, is the same as the latest observed raw data $R(t_0)$; a mirror extension assumes that future raw data is a mirror image of history data, that is, $R(t_0 + j) = R(t_0 - j + 1)$; a wraparound assumes that after a period of T, the time series repeats itself; and a zero extension assumes future data $R(t_0 + j)$ to be zero. Using the extended raw data, low-pass filtering is then applied to obtain the de-noised data in the lag period. In our work, we do not consider wraparound and zero extension, as they are applicable only when the time series is stationary and has zero mean.

Figure 2.7 shows the mean absolute errors between the true low-pass data of the closing stock prices of IBM and its corresponding predicted low-pass data using flat and

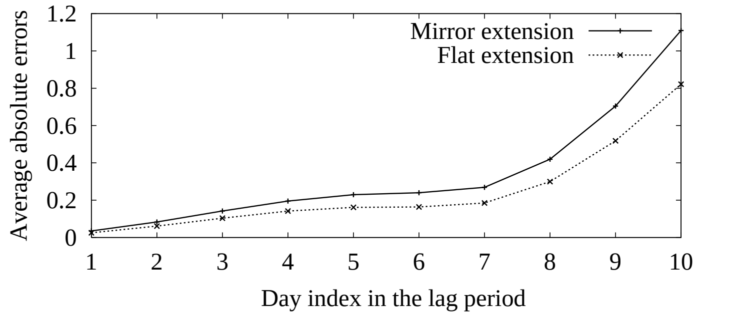


Figure 2.7: The average absolute errors diverge quickly when predicting missing low-pass data in the lag period of ten days.

mirror extensions. Although flat extension performs slightly better than mirror extension in this case, both show that the average errors of low-pass data in the last three days are considerably larger than those in the rest of the lag period. As the low-pass values in the first seven days of the lag period are quite accurate, they can be used as training patterns as if they were true low-pass values. In our approach described in Chapter 4, we design a special architecture for lag-period prediction and predict patterns in the latter part of the lag period and beyond. We further use constraints on the raw data in the lag period in order to have more accurate predictions in the lag period.

2.6 Artificial Neural Networks

It is well-known that artificial neural networks (ANNs) are universal function approximators, and that they do not require *a priori* knowledge on the process under consideration. ANNs are also well-known for their ability to model nonlinear systems. As these properties are exactly what people are looking for in modeling nonlinear time series with unknown or very complex nature, ANNs are becoming more attractive as tools for time-series predictions.

The ANNs studied in this thesis are mainly multilayer perceptrons (MLP), including time-delayed neural networks (TDNN), FIR neural networks (FIR-NN), recurrent neural networks (RNN), and simple feedforward networks [56]. MLPs and radial-basis-function neural network (RBF) [125, 124] are arguably two of the most popular types of neural networks that have been shown to be *universal approximators* [62, 124].

2.6.1 ANN architectures for time-series modeling

ANNs for modeling time series generally have special structures that store temporal information either explicitly using time-delayed structures or implicitly using feedback structures. Examples of the first class include time-delayed neural networks (see Figure 2.8) (TDNN) [80, 166, 165, 135] and FIR neural networks (FIR-NN) [168, 169], whereas examples of the latter include recurrent neural networks (RNN, see Figure 2.9) [121, 34, 175, 139]. Other architectures, such as radial-basis-function networks (RBF) [105, 113, 112, 63] and support vector machines [68, 108], store approximate history information in either radial-basis functions or the so-called support vectors.

In a time-delayed neural network, historical information is stored in its input layer explicitly. In practice, it is hard to determine how long this memory should be. An FIR neural network is a feedforward network just like TDNN, but differs in its connections. In a TDNN, the connection between two nodes is always a single connection with a single weight; whereas in an FIR network (Figure 4.1), the connection is modeled by a finite impulse-response (FIR) filter that may contain more than one weights. Both TDNN and

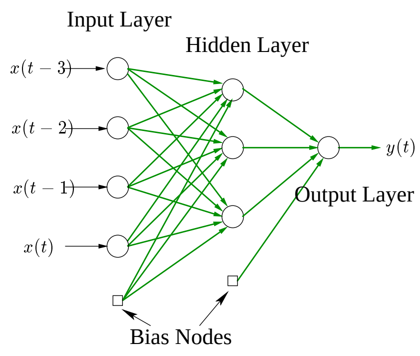


Figure 2.8: Structure of a time-delayed neural network (TDNN).

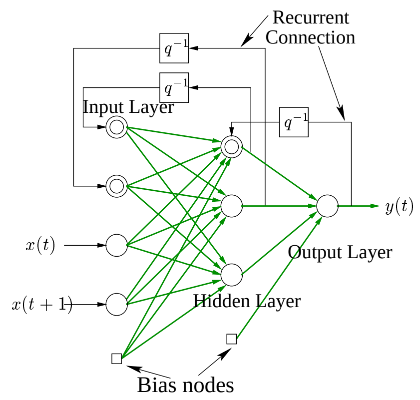


Figure 2.9: Structure of a recurrent neural network (RNN). Here q^{-1} is the delay operator that delays $x(t)$ by one unit time.

FIR NN use an explicit memory to store historical information. Although such a memory gives 100% accurate historical information, its size is always limited because an increase in memory size increases significantly the complexity of the structure and makes learning much more difficult.

Figure 2.9 gives a frequently used structure of recurrent networks. It differs from a general feedforward network in its feedback structure. A feedback connection enables information to be stored implicitly inside the network because the outputs of a layer closer to the output layer are fed back to an earlier layer. The recurrent structure make it possible to use an infinitely long history in modeling a time series. However, its drawback is that the historical information is not always accurate and errors may be amplified in the closed-loop iterative structure.

A radial-basis-function network is different from TDNN in terms of the transfer functions used in its hidden nodes. The transfer function in a TDNN is a univariate function (such as a sigmoidal or a tanh function), whereas the transfer function in an RBF network is a multivariate function called the radial basis function (e.g. Gaussian function) in which each variable corresponds to an input. The parameters in a radial basis function are a vector having the same dimension as the input vector. The radial basis function computes the distance between the input vector and its parameter vector and performs a nonlinear transformation. RBF network is a local approximator that stores history input patterns in its radial basis approximately. As a result, it may require a lot of radial basis if the time series has little periodicity and is highly nonlinear.

Support vector machines try to find optimal hyperplanes to separate input patterns. Since they are quite different from the architectures studied in this thesis, we will not

discuss them in details. Readers can refer to Haykin's book [57, 68, 108] for further reading.

2.6.2 Learning of traditional ANNs

Time-series predictions using ANNs have traditionally been formulated as unconstrained optimization problems that minimize the mean squared errors (MSE) defined as follows:

$$\min_{w} \; \mathcal{E}(w) = \sum_{t=1}^{n} \sum_{i=1}^{N_o} (o_i(t) - d_i(t))^2, \tag{2.26}$$

where N_o is the number of output nodes in the ANN, $o(t)$ and $d(t)$ are, respectively, the actual and desired outputs of the ANN at time t; w is a vector of all the weights; and the training data consists of patterns observed at $t = 1, \cdots, n$.

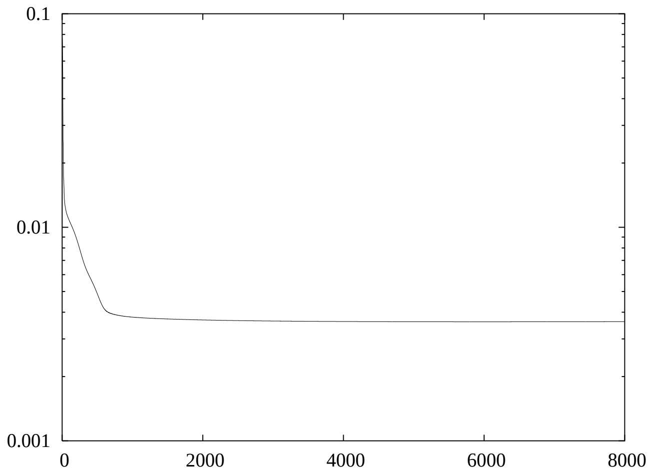
Extensive past research has been conducted on designing learning algorithms using an unconstrained formulation in order to lead to ANNs with a small number of weights that can generalize well. However, such learning algorithms have limited success because little guidance is provided in an unconstrained formulation when a search is stuck in a local minimum of the weight space. In this case, the unconstrained objective in (4.1) (to be discussed later) does not indicate which patterns are violated and the best direction for the trajectory to move.

Figure 2.10 illustrates a search getting stuck with a lack of guidance when an unconstrained formulation is used. In this example, an ANN was trained by backpropagation to predict the *Sunspot* time series. Figure 2.10a shows that the MSE in training decreases quickly in the first 1000 iterations but has little improvement after 2000 iterations. Further examination of the weights shows that they are almost frozen after 2000 iterations, and the gradients in all the iterations thereafter are very small. Yet the pattern errors in Figure 2.10b show that there are still considerably large errors for some patterns, and that these violated patterns are not identified in an unconstrained formulation. To address this issue, we propose in Chapter 4 a constrained formulation with a constraint on each pattern, and an efficient algorithm for searching in constrained space.

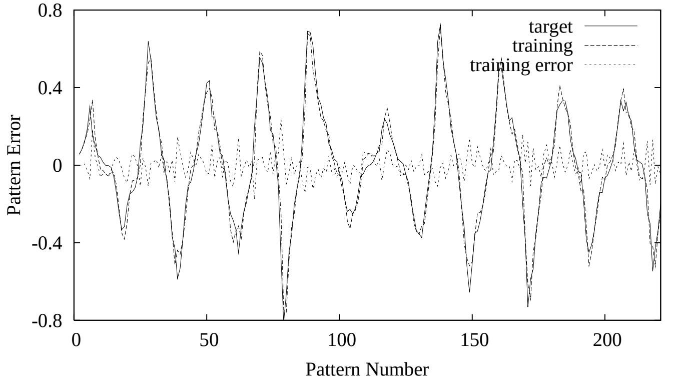
Besides minimizing training errors, cross validations have been used to prevent overfitting in ANN training. Traditional learning involving cross validations generally divides the available historical data into two disjoint training and validation sets and uses the MSE of the validation set as the sole objective. The reason for using only one validation set is due to the limitation of unconstrained formulations that can handle only one objective function.

A problem faced in traditional validations is in choosing a proper cross-validation set. Although there is no defined way on how long and where the validation set should be, one prefers to reserve the last portion of the historical data for validation in order for the ANN to generalize well into the future. Since the training and validations sets are disjoint, the use of the last portion of patterns as a validation set prevents them from being used as training patterns. As a result, the ANN learned does not have access to the most recent patterns in a time series that are usually the most important for predicting into the future. This is the dilemma in traditional cross validations when used in time-series predictions.

Another problem faced in traditional validations is related to piecewise chaotic time series. Since piecewise chaotic time series behave differently at change-over points, these points need to be learned specifically in order for the learned system to generalize well. For example, the *Laser* time series [173] in Figure 2.11 is a piecewise chaotic time series with two change-over points at 180 and 600, respectively. To learn these change-over



a) Progress of mean squared errors



b) Training errors on individual patterns after 2000 iterations

Figure 2.10: *Sunspots* time series trained by backpropagation using an unconstrained formulation in (4.1).

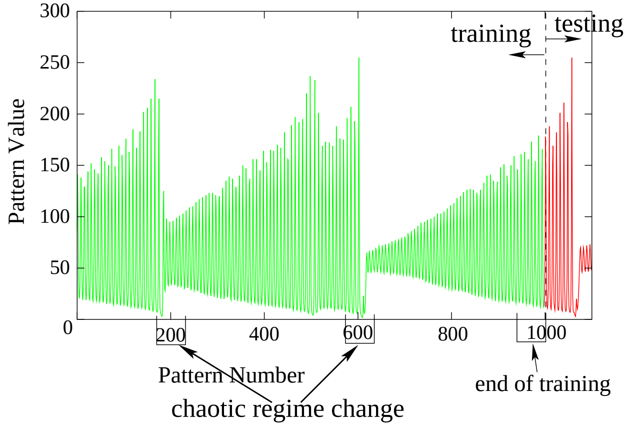


Figure 2.11: *Laser* is a piecewise chaotic time-series that requires at least three validation sets: two for the regime change-over sections and another at the end of training.

points, we like to have two validation sets using the segment around $t = 200$ and the segment around $t = 600$, and another validation set on the segment right before the end of the training set, say from $t = 900$ to $t = 1000$. Such multi-objective learning cannot be handled by traditional single-objective formulations but can be modeled in a constrained formulation that considers the error of each cross-validation set as an additional constraint.

2.6.3 Remarks on existing work on ANNs

Because traditional ANNs can only handle a single objective function, the average learning error (or the maximum learning error in some cases) in traditional ANNs is the only criterion in learning algorithms developed for traditional ANNs. As a result, these ANNs

have difficulty in dealing with problems that require more than one objectives. To this end, learning based on a constrained formulation can be more effective.

2.7 Summary

Some existing time-series models assume a specific form of nonlinearity (including linear form) on the time series under consideration. Those models fail when the assumptions made do not hold. Other existing time-series models that make no assumption cannot handle multiple objectives that are essential in learning and they often get stuck in suboptimal solutions. Existing work on handling noisy time series incurs lags, and the ways the lags are handled lead to certain edge effect. A new constrained formulation will address these limitations by allowing multiple objectives and user-specified domain-dependent constraints to be incorporated.

Chapter 3

Proposed Preprocessing Approaches

In this thesis, we are only concerned with the preprocessing of financial time series. It is well-known that financial time series is noisy (mainly containing high-frequency noise) and nonstationary. In this chapter, we propose a new preprocessing approach to handle both high-frequency noise and nonstationarity.

As discussed in Chapter 2, a time series contaminated with noise is very hard to model if the noise level is high. By performing preprocessing to remove noise and unnecessary information that confuses or complicate the modeling process, we can reveal important information in the time series that can be predicted. On the other hand, preprocessing may also lead to the loss of predictable, useful information. In this chapter, we study trade-offs between information loss and noise reduction.

Traditionally, there are two types of denoising approaches, namely, frequency-based denoising and coefficient-based denoising. Frequency-based denoising approaches [95, 96, 179, 176] assume that noises are present only in certain frequency channels. Based on this assumption, a time series is decomposed into different frequency channels using either low-pass filters or wavelet decomposition, and channels contaminated by noise are discarded.

On the other hand, coefficient-based denoising approaches [25, 31, 24, 122, 123] assume that noise has a small magnitude in a transformed domain (e.g. frequency or wavelet space). Based on this assumption, a time series is transformed into a different domain (frequency or wavelet domain), and small coefficients are removed in that domain.

Financial time series, such as stock-price time series often experience very short term event-driven price jumps/drops, and noisy small transactions but large price changes on false information. Such price movements cannot be predicted well. Since the price returns to its normal level very soon, those movements should be considered as noise. As those movements belong to high frequency components with considerable large magnitudes, they will not be eliminated by coefficient-based denoising. As a result, coefficient-based denoising will make both training and predictions more difficult. In this case, low-pass filtering is more suitable to remove high-frequency noise. However, the low-pass filtering always incurs lags in time-series predictions. To predict the future, one has to overcome the lag. In this chapter, we introduce a new approach to achieve a similar effect of low-pass filtering while incurring as little lags as possible.

As to nonstationarity, we will introduce a transformation called *pattern-wise standardization* to convert nonstationary components into a near chaotic series.

3.1 The need for preprocessing

There are two main goals in time-series preprocessing. The first goal is to make sure that preprocessing will output a set or multiple sets of *predictable* time series with little high-frequency noise, whereas the second is to convert a nonstationary time series into signals that are almost chaotic.

To achieve the first goal, we need to make sure that the preprocessed time series is predictable. In this case, we need to assess the predictability of the time series using both the autocorrelation and predictability metrics defined in Chapter 1. To make sure that the preprocessed time series has little noise, the techniques presented in Section 2.5 should be applied.

The second goal is to remove nonstationarity and convert a time series to a near-chaotic one so that ANN models developed for predicting chaotic time series can be applied. Since a financial time series has clear long-term nonstationarity, what has been learned from historical behavior does not always apply to the future. In this chapter, we discuss methods to extract the majority of nonstationary components from a financial time series and convert the time series into a chaotic one.

Throughout this chapter, we use stock price time series as examples to illustrate our proposed preprocessing techniques.

To determine whether the first goal is achieved, one has to determine if the resulting signals are predictable or not. To this end, one has to train a predictor and perform in-sample predictions to see if the predictions are satisfactory. But this is a time consuming process. One shortcut is to compute the autocorrelation and predictability measures described in Section 1.3 for the resulting signals. Assume that preprocessed time series A with predictability $P_1(A)$ and autocorrelation $\rho_1(A)$ is not predictable. Then, if preprocessed time series B (obtained through a different set of preprocessing parameters) with predictability measure $P_1(B) < P_1(A)$ and autocorrelation $\rho_1(B) < \rho_1(A)$, then B is not predictable either. This property facilitates the selection of parameters in our time-series preprocessing. For example, it helps us exclude certain filters from consideration without training the resulted time series. Here, we only look at one step in the horizon because,

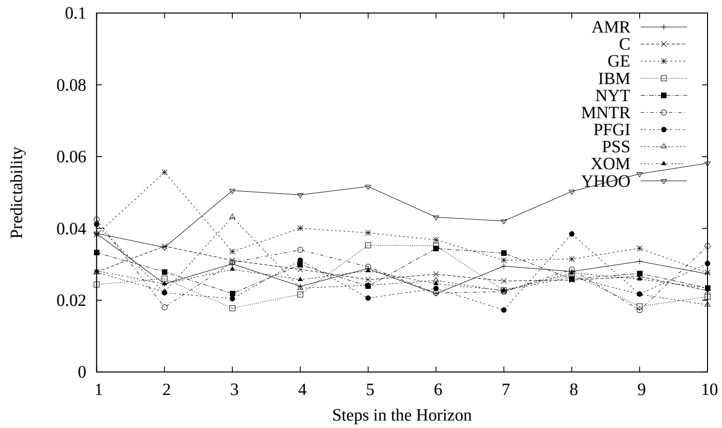


Figure 3.1: Predictability of ten stock-price benchmarks (average price of daily low and high prices) at different prediction horizons.

if a time series does not have a high predictability for one step in the horizon, then it cannot be predicted well for longer steps in the horizon.

Figure 3.1 shows the predictability of ten stock-price benchmarks presented in Chapter 1. The upper panel in Figure 3.2 shows large correlations of raw stock-price time series, and the lower panel shows small correlations of the corresponding differenced series (difference between two consecutive values of the average of daily low and high prices). The upper panel indicates that the overall price levels are predictable. This is not surprising because the price level of mature stocks, such as GE, changes little from day to day. The lower panel indicates that price movements are not predictable for raw stock-price time series. The low predictability $P_1(X)$ of stock-price time series is easily connected to the random walk behavior of stock prices. In fact, this observation is consistent with some

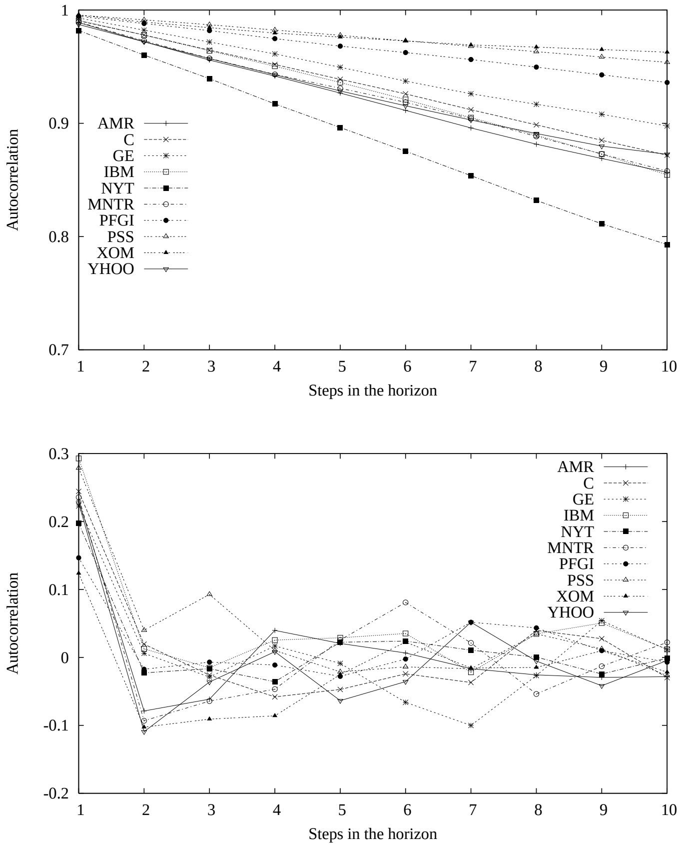


Figure 3.2: Upper panel: autocorrelation of the average daily low and high prices for ten stocks at different prediction horizons. Lower panel: autocorrelation of the differenced series for the average daily low and high prices for the same stocks.

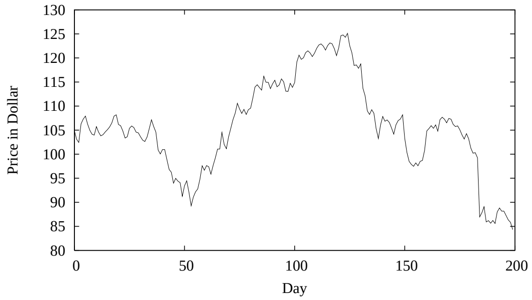


Figure 3.3: Averaged daily low and high prices for IBM stock

findings in the literature [88, 89, 90]. Our measurements on predictability on stock-price time series further verify that the raw stock prices are hard to predict.

The discussion above shows that stock-price time series has very low predictability. To this end, preprocessing can be of great help. Preprocessing can improve the predictability of a time series in order to make modeling and prediction easier. First, consider an example of stock-price time-series preprocessing.

Figure 3.3 plots the averaged daily low/high IBM stock prices, which exhibits a noisy behavior from day to day. Figure 3.4 shows the frequency response of a 10-tap low-pass filter used for low-pass filtering, and the frequency response of the corresponding high-pass filter to generate the residue noise. The low-pass filtered data is plotted in Figure 3.5(a), and the corresponding noise data is plotted in Figure 3.5(b). Compared with the original raw time series, the low-pass data are much smoother, and the residue data are dominated by high frequency noise. Figure 3.6 shows the predictabilities and autocorrelations for both the low-pass and high-pass data. P_1 and ρ_1 for the high-pass data are relatively low when compared to P_1 and ρ_1 for the low-pass data. Hence, it is

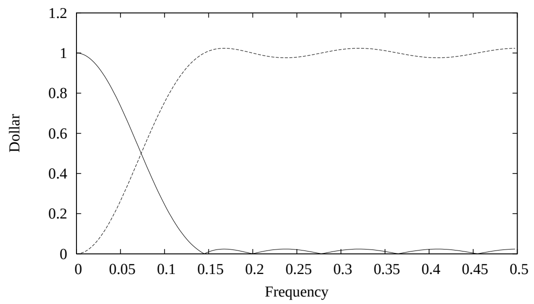
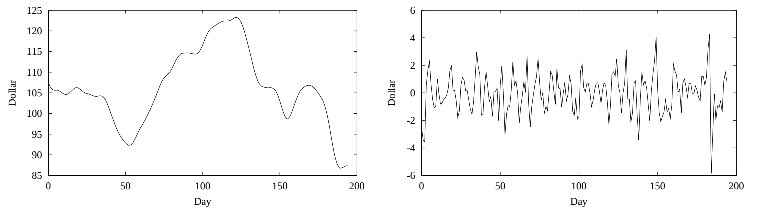


Figure 3.4: Frequency response of a 10-tap low-pass FIR filter (solid line) and the corresponding high-pass FIR filter (dashed line).

possible that the low-pass data is predictable, whereas the high-pass data may not. On the other hand, there may also be some useful information buried in the high-pass data. This can be seen from the fact that ρ_1 for high-pass data is close to 0.4 for one step in the horizon. In short, the balance between improving predictability and information loss is the main topic in preprocessing.

For the second goal of handling nonstationarity, a popular technique is differencing [95, 96, 59]. However, as seen in Figures 3.1 and 3.2, a differenced time series is also very hard to predict if it can be predicted at all. On the other hand, noise in stock price is mainly of high frequency; therefore, it will be helpful if we can decompose the original raw time series (called signals) into different frequency bands. The noise resides mainly in the high-frequency band, and the low-frequency band consists of mainly predictable signals. This property enables us to treat different frequency bands differently and to prevent information loss in the low-frequency band. In our work, we extract the majority



(a) (b)

Figure 3.5: Low-pass filtered data (left panel) and high-pass filtered data (right panel) for the averaged daily low and high prices for the IBM stock shown in Figure 3.3.

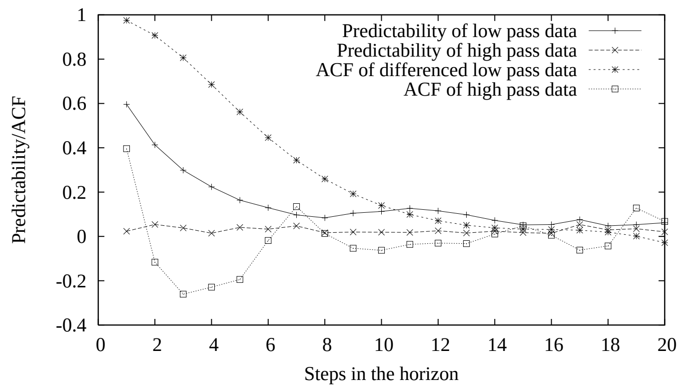


Figure 3.6: Predictabilities and autocorrelations of the low-pass and high-pass data. Both low-pass and high-pass filters have 10 taps (i.e. 11 filter coefficients).

of nonstationary components into one channel and then transform the nonstationary component into a near chaotic series.

In summary, preprocessing is a process in which one employs a variety of methods to remove unpredictable components by using techniques such as denoising, or to improve predictability by decomposing a time series into different components so that each component can be better predicted.

In this chapter, we first study systematically the traditional *simple low-pass filtering* approach and select three representative low-pass filters. In a simple low-pass filtering approach, an input time series is passed through a low-pass filter to obtain a preprocessed time series. We discuss how filters are selected in this approach. We then present our new preprocessing approach that combines wavelet decomposition with low-pass filtering. The wavelet decomposition decomposes a time series into several different frequency channels. The majority of nonstationarity (in terms of magnitude) is absorbed into one channel, and the majority of noise is distributed into other channels with smaller magnitudes and close to chaotic. The nonstationary channel is then transformed into near chaotic signals, and the noisy channels are passed through channel-specific filters. The result is a collection of time series of different characteristics.

3.2 Traditional low-pass filtering

In this section, we study traditional low-pass filtering and trade-offs between noise reduction and information loss. We then select three representative low-pass filters as benchmarks (targets) for our new preprocessing approach.

As discussed in Section 3.1, the main goal for time-series preprocessing is to improve predictability while minimizing information loss. Throughout this and the next section, we assume that stock-price time series is contaminated mainly by high-frequency noise.

Low-pass filtering may remove both noise and valuable information. If the cut-off frequency for a filter is much lower than what is necessary to remove high frequency noise, then there will be information loss. However, if the cut-off frequency is too high, then the noise remaining in the time series will make it difficult to predict. Since in reality, we have no idea on the optimal cut-off frequency, we keep three representative filters with low/medium/high cut-off frequencies in our approach. To obtain the three representative filters, we compare the performance of different filters designed in Matlab with different cut-off frequencies and number of filter taps. The selected representative filters should be relatively well-balanced between noise reduction and information preservation when compared to their peer filters. Note that our approach is specific to stock-price time series, and the filters selected may not generalize well to other noisy time series.

Consider the frequency spectra for the ten benchmark stocks studied. The frequency spectra for the averaged daily low and high prices are plotted in Figure 3.7. Those graphs show that the spectra are similarly distributed in terms of their relative magnitudes over different frequency bands. They show that a proper preprocessing approach for one stock is most likely also a proper one for another stock in the benchmark set.

3.2.1 Filter selection

Since low-pass data is the result of denoising, we expect that there will be a certain fraction of data outside the daily low/high price range. It is reasonable to expect that the ratio of low-pass data within the daily low/high price range to be high. Consider

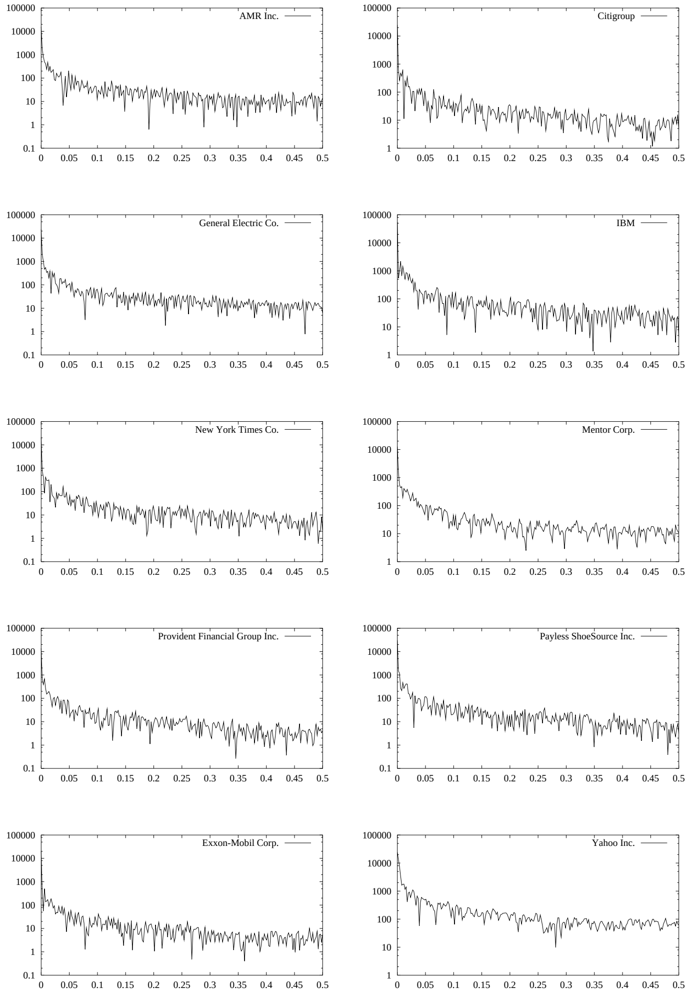


Figure 3.7: Spectra for the averaged daily low and high prices for the ten benchmark stocks in Table 1.3. The horizontal label for each graph is frequency and the vertical label for each graph is the magnitude for the frequency in the frequency space.

the notations used in Section 1.3.3, and define the low/high *hit ratio* α_0, *over-hit ratio* α_+, and *under-hit ratio* α_-. Let $s(t)$ be the low-pass data, $\mathcal{A}_0$ be the set $\{t|R_l(t) \leq s(t) \leq R_h(t)$ for all $t \in [t_0, t_1]\}$, $\mathcal{A}_+$ be the set $\{t|s(t) > R_h(t)\}$, and $\mathcal{A}_-$ be the set $\{t|s(t) < R_l(t)\}$. $\mathcal{A}_0$ consists of all the days when the corresponding low-pass data is within the daily low/high price range of that day; $\mathcal{A}_+$ consists of all the days when the corresponding low-pass data is higher than the daily high price of that day; and $\mathcal{A}_-$ consists of all the days when the corresponding low-pass data is lower than the daily low price. The hit ratio α_0, over-hit ratio α_+, and under-hit ratio α_- are then defined as:

$$\begin{aligned} \alpha_0 &= \tfrac{||\mathcal{A}_0||}{t_1-t_0+1}, \\ \alpha_+ &= \tfrac{||\mathcal{A}_+||}{t_1-t_0+1}, \\ \alpha_- &= \tfrac{||\mathcal{A}_-||}{t_1-t_0+1}. \end{aligned} \tag{3.1}$$

Hit ratio α_0 shows the ratio of the low-pass data falling inside the low/high price range, and over-hit ratio α_+ (resp. under-hit ratio α_-) shows the ratio of the low-pass data higher (resp. lower) than daily high (resp. low) price. These ratios show how good the low-pass data follows the raw price data. Hit ratios w.r.t. daily low/high price range is extensively used in performance evaluation in Chapter 6.

The ratios of low-pass data with respect to potential buy/sell signals (Section 1.3.3.4) indicate the success ratio of a potential buy or sell transaction when the bid at the predicted price is to be placed. In case of a potential buy signal is given, we can define ratios w.r.t. buy/sell signals similar to Equation (3.1). Let $\mathcal{B}_0$ be the set $\{t|G_t(t+1) = 0\}$ with $G_t(t+1)$ defined by (1.24), $\mathcal{B}_+ = \{t|G_t(t+1) > 0\}$, and $\mathcal{B}_- = \{t|G_t(t+1) < 0\}$.

$$\begin{aligned} \beta_0 &= \tfrac{||\mathcal{B}_0||}{t_1-t_0+1}, \\ \beta_+ &= \tfrac{||\mathcal{B}_+||}{t_1-t_0+1}, \\ \beta_- &= \tfrac{||\mathcal{B}_-||}{t_1-t_0+1}. \end{aligned} \tag{3.2}$$

β_0 is the ratio that one can buy/sell stock at the exact predicted prices. β_+ represents the ratio that one can buy (resp. sell) stock at lower (resp. higher) than the predicted prices. β_- represents the ratio that one can only buy (resp. sell) stock at higher (resp. lower) than the predicted prices in order to complete the signaled transactions.

To select a proper filter, we first design a set of filters with a wide range of cut-off frequencies in order to make sure the frequency band contaminated by noise will be covered in the high frequency band and removed by the low-pass filter. According to discussion in Section 3.1, we then select suitable filters that satisfies the following criteria:

Criterion 1 The low-pass data should have relatively high predictabilities and ACF as compared to those of the raw data.

Criterion 2 The low-pass data should have considerably high ratios of data inside the low/high range; for example, a ratio of 60% to 70%.

Criterion 3 The low-pass data should have considerably high ratios of data consistent with the buy/sell signals.

Criterion 4 The selected filter should incur as short a lag as possible.

Next, we consider the filter design procedure and each criterion in more details.

We use the Matlab filter design function **fircls** [99] in our work. This filter design function uses a constrained least mean squared error to design FIR filters. A user can specify cut-off frequency f_0 and the number of filter taps T. A lower cut-off frequency will lead to a smoother preprocessed data and more information loss, whereas the number of filter taps determines the number of filter coefficients. A larger number of taps leads to a sharper cut-off edge in its frequency response but also incurs more lags. A filter will

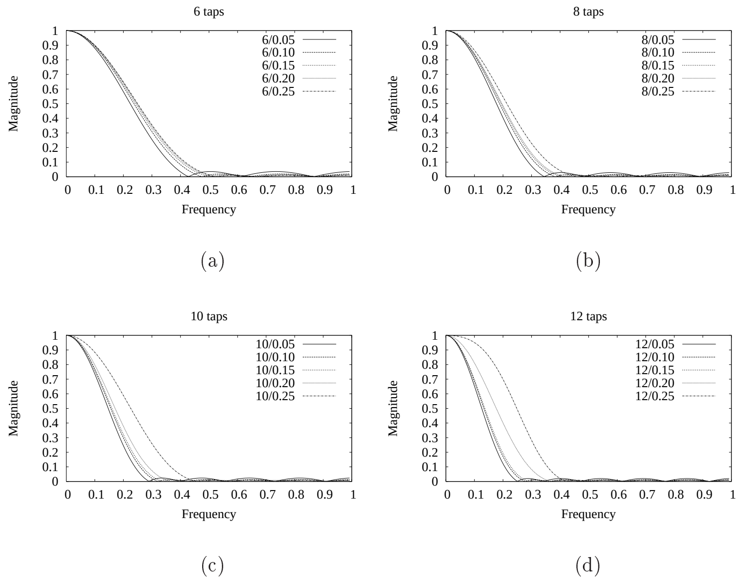


Figure 3.8: Frequency response of filters with cut-off frequency of 0.05, 0.10, 0.15, 0.20, and 0.25, respectively. (a) 6-tap filters. (b) 8-tap filters. (c) 10-tap filters. (d) 12-tap filters.

Table 3.1: Coefficients for the filters studied. All the filters are symmetric, and only the first $1 + \frac{T}{2}$ coefficients are listed for a T-tap filter.

Filter	Taps	Coefficients
6/0.05	6	0.0628, 0.1294, 0.1961, 0.2233
6/0.10	6	0.0496, 0.1256, 0.2052, 0.2392
6/0.15	6	0.0442, 0.1232, 0.2092, 0.2467
6/0.20	6	0.0412, 0.1217, 0.2115, 0.2512
6/0.25	6	0.0394, 0.1206, 0.2129, 0.2541
8/0.05	8	0.0419, 0.0794, 0.1264, 0.1635, 0.1777
8/0.10	8	0.0312, 0.0735, 0.1275, 0.1727, 0.1903
8/0.15	8	0.0270, 0.0704, 0.1276, 0.1768, 0.1963
8/0.20	8	0.0249, 0.0685, 0.1275, 0.1792, 0.1998
8/0.25	8	0.0131, 0.0605, 0.1260, 0.1916, 0.2176
10/0.05	10	0.0307, 0.0533, 0.0857, 0.11689, 0.1395, 0.1478
10/0.10	10	0.0221, 0.0476, 0.0835, 0.1201, 0.1477, 0.1580
10/0.15	10	0.0189, 0.0448, 0.0821, 0.1214, 0.1515, 0.1628
10/0.20	10	0.0103, 0.0398, 0.0779, 0.1242, 0.1608, 0.1742
10/0.25	10	-0.0060, 0.0102, 0.0556, 0.1283, 0.1983, 0.2274
12/0.05	12	0.0240, 0.0383, 0.0611, 0.0853, 0.1066, 0.1214, 0.1267
12/0.10	12	0.0169, 0.0333, 0.0578, 0.0853, 0.1106, 0.1286, 0.1351
12/0.15	12	0.0144, 0.0310, 0.0561, 0.0850, 0.1122, 0.1318, 0.1389
12/0.20	12	-0.0035, 0.0101, 0.0362, 0.0779, 0.1257, 0.1642, 0.1790
12/0.25	12	-0.0138, -0.0177, 0.0007, 0.0558, 0.1382, 0.2142, 0.2452
B2	2	$\frac{1}{4}, \frac{1}{2}$
B3	4	$\frac{1}{16}, \frac{1}{4}, \frac{3}{8}$
B4	6	$\frac{1}{64}, \frac{3}{32}, \frac{15}{64}, \frac{10}{32}$
B6	10	$\frac{1}{1024}, \frac{10}{1024}, \frac{45}{1024}, \frac{120}{1024}, \frac{210}{1024}, \frac{252}{1024}$

be referred to Filter T/f_0 unless otherwise specified. Table 3.1 lists the coefficients for all the filters studied along with the B-spline filters used in the next section.

- Figure 3.8(a) shows five 6-tap filters with cut-off frequencies of $0.05, 0.10, 0.15, 0.20$, and 0.25, respectively.
- Figure 3.8(b) shows five 8-tap filters with cut-off frequencies of $0.05, 0.10, 0.15, 0.20$, and 0.25, respectively.
- Figure 3.8(c) shows five 10-tap filters with cut-off frequencies of $0.05, 0.10, 0.15, 0.20$, and 0.25, respectively.
- Figure 3.8(d) shows five 12-tap filters with cut-off frequencies of $0.05, 0.10, 0.15, 0.20$, and 0.25, respectively.

In practice, it is very hard to overcome a large number of lags incurred by low-pass filtering due to the nonstationarity of stock market prices. Hence, we would like to select filters with a small number of taps among a group of filters with similar performance. We also already exclude filters with cut-off parameter above 0.25, as we have found that the resulting signals are hard to predict when filtered by cut-off parameters of 0.25. Moreover, filters with cut-off parameters above 0.25 lead to lower autocorrelation and lower predictability. We list the measured metrics for low-pass data for the ten selected benchmarks in Table 3.2. We measure the ACF, predictability, RMSE, and error w.r.t. low/high range.

Table 3.2: Performance for low-pass filtered stock-price time series using different filters. ACF and P indicate autocorrelation and predictability. "Hit Ratios" shows ratios w.r.t. the daily low/high price ($\alpha_0/\alpha_+/\alpha_-$) defined by (3.1) in Section 3.2.1. "Buy/Sell" shows the ratios w.r.t. potential buy/Sell signals ($\beta_0/\beta_+/\beta_-$) defined by (3.2) in Section 3.2.1.

American Airline (AMR)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.921	0.398	0.018	0.733/0.135/0.132	0.726/0.119/0.155
6/0.10	0.912	0.378	0.017	0.751/0.125/0.124	0.742/0.114/0.144
6/0.15	**0.906**	**0.367**	**0.017**	**0.761/0.122/0.117**	**0.751/0.114/0.135**
6/0.20	0.903	0.363	0.017	0.764/0.120/0.116	0.754/0.113/0.134
6/0.25	0.901	0.354	0.017	0.767/0.118/0.115	0.757/0.111/0.132
8/0.25	0.927	0.414	0.019	0.728/0.137/0.135	0.721/0.120/0.159
10/0.25	0.922	0.400	0.018	0.737/0.132/0.131	0.729/0.118/0.153
12/0.25	0.914	0.376	0.017	0.758/0.125/0.117	0.747/0.118/0.136
8/0.05	0.949	0.474	0.021	0.667/0.168/0.165	0.660/0.141/0.199
8/0.10	**0.944**	**0.449**	**0.020**	**0.688/0.157/0.155**	**0.680/0.133/0.186**
8/0.15	0.941	0.437	0.020	0.697/0.151/0.151	0.690/0.130/0.180
8/0.20	0.939	0.434	0.020	0.699/0.151/0.150	0.691/0.130/0.178
10/0.20	0.953	0.483	0.022	0.654/0.176/0.170	0.647/0.147/0.206
12/0.20	0.950	0.475	0.021	0.666/0.169/0.166	0.657/0.143/0.200
10/0.10	0.960	0.526	0.023	0.633/0.185/0.182	0.625/0.151/0.223
10/0.15	0.958	0.518	0.023	0.640/0.182/0.179	0.632/0.150/0.218
10/0.05	**0.964**	**0.544**	**0.024**	**0.611/0.197/0.192**	**0.610/0.152/0.238**
12/0.05	0.972	0.575	0.027	0.565/0.222/0.214	0.569/0.160/0.271
12/0.10	0.970	0.562	0.026	0.587/0.209/0.204	0.588/0.156/0.256
12/0.15	0.969	0.554	0.026	0.594/0.204/0.202	0.593/0.155/0.252

continued on next page

continued from previous page

Citigroup Stock					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.911	0.375	0.011	0.738/0.128/0.134	0.729/0.117/0.154
6/0.10	0.903	0.365	0.010	0.760/0.116/0.124	0.753/0.106/0.141
6/0.15	**0.898**	**0.356**	**0.010**	**0.767/0.113/0.121**	**0.760/0.103/0.137**
6/0.20	0.895	0.347	0.010	0.772/0.109/0.119	0.766/0.100/0.134
6/0.25	0.894	0.346	0.010	0.777/0.106/0.117	0.771/0.097/0.133
8/0.25	0.917	0.395	0.011	0.728/0.132/0.141	0.720/0.116/0.164
10/0.25	0.912	0.382	0.010	0.743/0.124/0.132	0.735/0.113/0.152
12/0.25	0.904	0.363	0.010	0.767/0.111/0.122	0.758/0.104/0.138
8/0.05	0.941	0.450	0.012	0.670/0.163/0.167	0.667/0.129/0.204
8/0.10	**0.934**	**0.424**	**0.012**	**0.687/0.156/0.157**	**0.683/0.128/0.190**
8/0.15	0.931	0.419	0.012	0.696/0.151/0.153	0.692/0.126/0.182
8/0.20	0.929	0.421	0.011	0.703/0.148/0.149	0.699/0.124/0.177
10/0.20	0.944	0.467	0.013	0.662/0.168/0.171	0.661/0.131/0.208
12/0.20	0.942	0.452	0.012	0.670/0.162/0.168	0.667/0.129/0.205
10/0.10	0.954	0.493	0.013	0.632/0.180/0.187	0.631/0.136/0.233
10/0.15	0.951	0.489	0.013	0.640/0.177/0.183	0.637/0.136/0.227
10/0.05	**0.959**	**0.506**	**0.014**	**0.610/0.193/0.196**	**0.614/0.141/0.245**
12/0.05	0.971	0.547	0.015	0.571/0.216/0.213	0.589/0.140/0.271
12/0.10	0.967	0.535	0.015	0.584/0.208/0.208	0.602/0.137/0.262
12/0.15	0.965	0.537	0.014	0.591/0.206/0.203	0.605/0.139/0.256

continued on next page

continued from previous page

General Electric (GE)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.921	0.398	0.018	0.733/0.135/0.132	0.726/0.119/0.155
6/0.10	0.912	0.378	0.017	0.751/0.125/0.124	0.742/0.114/0.144
6/0.15	**0.906**	**0.367**	**0.017**	**0.761/0.122/0.117**	**0.751/0.114/0.135**
6/0.20	0.903	0.363	0.017	0.764/0.120/0.116	0.754/0.113/0.134
6/0.25	0.901	0.354	0.017	0.767/0.118/0.115	0.757/0.111/0.132
8/0.25	0.927	0.414	0.019	0.728/0.137/0.135	0.721/0.120/0.159
10/0.25	0.922	0.400	0.018	0.737/0.132/0.131	0.729/0.118/0.153
12/0.25	0.914	0.376	0.017	0.758/0.125/0.117	0.747/0.118/0.136
8/0.05	0.949	0.474	0.021	0.667/0.168/0.165	0.660/0.141/0.199
8/0.10	**0.944**	**0.449**	**0.020**	**0.688/0.157/0.155**	**0.680/0.133/0.186**
8/0.15	0.941	0.437	0.020	0.697/0.151/0.151	0.690/0.130/0.180
8/0.20	0.939	0.434	0.020	0.699/0.151/0.150	0.691/0.130/0.178
10/0.20	0.953	0.483	0.022	0.654/0.176/0.170	0.647/0.147/0.206
12/0.20	0.950	0.475	0.021	0.666/0.169/0.166	0.657/0.143/0.200
10/0.10	0.960	0.526	0.023	0.633/0.185/0.182	0.625/0.151/0.223
10/0.15	0.958	0.518	0.023	0.640/0.182/0.179	0.632/0.150/0.218
10/0.05	**0.964**	**0.544**	**0.024**	**0.611/0.197/0.192**	**0.610/0.152/0.238**
12/0.05	0.972	0.575	0.027	0.565/0.222/0.214	0.569/0.160/0.271
12/0.10	0.970	0.562	0.026	0.587/0.209/0.204	0.588/0.156/0.256
12/0.15	0.969	0.554	0.026	0.594/0.204/0.202	0.593/0.155/0.252

continued on next page

continued from previous page

International Business Machine (IBM)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.931	0.396	0.009	0.749/0.118/0.133	0.749/0.103/0.147
6/0.10	0.923	0.383	0.009	0.770/0.108/0.122	0.771/0.095/0.134
6/0.15	**0.919**	**0.378**	**0.009**	**0.780/0.104/0.116**	**0.781/0.093/0.126**
6/0.20	0.916	0.365	0.009	0.782/0.102/0.116	0.783/0.092/0.125
6/0.25	0.914	0.365	0.009	0.786/0.101/0.113	0.787/0.092/0.121
8/0.25	0.937	0.419	0.010	0.738/0.125/0.136	0.739/0.109/0.151
10/0.25	0.931	0.404	0.009	0.753/0.116/0.130	0.754/0.102/0.144
12/0.25	0.925	0.381	0.009	0.786/0.104/0.110	0.786/0.095/0.118
8/0.05	0.956	0.475	0.011	0.686/0.150/0.165	0.686/0.122/0.192
8/0.10	**0.951**	**0.458**	**0.011**	**0.705/0.140/0.155**	**0.704/0.117/0.178**
8/0.15	0.948	0.441	0.010	0.710/0.137/0.153	0.709/0.117/0.174
8/0.20	0.947	0.436	0.010	0.715/0.136/0.149	0.715/0.116/0.169
10/0.20	0.960	0.483	0.011	0.679/0.152/0.169	0.679/0.123/0.198
12/0.20	0.959	0.481	0.011	0.688/0.146/0.166	0.687/0.120/0.193
10/0.10	0.967	0.508	0.012	0.651/0.166/0.182	0.654/0.128/0.218
10/0.15	0.965	0.498	0.012	0.658/0.164/0.179	0.661/0.128/0.211
10/0.05	**0.970**	**0.521**	**0.013**	**0.636/0.172/0.192**	**0.640/0.129/0.231**
12/0.05	0.979	0.567	0.014	0.592/0.195/0.212	0.604/0.135/0.261
12/0.10	0.976	0.554	0.013	0.611/0.185/0.204	0.619/0.132/0.249
12/0.15	0.975	0.544	0.013	0.622/0.181/0.197	0.629/0.132/0.239

continued on next page

continued from previous page

Mentor (MNTR)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.917	0.390	0.010	0.814/0.099/0.087	0.821/0.077/0.101
6/0.10	0.906	0.371	0.010	0.829/0.091/0.080	0.835/0.072/0.093
6/0.15	**0.900**	**0.363**	**0.009**	**0.835/0.088/0.077**	**0.840/0.070/0.090**
6/0.20	0.897	0.359	0.009	0.839/0.086/0.074	0.843/0.071/0.086
6/0.25	0.894	0.357	0.009	0.842/0.085/0.073	0.845/0.070/0.085
8/0.25	0.923	0.405	0.010	0.806/0.106/0.088	0.814/0.083/0.103
10/0.25	0.916	0.398	0.010	0.817/0.098/0.086	0.823/0.077/0.100
12/0.25	0.910	0.383	0.010	0.833/0.087/0.080	0.837/0.071/0.092
8/0.05	0.947	0.465	0.012	0.748/0.132/0.119	0.760/0.096/0.143
8/0.10	**0.941**	**0.451**	**0.011**	**0.767/0.125/0.108**	**0.779/0.091/0.129**
8/0.15	0.937	0.440	0.011	0.776/0.120/0.104	0.788/0.088/0.124
8/0.20	0.935	0.435	0.011	0.781/0.118/0.101	0.792/0.088/0.120
10/0.20	0.950	0.480	0.012	0.741/0.135/0.124	0.753/0.097/0.150
12/0.20	0.948	0.480	0.012	0.749/0.132/0.119	0.761/0.095/0.143
10/0.10	0.958	0.510	0.013	0.708/0.149/0.142	0.722/0.104/0.173
10/0.15	0.956	0.501	0.013	0.721/0.144/0.135	0.734/0.102/0.164
10/0.05	**0.961**	**0.526**	**0.013**	**0.691/0.160/0.149**	**0.709/0.108/0.183**
12/0.05	0.971	0.558	0.015	0.650/0.181/0.169	0.674/0.114/0.212
12/0.10	0.968	0.551	0.014	0.667/0.171/0.163	0.689/0.108/0.203
12/0.15	0.967	0.544	0.014	0.674/0.168/0.158	0.695/0.109/0.196

continued on next page

continued from previous page

New York Times (NYT)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.918	0.378	0.007	0.773/0.115/0.112	0.771/0.101/0.129
6/0.10	0.909	0.358	0.007	0.793/0.104/0.103	0.788/0.094/0.118
6/0.15	**0.905**	**0.349**	**0.007**	**0.800/0.100/0.100**	**0.796/0.090/0.114**
6/0.20	0.902	0.345	0.007	0.804/0.098/0.097	0.800/0.090/0.110
6/0.25	0.900	0.337	0.007	0.808/0.097/0.095	0.803/0.089/0.108
8/0.25	0.924	0.401	0.008	0.765/0.119/0.116	0.764/0.103/0.132
10/0.25	0.917	0.375	0.007	0.778/0.113/0.110	0.775/0.098/0.126
12/0.25	0.910	0.361	0.007	0.801/0.099/0.100	0.796/0.090/0.113
8/0.05	0.947	0.443	0.008	0.711/0.145/0.145	0.711/0.118/0.170
8/0.10	**0.941**	**0.429**	**0.008**	**0.732/0.136/0.132**	**0.734/0.112/0.154**
8/0.15	0.937	0.426	0.008	0.740/0.131/0.130	0.742/0.108/0.150
8/0.20	0.935	0.422	0.008	0.743/0.128/0.130	0.743/0.107/0.150
10/0.10	0.959	0.481	0.009	0.674/0.161/0.165	0.681/0.122/0.197
10/0.15	0.957	0.472	0.009	0.683/0.156/0.160	0.690/0.119/0.191
10/0.20	0.950	0.456	0.009	0.703/0.148/0.149	0.706/0.118/0.176
12/0.20	0.949	0.455	0.008	0.711/0.143/0.146	0.712/0.116/0.172
10/0.10	0.959	0.481	0.009	0.674/0.161/0.165	0.681/0.122/0.197
10/0.15	0.957	0.472	0.009	0.683/0.156/0.160	0.690/0.119/0.191
10/0.05	**0.963**	**0.494**	**0.009**	**0.657/0.171/0.172**	**0.666/0.127/0.207**
12/0.05	0.974	0.528	0.010	0.619/0.187/0.194	0.636/0.125/0.239
12/0.10	0.971	0.525	0.010	0.635/0.179/0.186	0.648/0.126/0.226
12/0.15	0.969	0.518	0.010	0.644/0.176/0.180	0.656/0.126/0.219

continued on next page

continued from previous page

Provident Financial Group (PFGI)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.919	0.415	0.008	0.790/0.114/0.097	0.792/0.099/0.109
6/0.10	0.911	0.395	0.007	0.805/0.107/0.089	0.808/0.094/0.098
6/0.15	**0.906**	**0.387**	**0.007**	**0.812/0.103/0.085**	**0.815/0.093/0.093**
6/0.20	0.903	0.381	0.007	0.816/0.101/0.083	0.819/0.091/0.090
6/0.25	0.901	0.373	0.007	0.818/0.100/0.082	0.821/0.090/0.089
8/0.25	0.927	0.439	0.008	0.783/0.117/0.099	0.787/0.100/0.113
10/0.25	0.919	0.425	0.008	0.793/0.111/0.096	0.795/0.098/0.107
12/0.25	0.909	0.399	0.007	0.822/0.097/0.081	0.823/0.088/0.089
8/0.05	0.950	0.481	0.009	0.741/0.137/0.122	0.747/0.109/0.143
8/0.10	**0.944**	**0.472**	**0.008**	**0.753/0.131/0.116**	**0.759/0.107/0.135**
8/0.15	0.940	0.465	0.008	0.761/0.128/0.111	0.766/0.105/0.129
8/0.20	0.938	0.457	0.008	0.763/0.126/0.111	0.769/0.103/0.128
10/0.20	0.953	0.490	0.009	0.737/0.138/0.125	0.745/0.107/0.148
12/0.20	0.950	0.483	0.009	0.742/0.137/0.122	0.748/0.109/0.143
10/0.10	0.962	0.531	0.010	0.713/0.150/0.136	0.726/0.112/0.162
10/0.15	**0.960**	**0.523**	**0.009**	**0.722/0.147/0.132**	**0.734/0.110/0.156**
10/0.05	0.966	0.553	0.010	0.696/0.161/0.143	0.711/0.116/0.173
12/0.05	0.975	0.593	0.011	0.660/0.176/0.164	0.677/0.120/0.203
12/0.10	0.972	0.579	0.011	0.673/0.171/0.156	0.690/0.119/0.191
12/0.15	0.971	0.572	0.010	0.677/0.170/0.153	0.693/0.120/0.188

continued on next page

continued from previous page

Payless ShowSource (PSS)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.929	0.415	0.007	0.747/0.133/0.120	0.735/0.116/0.149
6/0.10	0.923	0.388	0.007	0.765/0.126/0.109	0.753/0.115/0.132
6/0.15	**0.920**	**0.383**	**0.007**	**0.770/0.125/0.105**	**0.758/0.114/0.128**
6/0.20	0.918	0.378	0.007	0.776/0.122/0.102	0.762/0.113/0.124
6/0.25	0.916	0.377	0.007	0.777/0.120/0.102	0.764/0.112/0.124
8/0.25	0.935	0.426	0.007	0.732/0.141/0.126	0.722/0.121/0.158
10/0.25	0.930	0.412	0.007	0.751/0.132/0.117	0.742/0.114/0.144
12/0.25	0.923	0.397	0.007	0.775/0.117/0.108	0.762/0.108/0.130
8/0.05	0.953	0.485	0.008	0.664/0.176/0.159	0.657/0.141/0.202
8/0.10	**0.948**	**0.458**	**0.008**	**0.682/0.169/0.148**	**0.675/0.139/0.187**
8/0.15	0.945	0.451	0.008	0.695/0.163/0.142	0.688/0.134/0.178
8/0.20	0.944	0.451	0.008	0.701/0.158/0.140	0.691/0.133/0.176
10/0.20	0.956	0.494	0.009	0.649/0.183/0.168	0.643/0.147/0.210
12/0.20	0.954	0.491	0.008	0.660/0.177/0.163	0.654/0.142/0.204
10/0.10	0.963	0.520	0.009	0.617/0.196/0.187	0.612/0.150/0.237
10/0.15	0.961	0.511	0.009	0.624/0.192/0.184	0.617/0.150/0.233
10/0.05	**0.966**	**0.532**	**0.009**	**0.599/0.208/0.193**	**0.598/0.155/0.248**
12/0.05	0.974	0.563	0.010	0.565/0.225/0.210	0.577/0.146/0.278
12/0.10	0.972	0.547	0.010	0.574/0.221/0.205	0.580/0.153/0.268
12/0.15	0.970	0.546	0.010	0.579/0.217/0.204	0.581/0.154/0.265

continued on next page

continued from previous page

Exxon-Mobil (XOM)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.901	0.387	0.007	0.796/0.104/0.100	0.788/0.098/0.114
6/0.10	0.891	0.366	0.006	0.812/0.096/0.092	0.803/0.093/0.105
6/0.15	**0.885**	**0.360**	**0.006**	**0.817/0.095/0.088**	**0.808/0.093/0.099**
6/0.20	0.882	0.357	0.006	0.821/0.093/0.086	0.812/0.091/0.098
6/0.25	0.880	0.353	0.006	0.822/0.093/0.085	0.813/0.090/0.097
8/0.25	0.909	0.405	0.007	0.789/0.108/0.103	0.780/0.101/0.118
10/0.25	0.902	0.391	0.007	0.801/0.101/0.098	0.793/0.096/0.111
12/0.25	0.894	0.376	0.006	0.818/0.092/0.090	0.810/0.089/0.102
8/0.05	0.935	0.442	0.008	0.739/0.129/0.132	0.732/0.114/0.153
8/0.10	**0.929**	**0.431**	**0.007**	**0.756/0.121/0.123**	**0.747/0.111/0.142**
8/0.15	0.925	0.428	0.007	0.763/0.118/0.119	0.753/0.109/0.138
8/0.20	0.923	0.423	0.007	0.766/0.117/0.117	0.756/0.108/0.136
10/0.20	0.940	0.459	0.008	0.736/0.131/0.134	0.729/0.115/0.155
12/0.20	0.937	0.446	0.008	0.742/0.127/0.131	0.736/0.112/0.153
10/0.10	0.950	0.478	0.008	0.709/0.144/0.147	0.705/0.123/0.172
10/0.15	0.948	0.470	0.008	0.719/0.138/0.143	0.713/0.119/0.168
10/0.05	**0.955**	**0.494**	**0.009**	**0.692/0.153/0.155**	**0.691/0.124/0.185**
12/0.05	0.967	0.538	0.009	0.653/0.173/0.175	0.658/0.130/0.212
12/0.10	0.964	0.527	0.009	0.669/0.164/0.166	0.675/0.124/0.201
12/0.15	0.962	0.521	0.009	0.677/0.160/0.163	0.681/0.123/0.196

continued on next page

continued from previous page

Yahoo (YHOO)					
Filter	ACF	P	RMAE	Hit Ratios	Buy/Sell
6/0.05	0.909	0.402	0.020	0.769/0.122/0.109	0.777/0.097/0.126
6/0.10	0.900	0.389	0.019	0.780/0.116/0.104	0.787/0.095/0.119
6/0.15	**0.895**	**0.377**	**0.018**	**0.792/0.111/0.097**	**0.798/0.092/0.110**
6/0.20	0.893	0.371	0.018	0.799/0.107/0.094	0.805/0.089/0.106
6/0.25	0.891	0.367	0.018	0.800/0.107/0.093	0.806/0.089/0.105
8/0.25	0.918	0.417	0.020	0.762/0.127/0.111	0.773/0.098/0.129
10/0.25	0.914	0.407	0.019	0.772/0.120/0.108	0.780/0.096/0.124
12/0.25	0.907	0.394	0.019	0.791/0.111/0.098	0.793/0.098/0.109
8/0.05	0.939	0.480	0.023	0.706/0.152/0.142	0.714/0.116/0.170
8/0.10	**0.934**	**0.462**	**0.022**	**0.726/0.142/0.132**	**0.736/0.108/0.157**
8/0.15	0.931	0.452	0.022	0.733/0.139/0.128	0.742/0.106/0.152
8/0.20	0.929	0.444	0.021	0.738/0.139/0.123	0.748/0.107/0.145
10/0.20	0.944	0.492	0.023	0.697/0.155/0.148	0.705/0.117/0.178
12/0.20	0.943	0.481	0.023	0.707/0.151/0.142	0.715/0.115/0.170
10/0.10	0.952	0.519	0.025	0.666/0.172/0.162	0.679/0.126/0.195
10/0.15	0.950	0.514	0.024	0.675/0.166/0.159	0.685/0.123/0.192
10/0.05	**0.955**	**0.526**	**0.026**	**0.649/0.184/0.167**	**0.666/0.128/0.206**
12/0.05	0.966	0.568	0.028	0.607/0.206/0.187	0.635/0.131/0.234
12/0.10	0.964	0.556	0.027	0.625/0.195/0.180	0.646/0.129/0.225
12/0.15	0.962	0.554	0.027	0.632/0.193/0.176	0.651/0.130/0.218

Table 3.2 shows the consistent results that if a filter leads to higher ACFs and predictabilities, then it has relatively lower RMAE and hit ratios, and vice versa. We group the filters studied into three categories (separated by horizontal lines in the tables). The first group has the lowest ACFs/predictabilities but the highest RMAEs and hit ratios. The third group has the highest ACFs/predictabilities but the lowest RMAEs and hit ratios. And the second group is in between. Last, we select a representative filter from each group to be used for our financial time-series preprocessing. The selected filters are listed below.

- Filter 6/0.15: low ACF and predictability, good RMAE, high ratios w.r.t. high/low price range and buy/sell signals, and short lags incurred. It represents a filter with high cut-off frequency.

- Filter 10/0.05: high ACF and predictability, poor RMAE, low ratios w.r.t. high/low price range and buy/sell signals, and short lags incurred. It represents a filter with low cut-off frequency.

- Filter 8/0.10: relative high ACF and predictability, medium level of RMAE and ratios w.r.t. high/low price range and buy/sell signals, and short lags incurred. It represents a filter with a medium cut-off frequency.

It's clear from the ten tables that the relative performance of low-pass filters is very consistent for the ten sets of financial time-series studied. We select three filters here and leave them for ANN training to determine which filter is to be used.

The traditional low-pass filtering approach incurs lags for all frequency components, even though certain frequency bands are noise free. Another important characteristics of financial time series is its stationarity, which is left untouched in low-pass filtering.

3.3 Wavelet decomposition

As discussed in Section 3.1, financial time series is contaminated mainly by high frequency noise. Hence, when low-pass filtering is applied to reduce high-frequency noise, it may also remove useful information in the low-frequency end. To prevent such loss of information, we need to decompose a financial time series into multiple frequency channels and perform denoising adaptively for each channel if needed. Both wavelet decomposition and multi-band filter decomposition can decompose signals into multiple frequency bands [6, 28, 93, 60, 151, 35, 10, 111, 114]. In our work, we use wavelet decomposition, with the understanding that our approach can be extended to multi-band filtering as well.

We are interested to design a new preprocessing approach by combining wavelet decomposition and channel-specific denosing in order to outperform traditional low-pass filtering. In our case, a preprocessing approach outperforms a given low-pass filter if it generates similar preprocessed signals while incurring less lags as compared to the target low-pass filter. In this section, we use the three low-pass filters obtained in the previous section (Filter 6/01.5, Filter 8/0.10, and Filter 10/0.05) as our target low-pass filters.

3.3.1 Redundant wavelet decomposition

Wavelet decomposition is equivalent to multiple-band filtering with appropriate filters. It decomposes signals into multiple channels using a single filter (called the mother filter) at different scales. At the lowest scale (level 0), the mother filter is convolved directly with the signals. At a higher scale (level l), the mother filter is convolved with equally spaced signals (separated by 2^l data points). In a conventional wavelet decomposition, decimation is performed at each level of decomposition, and the mother filter is generally a non-causal FIR filter. For a set of signals $\{x_1, x_2, x_3, \cdots, x_{2n}\}$, if we keep either the set $\{x_1, x_3, \cdots, x_{2n-1}\}$ or the set $\{x_2, x_4, \cdots, x_{2n}\}$ and discard the other set, then we call the process decimation. The number of data elements after decimation is reduced by half. Further, because of the non-causal FIR filters used, there are delays incurred in the decomposed bands. The more bands are generated, the longer the delay the process causes.

A redundant wavelet decomposition, on the other hand, performs no decimation, and an asymmetric causal mother filter $g(k)$ is normally used. Here, we call a filter $g(k)$ causal if $g(k) = 0$ for all $k < 0$. Because of the use of a causal filter, the process does not need future data for filtering and incurs no delay. Also, when compared with conventional wavelet decomposition, redundant wavelet decomposition is *shift invariant*, which means that the original signal at time t can be reconstructed from and only from the decomposed signals at time t. Due to shift invariance, there is no error propagation when performing reconstruction from individual channels. Redundant wavelet decomposition, however, requires more memory, since each decomposed channel has the same number of data elements as the original signals. This is not an issue in time-series predictions because the number of data elements involved is small. Given the shift invariance property, redundant wavelet decomposition is, therefore, more suitable for time-series predictions than conventional wavelet decomposition.

The mother filters we use in our work belong to the B-spline filter family defined as follows:

$$g(l) = \begin{cases} \frac{1}{2^N} \begin{pmatrix} N \\ l \end{pmatrix}, & \text{if } l = 0, 1, 2, \cdots, N. \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

set $c_0(t) = x(t)$; select a low-pass filter $g(k)$;
set total number of bands to be $M + 1$;
for $j \leftarrow 1$ **to** M **do**
$c_j(t) = \sum_{l=0}^{T} g(l) c_{j-1}(t - 2^{j-1} l)$;
$w_j(t) = c_{j-1}(t) - c_j(t)$;
end_for

Figure 3.9: Algorithm for redundant wavelet decomposition using a mother filter of $g(k)$. The algorithm decomposes $x(t)$ into $M + 1$ bands: $w_1(t), w_2(t), \cdots, w_M(t)$ and $c_M(t)$ [111, 178, 179].

The B-spline family are compact, analytical, and easy to implement. They are good for decomposing discrete time series. In our work, we use the B_2, B_3, B_4, and B_6-spline filters (Table 3.1) in our wavelet decomposition.

Mathematically, a redundant wavelet decomposition decomposes the original signal $c_0(t)$ into M channels of high-frequency signals $w_j(t)$ $(j = 1, 2, \cdots, M)$ and a low frequency channel $c_M(t)$ as follows.

$$c_0(t) = c_M(t) + \sum_{j=1}^{M} w_j(t). \tag{3.4}$$

The signals $w_j(t)$ and $c_M(t)$ are computed iteratively as depicted in Figure 3.9.

To further illustrate redundant wavelet decomposition, consider two wavelet decomposition scenarios with causal filters and non-causal filters.

First, when a non-causal B_2-spline mother filter $(g(-1) = 0.25, g(0) = 0.5, g(1) = 0.25)$ is used, Figure 3.10 shows 2 units of delay in w_1, 6 units of delay in w_2, and 14 units of delay in w_3 and c_3.

Second, when a causal B_2-spline mother filter $(g(0) = 0.25, g(1) = 0.5, g(2) = 0.25)$ is used, Figure 3.11 shows no delay in all channels. The main difference between Figures 3.10 and 3.11 is that the latter has no delay incurred in the redundant wavelet transformation using causal filters. Because of this property, we will only use redundant wavelet decomposition with causal filters in our work.

Figure 3.12 demonstrates the results using a causal B_3-spline wavelet decomposition and $M = 2$. It's clear that the low-pass channel is very smooth, that the high frequency channel (Channel H) is very noisy, and that Channel LH is in between. Further, the low-pass channel (Channel LL) dominates the other two channels in terms of magnitude.

As mentioned before, wavelet transformation is equivalent to multi-band filtering. In multi-band filtering, we obtain the frequency response for each band readily by performing an FFT using the filter coefficients for each channel. To understand each channel in the wavelet decomposition, we would like to derive the frequency response and the frequency coefficients for each channel. Consider the recursive relationship between $c_j(t)$

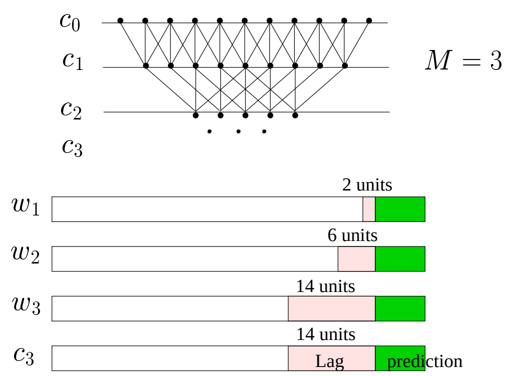


Figure 3.10: Redundant wavelet decomposition using symmetric non-causal B_2-spline filters. It causes different lags in different channels. c_0 is the original signal to be decomposed. c_1, c_2 and c_3 are low-pass signals obtained using a B_2-spline filter convolved with the previous channel. The links between two consecutive channels indicate which three data elements in Channel l are used to obtain the new data element in Channel $l + 1$. w_l is computed as $w_l(t) = c_{l+1}(t) - c_l(t)$. Because of the use of a non-causal filter, c_l has delay over c_{l-1}, and therefore, w_l also has delay over c_{l-1} or w_{l-1}. The lightly shaded segments in each bar indicates the number of lags incurred due to the use of symmetric mother filters, and the darkly shaded segment shows the prediction horizon.

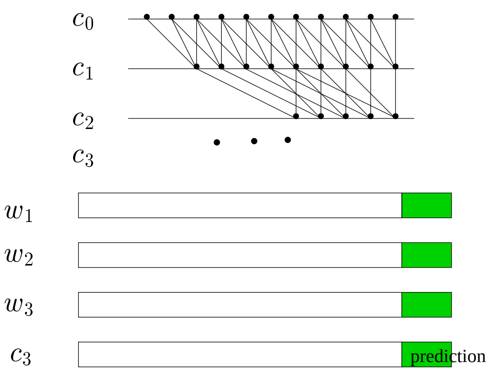


Figure 3.11: Redundant wavelet decomposition using an asymmetric causal B_2-spline mother filter. It does not incur any lag for all channels. See Figure 3.10 for an explanation of the graph.

and $c_{j+1}(t)$ described in Figure 3.9.

$$\begin{aligned}
c_{j+1}(t) &= \sum_{l_j=0}^{T} g(l_j) c_j(t - 2^j l_j) \\
&= \sum_{l_j=0}^{T} g(l_j) \sum_{l_{j-1}=0}^{T} g(l_{j-1}) c_{j-1}(t - 2^j l_j - 2^{j-1} l_{j-1}) \\
&= \sum_{l_j, l_{j-1}=0}^{T} g(l_j) g(l_{j-1}) c_{j-1}(t - 2^j l_j - 2^{j-1} l_{j-1}) \\
&= \cdots \\
&= \sum_{l_j, l_{j-1}, \cdots, l_0=0}^{T} g(l_j) g(l_{j-1}) \cdots g(l_0) c_0 \left(t - \sum_{k=0}^{j} 2^k l_k\right). \\
&= \sum_{l_j, l_{j-1}, \cdots, l_0=0}^{T} \left(\prod_{m=0}^{j} g(l_m)\right) c_0 \left(t - \sum_{k=0}^{j} 2^k l_k\right).
\end{aligned} \tag{3.5}$$

Eq (3.5) shows that the most historical and most recent c_0's involved are $c_0(t - \sum_{k=0}^{j} 2^k T) = c_0(t - (2^{j+1} - 1)T)$ and $c_0(t)$, respectively. Hence, the filter coefficients for

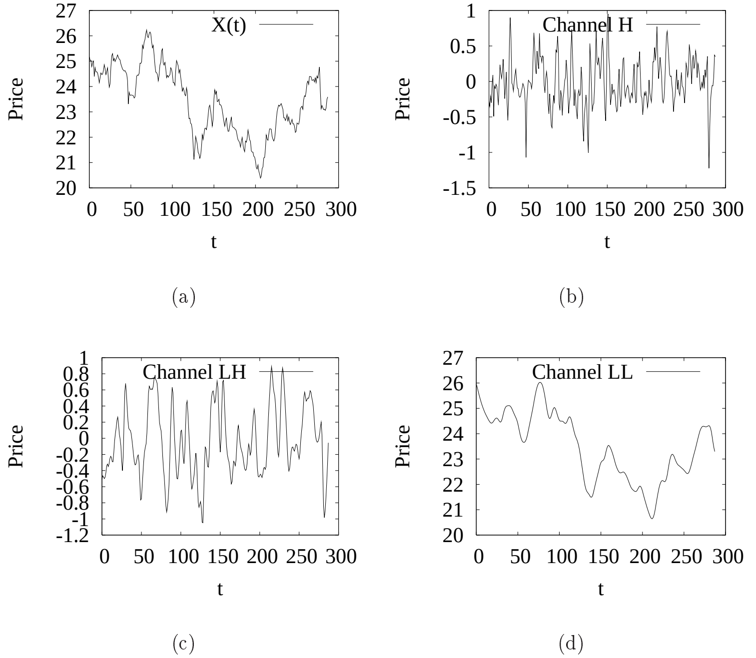


Figure 3.12: Redundant wavelet decomposition using B_3-spline causal mother filter. (a) Raw time series: average of daily low/high prices for IBM stock; (b) Channel H: $w_1(t)$; (c) Channel LH: $w_2(t)$; (d) Low-pass channel: $c_2(t)$.

channel j have non-zero values from $g(0)$ to $g((2^j-1)T)$. To compute individual filter coefficients, e.g. $g_j(k)$, one should expand the right side of (3.5) to find the coefficient for the term $c_0(t-k)$. We have:

$$c_{j+1}(t) \quad = \quad \sum_{k_0=0}^{(2^{j+1}-1)T} h(k)c_0(t-k), \tag{3.6}$$

with $h(k)$ being the causal filter obtained though the method stated above.

After obtaining $g_j(k)$, one can calculate the frequency response by either using Matlab function *freqz* [99] or apply discrete FFT. The frequency response for each channel for the redundant wavelet decomposition using, respectively, the B_3, B_4, and B_6-spline mother filters are plotted in Figure 3.13. When the B_3-spline filter is used, Channel LL retains a considerable amount of frequency components of 0.2, and Channel LH contains a significant amount of frequency components as high as 0.4. When the B_4-spline filter is used, Channel LL takes very little frequency components around 0.2, and Channel LH allows very little frequency components above 0.3. When the B_6-spline filter is used, the frequency responses for Channels LL and LH are shifted further towards the lower frequency end.

3.3.2 Transformations on low-frequency channels

Figure 3.12(d) clearly shows that the low-frequency channel has a long term trend, which is nonstationary. Since the ANN model we have developed is more suitable for chaotic time series, we need to transform the nonstationary signals into near chaotic signals before ANN learning can be performed.

In order to transform nonstationary low-pass signals into near chaotic signals, we perform the following pattern-wise transformation. Assume the input data are $(x(t-n+$

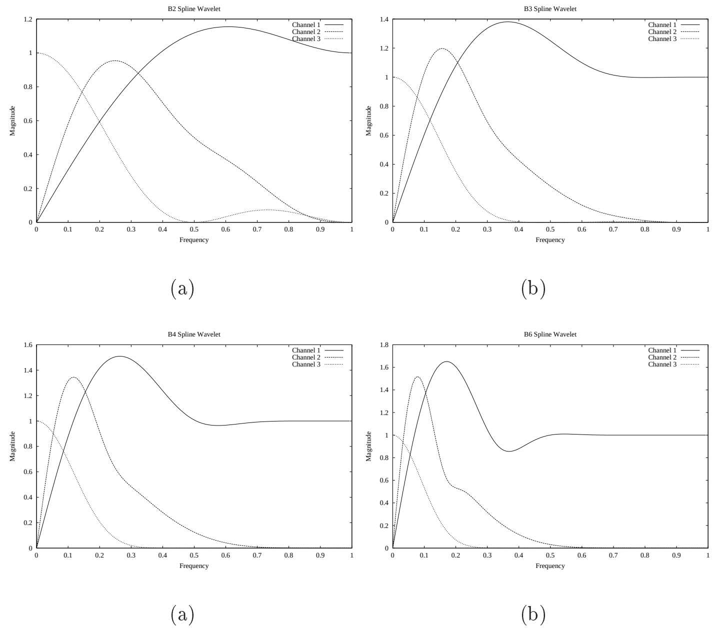


Figure 3.13: Frequency response of the three bands obtained by wavelet decomposition using a B_2-, B_3-, B_4-, and B_6-spline mother filters in Panels (a), (b), (c), and (d) respectively.

1), $x(t-n+2), \cdots, x(t))$ and the desired output is $x(t+1)$. We define:

$$\begin{aligned} \mu &= \frac{1}{n}\sum_{i=n-1}^{0} x(t-i), \\ a &= \min_{0 \le i \le n-1}\{x(t-i)\}, \\ b &= \max_{0 \le i \le n-1}\{x(t-i)\}, \\ s &= \max\{\mu - a, b - \mu\}. \end{aligned} \tag{3.7}$$

Then the transformed signals are given by

$$y(t-i) = (x(t-i) - \mu)/s, \qquad \text{for all} \quad i = n-1, n-2, \cdots, 0, -1. \tag{3.8}$$

The transformation in (3.8) confines the transformed input patterns in the range of $[-1, 1]$ with a mean of 0. Note that the transformation is completely determined by the input patterns, and the transformed desired output may be outside the range $[-1, 1]$. Figure 3.14 illustrates the transformation. It shows that two patterns of the same shape will be transformed to identical patterns in term of the resulting transformed values, regardless of the the absolute level (μ) of the input pattern and the magnitude of the variations within the input patterns (captured by s). What really matters is the shape of the input pattern.

Figure 3.15 presents the desired output for each t before and after the transformation for Citigroup, IBM, and Exxon-Mobil stock prices. Clearly, the nonstationarity, especially the long term trend, is removed. The resulting signals resemble MacKey-Glass (30) (Fig 5.3(b)) to some degree. The transformed signals can be trained and predicted by ANN models developed for chaotic time series.

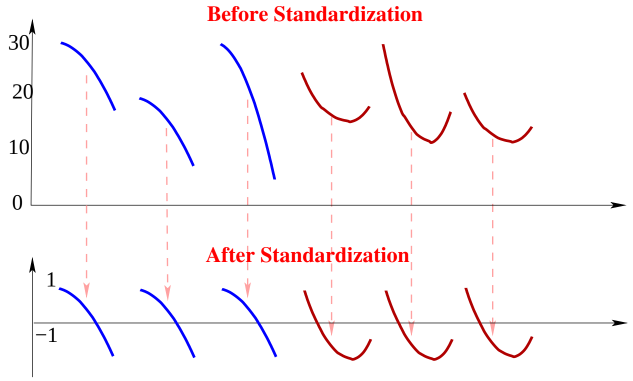
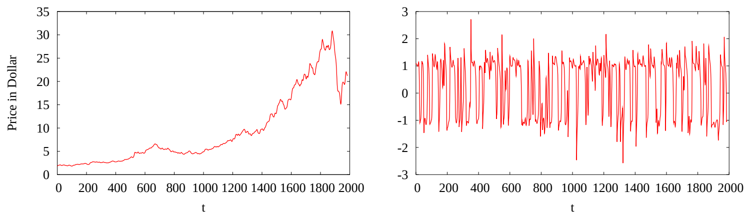


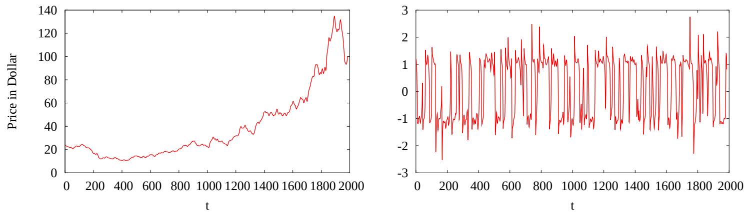
Figure 3.14: Transforming a nonstationary series into a chaotic series by performing standardization on input patterns.

3.3.3 Denoising on high-frequency channels

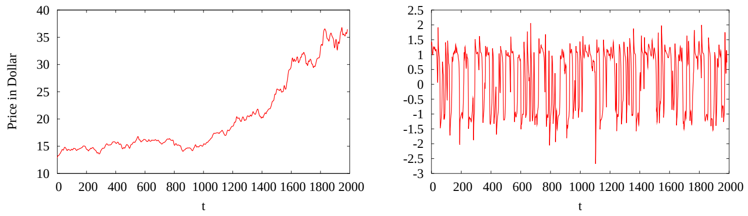
As mentioned above, there is high-frequency noise in financial time series. The presence of noise makes training more difficult, and generalization even more difficult because noise learned during training cannot be generalized. Hence, denoising on each high-frequency channel is needed. Since each high-frequency channel may have a different noise level, we need to find an appropriate filter for each. In our decomposed signals, the most important band, i.e., the low frequency band, does not need any filtering, and, therefore, incurs no additional delay. On the other hand, high-frequency channels may require a low-pass filter with a large number of taps, and, therefore, incur more delays. The preprocessing approach of combining wavelet decomposition and low-pass filtering is illustrated in Fig 3.16. In our case, Channel m, the low-frequency channel, does not require any low-pass filtering.



a) Citigroup



b) IBM



c) Exxon-Mobil

Figure 3.15: Signals before (left panels) and after (right panels) performing input pattern standardization for a) Citigroup, b) IBM, and c) Exxon-Mobil.

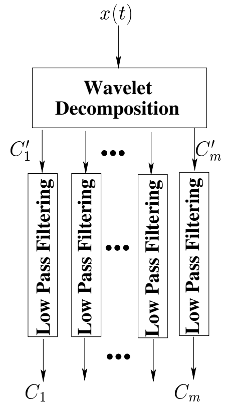


Figure 3.16: Wavelet decomposition is performed first, then different low-pass filters can be applied to the decomposed bands according to the noise level in each band.

Figure 3.17 shows the filter coefficients for each channel in a 3-channel wavelet decomposition with a B_3-spline mother filter, and the coefficients of the low-pass filter for each high-frequency channel.

After individual channels are low-pass filtered, the overall preprocessed signal can be reconstructed by summing the preprocessed signals from individual channels. The resulting signals can be obtained by a single equivalent composite filter. The composite filter equivalent to the process in Figure 3.17 is plotted in Figure 3.18. The procedure of obtaining the coefficients of the composite filter is given as follows.

Step. 1 Let the original signal be $x(t)$, and the coefficients of the wavelet decomposed signal in Channel k be

$$C_k(t) = \sum_{l=m_k}^{n_k} g_k(l)x(t-l), \;\; k = 1, 2, \cdots, B \tag{3.9}$$

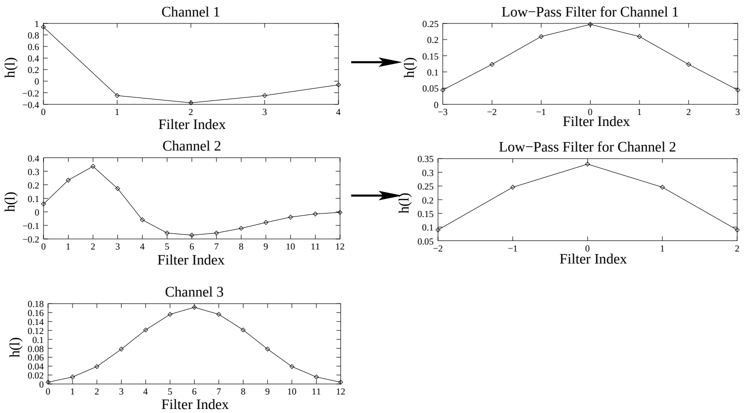


Figure 3.17: Wavelet decomposition and low-pass filtering on channels H and LH. The left panels plot the filter coefficients for each wavelet channel. The right panels plot the coefficients of the low-pass filters applied to the two high-frequency channels. The horizontal axes represent the indices for filters and the vertical axes are for filter coefficients.

Step. 2 Low-pass filtering on the decomposed Band k yields:

$$y_k(t) = \sum_{L=M_k}^{N_k} G_k(L)C(t-L) = \sum_{L=M_k}^{N_k} \sum_{l=m_k}^{n_k} G_k(L)g_k(l)x(t-l-L). \tag{3.10}$$

Step. 3 Unifying index for all B bands by setting

$$\begin{aligned} & M = \min_k\{M_k\}, m = \min_k\{m_k\}, N = \max_k\{N_k\}, n = \max_k\{n_k\} \\ & \hat{G}_k(L) = \begin{cases} G_k(L), & \text{if } L \in [M_k, N_k] \\ 0, & \text{otherwise} \end{cases} \\ & \hat{g}_k(l) = \begin{cases} g_k(l), & \text{if } l \in [m_k, n_k] \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.11}$$

Step. 4 Adding the filtered signals from each band to construct composite signal, we have:

$$S(t) = \sum_{k=1}^{B} y_k(t) = \sum_{k=1}^{B} \sum_{L=M}^{N} \sum_{l=m}^{n} \hat{G}_k(L)\hat{g}_k(l)x(t-l-L) \tag{3.12}$$

Step. 5 By defining composite filter, we have:

$$S(t) = \sum_{j=M+m}^{N+n} h(j)x(t-j) \tag{3.13}$$

Step. 6 Comparing (3.12) with (3.14), one has:

$$h(j) = \sum_{B=1}^{B} \sum_{L+l=j} \hat{G}(L)\hat{g}(l), \;\; L \in [M, N], l \in [m, n]. \tag{3.14}$$

Figure 3.18 shows the coefficients of the equivalent composite filter corresponding to the wavelet decomposition and low-pass filtering processes depicted in Figure 3.17.

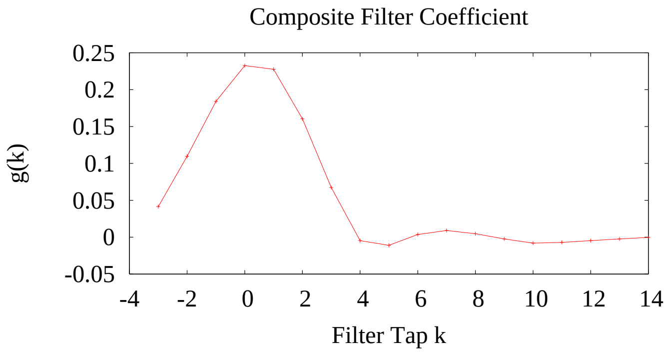


Figure 3.18: Coefficients of the equivalent composite filter for the preprocessing process illustrated in Figure 3.17.

After the composite filter coefficients have been obtained, the corresponding frequency response (both magnitude and phase) of the composite filter can be obtained using FFT or the Matlab $freqz$ function.

The magnitude (resp. phase) of the frequency response for the composite filter displayed in Figure 3.18 is plotted in Figure 3.19 (resp, Figure 3.20) along with Filter 6/0.15. Note that the phase distortion at the high-frequency range is not critical because the magnitude of the frequency response is small there. In comparison, Filter 6/0.15 has a constant 0 phase angle. The phase distortion is due to the introduction of nonlinearity in the composite filter. When different low-pass filtering is applied in different channels, the linearity is destroyed when the multiple channels are combined. Figure 3.20 also shows that phase shifts happen mainly at the high frequency part. These have little effects on financial time-series preprocessing, as there is very little energy associated with

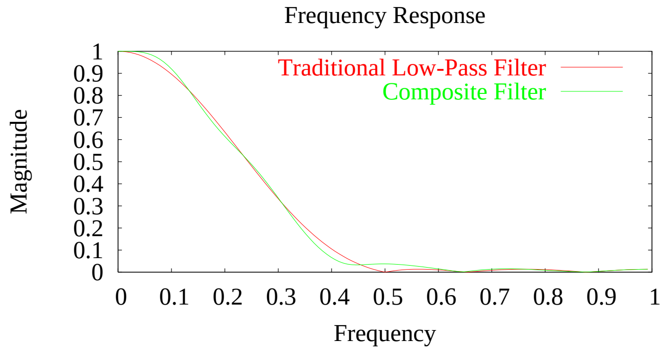


Figure 3.19: Magnitude of the frequency response for the composite filter.

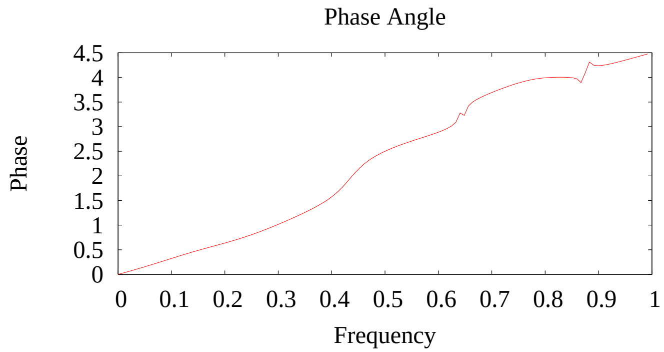


Figure 3.20: Phase angle of the frequency response for the composite filter.

high-frequency components (Figure 3.7). To measure the effect of nonlinearity introduced in the composite filter, we compare in Figures 3.21 to 3.23 the optimal linear filter with zero phase shift and the composite filter.

To show that the non-zero phase angle is not critical, we compare the preprocessed signals using the composite filter and those using the corresponding optimal low-pass filter with zero phase angle. An optimal low-pass filter with zero phase angle is one that minimizes the mean square error, when compared to the preprocessed data using the composite filter. This optimal filter can be obtained by performing a discrete FFT on the composite filter coefficients, setting all phases to zero, and performing an inverse FFT in order to obtain a set of coefficients of the same number as that of the original filter. The top three panels in Figures 3.21 – 3.23 show the composite filter coefficients, along with the corresponding optimal filters for the three composite filters used in our work. The preprocessed data using the optimal low-pass filters are very similar to that obtained by using wavelet decomposition with low-pass filtering. More importantly, when the high/low price range is considered, the fractions of preprocessed data falling within the low-high range using these two methods are almost identical.

Even though low-pass optimal FIR filters with zero phase are attractive in terms of noise reduction, the corresponding composite filters obtained through wavelet decomposition and channel-specific low-pass filtering can achieve almost an equivalent low-pass signal. Using a single low-pass filter, delay is incurred in every frequency component. By using redundant wavelet decomposition along with low-pass filtering on individual channels, we can discriminatively perform preprocessing on different bands and attain less delays in some channels. For example, the band with the lowest frequency has the highest predictability and does not require further low-pass filtering. As a result, there

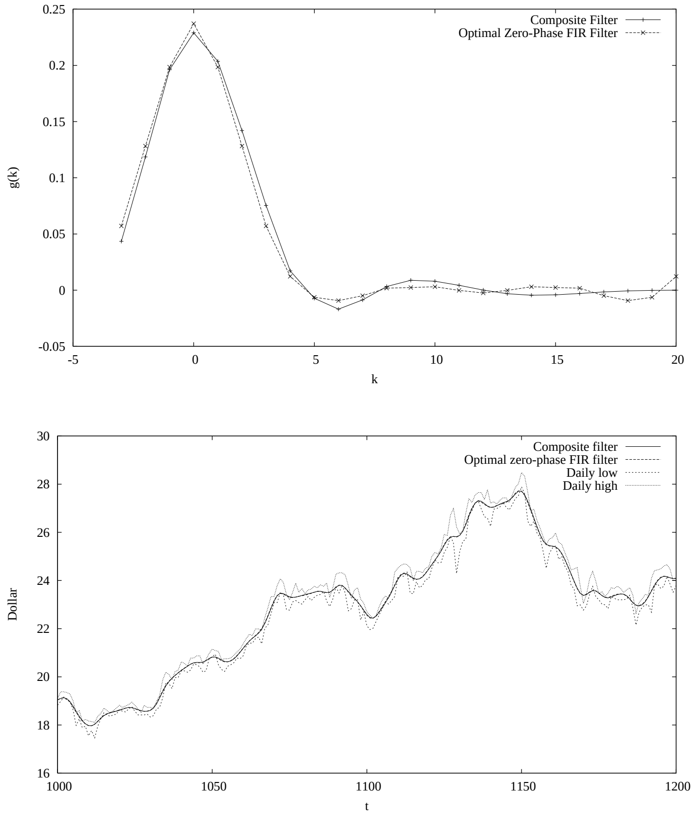


Figure 3.21: Coefficients of composite filter B4_63_43 and its corresponding optimal low-pass FIR filter with zero phase (upper panel) and their preprocessed IBM stock prices.

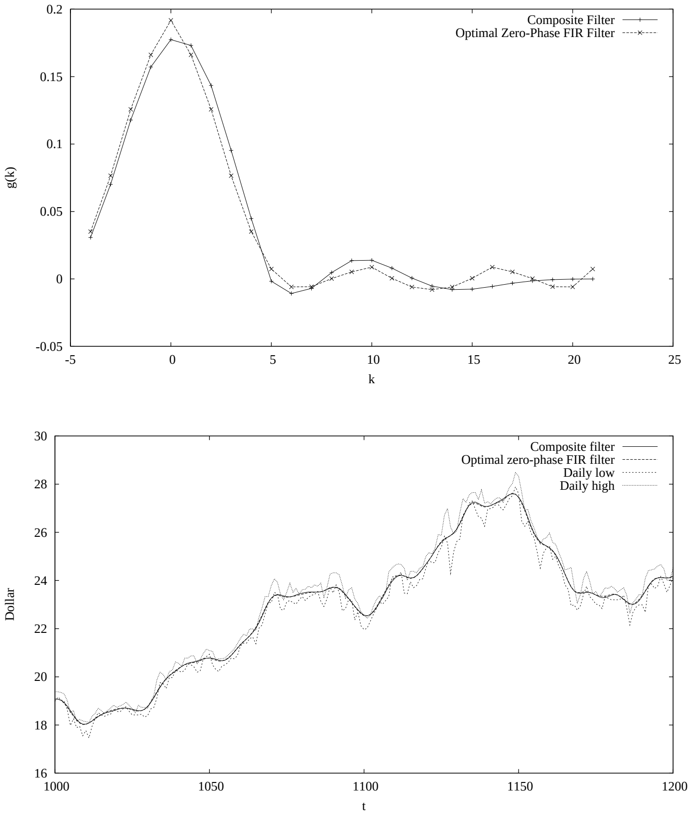


Figure 3.22: Coefficients of composite filter B4_82_62 and its corresponding optimal low-pass FIR filter with zero phase (upper panel) and their preprocessed IBM stock prices.

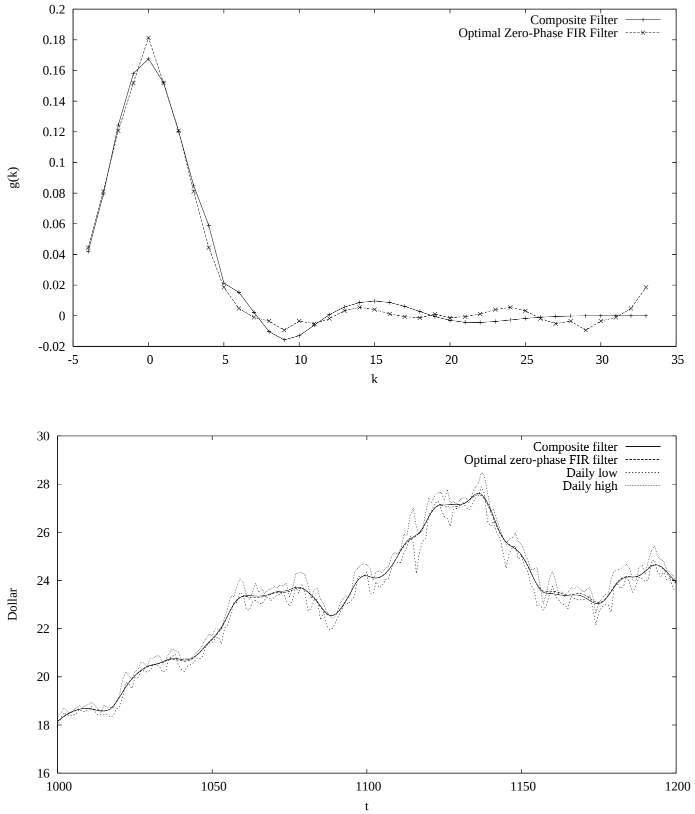


Figure 3.23: Coefficients of composite filter B6_81_61 and its corresponding optimal low-pass FIR filter with zero phase (upper panel) and their preprocessed IBM stock prices.

will be no additional delays incurred. Likewise, Channel LH has relatively lower frequency than Channel H, and can be low-pass filtered with a smaller number of taps and a higher cut-off frequency than the low-pass filter used for Channel H. As a result, less delays will be incurred in Channel LH than those in Channel H. In contrast, if a single low-pass filter is used, then the channel will incur a constant delay.

3.3.4 Parameter selection

For a given low-pass filter, we need to find an appropriate set of parameters for preprocessing, which combines wavelet decomposition and low-pass filtering, in order to incur less lags while achieving similar preprocessed signals as those of the low-pass filter. The parameters in our preprocessing approach include the mother filter, the number of channels to be decomposed, and the low-pass filters used for each channel.

3.3.4.1 Number of channels

When a mother filter is used to decompose a time series, we notice that, in order to obtain the $M+1$-channel decomposition from the M-channel decomposition, we only need to further decompose the low-frequency channel, and the high-frequency channels are unchanged. Hence, if the low-frequency channel in an M-channel decomposition is predictable, then there is no need for further decomposition. At the same time, we need to make sure the high-frequency channels are predictable as well. For the high-frequency channels, prediction has to overcome lags introduced by the additional low-pass filtering. A high-frequency channel may be not predictable because long lags are introduced when using a low-pass filter with a large number of taps. It may also be unpredictable because a substantial amount of noise exists due to the use of a short

low-pass filter. We leave the selection of low-pass filters for high-frequency channels for further denoising to Section 3.3.4.2. Here, we only examine the low-frequency channel to determine the number of channels needed.

There are two observations. First, if the predictions on the low-frequency channel in an M-channel decomposition is satisfactory, there is no need to further decompose the signals into $M+1$ channels. This is true because further decompositions only decompose the low-frequency channel. Since the low-frequency channel is already predictable, there is no need to further decompose it. Second, the higher-order spline filter leads to easy prediction of the low-frequency channel. This is true because a high-order spline filter leads to a lower and sharper cut-off frequency. Based on the two observations, we only need to find the minimum number of channels (denoted as M_{min}) for our wavelet decomposition in order to make predictions on the low-frequency channel satisfactory when B_2 is used as the mother filter. An M_{min}-channel decomposition using B_2, B_3, B_4, and B_6 spline mother filters will all generate a predictable low-frequency channel.

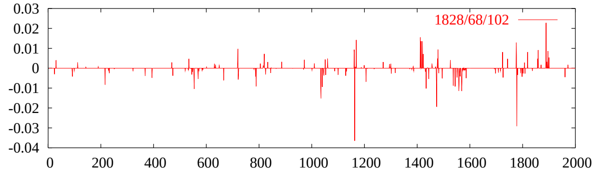
We have performed experiments on low-frequency channels in three wavelet decomposition instances. The first one is a 2-channel decomposition with a B_2-spline mother filter. The second one uses a 3-channel decomposition, also with a B_2-spline mother filter. The third one uses a B_4-spline mother filter and decomposes signals into three channels.

To predict the signals in a low-frequency channel, we first transform them by the pattern-wise standardization method discussed in Section 3.3.2. We then train the transformed signals using the ANN model developed in Chapter 4. Figure 3.24 shows the range errors of the predictions of Horizon 1 for the low-frequency channel in three cases, and Figure 3.25 presents the predicted values for these three cases. One can clearly see that when we increase the number of channels from two to three, the quality in predicting the low-frequency channel improves. When switching from the B_2-spline mother filter to the B_4-spline mother filter, there is no substantial improvement. Therefore, the 3-channel decomposition using any mother filters of B_2-, B_3-, B_4-, and B_6-spline is sufficient to predict the low-pass channel. We will, therefore, use the 3-channel decomposition throughout our work. Next, we need to find out how to select low-pass filter parameters in order to make high-frequency channels predictable.

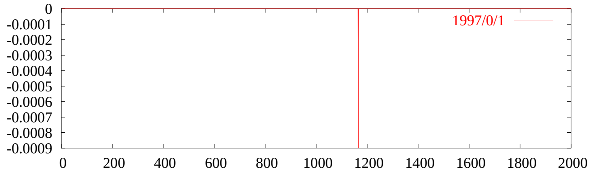
3.3.4.2 Filter parameters

Our goals in preprocessing is to make the composite filter having similar frequency response as the target low-pass filter, while incurring less delays at least for some frequency bands. In order to find a good combination of wavelets and low-pass filters, we have tried different combinations of mother filters and low-pass filters and identified several sets of parameters that can generate similar frequency response as the target low-pass filter. The resulting designs are listed in Table 3.3. For the mother filter, we use the B_3, B_4, and B_6 spline filters. For Channel H, we apply low-pass filtering with 6 to 10 taps and a cut-off frequency of 0.05 to 0.15, whereas low-pass filters with 4 to 8 taps, and 0.05 to 0.15 cut-off frequency are applied to Channel LH. Since Channel LL has little noise, we leave it untouched. Frequency responses (both magnitude and phase) for the composite filters listed in the table are plotted in Figures 3.26 to 3.28.

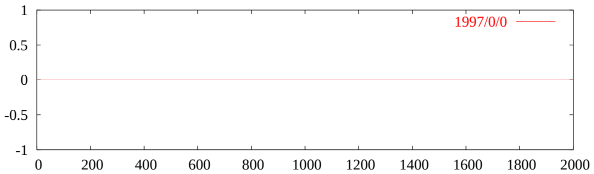
We have collected the performance for the 11 combinations listed in Table 3.3. The frequency response (magnitude and phase response) of those composite filters along with the three target low-pass filters are plotted in Figures 3.26 to 3.28. The performance for the ten benchmark stocks are presented in Table 3.4. Again, we can see the relative performance for each combination is very consistent across different stocks. Based on



a) 2-channel decomposition using B_2-spline mother filter

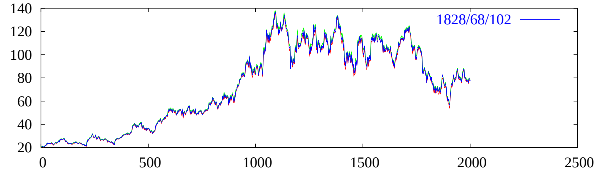


b) 3-channel decomposition using B_2-spline mother filter

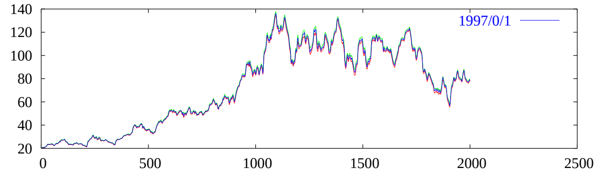


c) 3-channel decomposition using B_4-spline mother filter

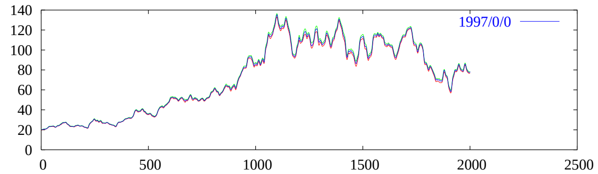
Figure 3.24: Range error in predicting the low-frequency channel of IBM stock price at horizon of one using different wavelet decomposition. The numbers $a/b/c$ in the legends indicate the number of predictions in/above/below the daily low-high price range.



a) 2-channel decomposition using B_2-spline mother filter



a) 3-channel decomposition using B_2-spline mother filter



a) 3-channel decomposition using B_4-spline mother filter

Figure 3.25: The predicted values of the low-frequency channel of IBM stock prices at horizon of one using different wavelet decompositions. The numbers $a/b/c$ in the legends indicate the number of predictions in/above/below the daily low-high price range.

Table 3.3: Combinations of wavelet mother filter for wavelet decomposition and low-pass filters used for channel H and LH. The notation listed in column 1 will be used in Table 3.4. The second column lists the mother filter used for the wavelet decomposition. The third column tells which low-pass filter is used for Channel H; the last column shows the low-pass filter used for Channel LH. Channel LL is left unchanged.

Wavelet-Filter	Mother Filter	Channel H		Channel LH	
		Taps	f_0	Taps	f_0
B3_63_43	B_3 Spline	6	0.15	4	0.15
B4_63_43	B_4 Spline	6	0.15	4	0.15
B2_63_43	B_2 Spline	6	0.15	4	0.15
B3_82_62	B_3 Spline	8	0.10	6	0.10
B4_82_62	B_4 Spline	8	0.10	6	0.10
B2_82_62	B_2 Spline	8	0.10	6	0.10
B4_82_42	B_4 Spline	8	0.10	4	0.10
B3_81_61	B_3 Spline	8	0.05	6	0.05
B3_81_41	B_3 Spline	8	0.05	4	0.05
B6_81_61	B_6 Spline	8	0.05	6	0.05

the similar criteria used in previous section (Section 3.2), we select the following three groups of wavelets and filters that correspond to the three low-pass filters (Filter 6/0.15, 8/0.10, and 10/0.05) selected in Section 3.2:

- The first three sets of the 15 combinations in Table 3.4 cover low-pass filter 6/0.15. These combinations have medium ACF, predictability, RMAE and hit ratios with respect to daily low-high price ranges and buy/sell signal.

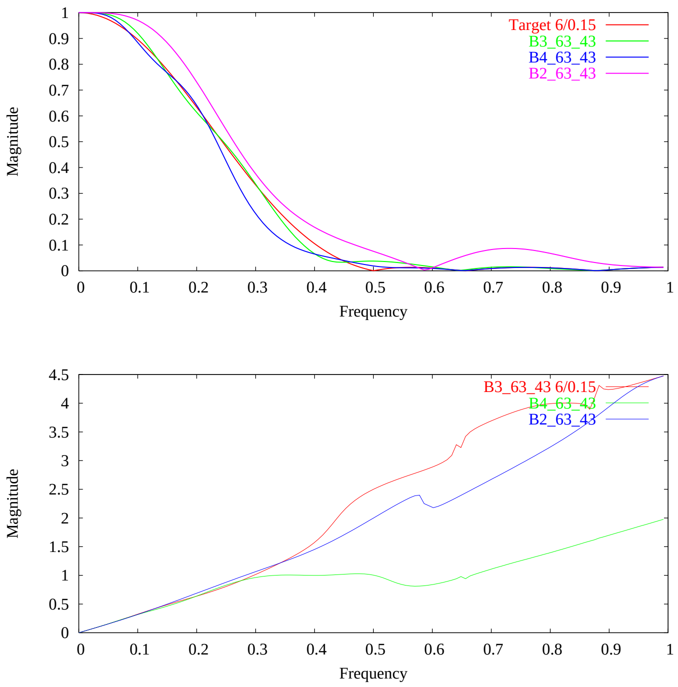


Figure 3.26: Frequency response of the composite filter compared with the 6 tap low-pass filter with a cut-off frequency of 0.15π. Upper panel: magnitude response. Lower panel: phase shift.

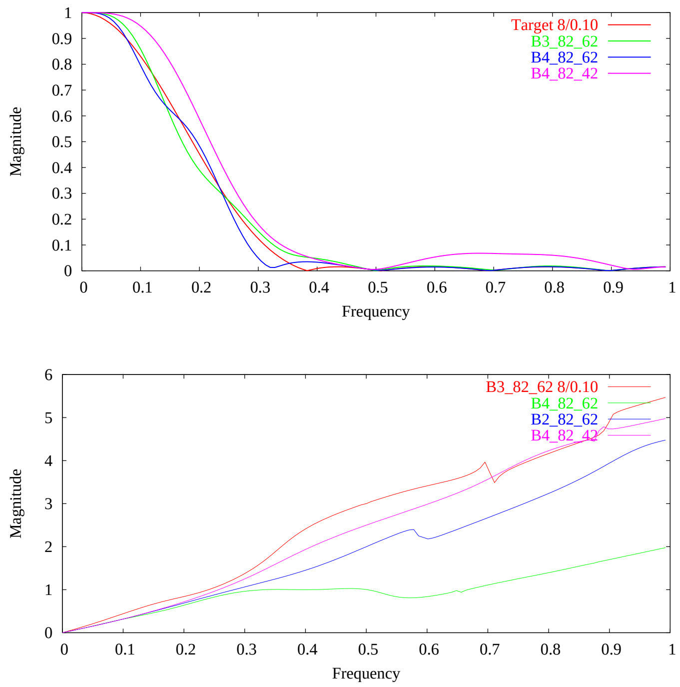


Figure 3.27: Frequency response of the composite filter compared with the 8 tap low-pass filter with a cut-off frequency of 0.10π. Upper panel: magnitude response. Lower panel: phase shift.

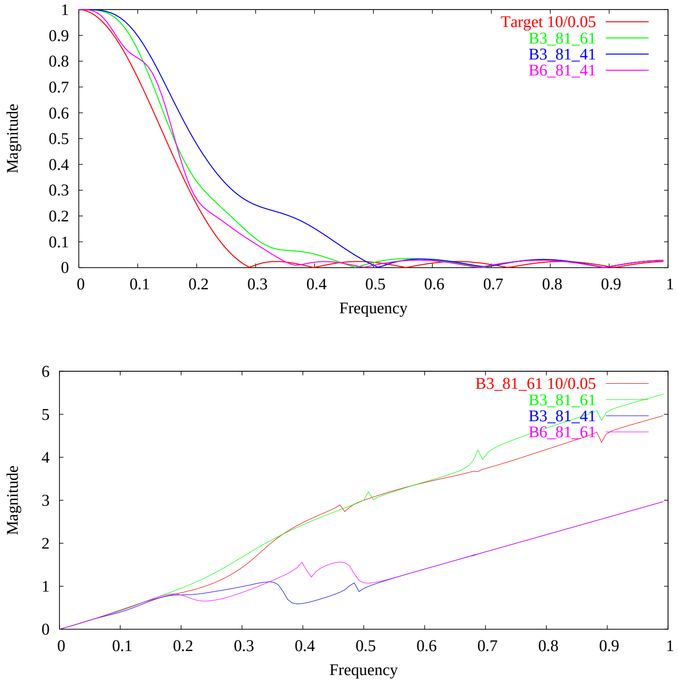


Figure 3.28: Frequency response of the composite filter compared with the 10 tap low-pass filter with a cut-off frequency of 0.05π. Upper panel: magnitude response. Lower panel: phase shift.

- The next four of the 15 combinations in Table 3.4 cover low-pass filter 8/0.10. These combinations have low ACF, predictability, RMAE, and high hit ratios with respect to daily low-high price ranges and buy/sell signal.

- The last four of the 15 combinations in Table 3.4 cover low-pass filter 10/0.05. These combinations have high ACF, predictability, and RMAE, but low hit ratios with respect to daily low-high price ranges and buy/sell signal.

Since our predictors need to overcome lags, and each combination may incur a different amount of lags, our measurements based on an ideal perfect predictor may not translate into realizable accurate predictions. Therefore, we have to rely on our ANN predictors to select the best combination.

3.4 Summary

In this chapter, we have studied the preprocessing of financial time series by examining the autocorrelation and predictability measures. We have further studied preprocessing using traditional low-pass filtering and have showed trade-offs between noise reduction and information loss. Based on the autocorrelation, predictability, RMAE, and hit ratios with respect to price range, we have selected three representative low-pass filters with low/middle/high cut-off frequencies as our benchmarks in preprocessing. As traditional low-pass filters incur lags in all frequency components, it leads us to design a new preprocessing approach by applying different preprocessing for different frequency channels. To this end, we have developed a new preprocessing approach by combining wavelet decomposition and low-pass filtering. The signals are first decomposed into multiple channels of different cut-off frequencies. For the channel with low cut-off frequency, we only need to handle its nonstationarity using pattern-wise standardization. For other channels, we perform channel-specific denoising by finding appropriate low-pass filters. The new technique incurs no lag for the low-frequency channel, while incurring as little lags as possible for each high-frequency channel. To compare with traditional low-pass filtering, we have identified three groups of configurations that lead to similar preprocessed signals as the ones generated by the target low-pass filters. We compare the prediction performance of three representative configurations in Chapter 6.

Table 3.4: Performance of preprocessed stock-prices time series using different wavelet decompositions along with channel-specific low-pass filtering. The notations for the filters are explained in Table 3.3. ACF indicates autocorrelation. The column labelled by P records the predictability measures. "Hit Ratios" shows hit/over-hit/under-hit ratios w.r.t. the daily low/high price ($\alpha_0/\alpha_+/\alpha_-$) defined by (3.1) in Section 3.2.1. "Buy/Sell" shows the ratios of zero/positive/negative errors w.r.t. potential buy/Sell signals ($\beta_0/\beta_+/\beta_-$) defined by (3.2) in Section 3.2.1.

American Airline (AMR)					
Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.910	0.362	0.017	0.746/0.130/0.123	0.746/0.042/0.212
B4_63_43	0.921	0.410	0.018	0.740/0.133/0.127	0.740/0.038/0.222
B2_63_43	0.887	0.328	0.016	0.769/0.115/0.116	0.769/0.045/0.186
B3_82_62	0.946	0.454	0.021	0.670/0.167/0.163	0.670/0.041/0.289
B4_82_62	0.946	0.468	0.022	0.665/0.169/0.166	0.665/0.038/0.298
B2_82_62	0.924	0.403	0.021	0.676/0.168/0.155	0.676/0.047/0.277
B4_82_42	0.940	0.453	0.023	0.646/0.180/0.174	0.646/0.040/0.314
B3_81_61	0.951	0.474	0.022	0.655/0.174/0.171	0.655/0.038/0.307
B3_81_41	0.920	0.419	0.022	0.654/0.177/0.169	0.654/0.046/0.300
B6_81_61	0.955	0.496	0.022	0.653/0.178/0.168	0.653/0.038/0.309

continued on next page

continued from previous page

Citigroup Stock

Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.901	0.357	0.010	0.754/0.120/0.126	0.754/0.055/0.191
B4_63_43	0.910	0.379	0.010	0.750/0.120/0.129	0.750/0.053/0.197
B2_63_43	0.882	0.340	0.010	0.783/0.102/0.115	0.783/0.055/0.162
B3_82_62	0.938	0.452	0.012	0.674/0.160/0.166	0.675/0.053/0.272
B4_82_62	0.936	0.446	0.012	0.673/0.161/0.166	0.673/0.051/0.276
B2_82_62	0.916	0.390	0.012	0.689/0.154/0.157	0.689/0.055/0.256
B4_82_42	0.932	0.416	0.013	0.649/0.174/0.178	0.649/0.052/0.300
B3_81_61	0.944	0.463	0.013	0.651/0.171/0.178	0.651/0.052/0.297
B3_81_41	0.915	0.390	0.013	0.657/0.168/0.174	0.657/0.055/0.287
B6_81_61	0.947	0.474	0.013	0.647/0.174/0.179	0.647/0.046/0.307

General Electric (GE)

Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.909	0.360	0.010	0.808/0.088/0.104	0.808/0.031/0.161
B4_63_43	0.911	0.386	0.010	0.804/0.089/0.107	0.804/0.031/0.165
B2_63_43	0.889	0.318	0.009	0.833/0.072/0.095	0.833/0.037/0.131
B3_82_62	0.940	0.455	0.012	0.732/0.123/0.145	0.732/0.034/0.233
B4_82_62	0.933	0.464	0.012	0.720/0.129/0.151	0.720/0.035/0.245
B2_82_62	0.919	0.404	0.012	0.736/0.119/0.144	0.736/0.036/0.228
B4_82_42	0.930	0.449	0.013	0.702/0.137/0.161	0.702/0.031/0.268
B3_81_61	0.945	0.480	0.013	0.710/0.132/0.158	0.710/0.037/0.253
B3_81_41	0.920	0.383	0.013	0.712/0.129/0.159	0.712/0.035/0.253
B6_81_61	0.947	0.480	0.013	0.699/0.139/0.162	0.699/0.032/0.269

continued on next page

continued from previous page

International Business Machine (IBM)

Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.920	0.379	0.009	0.771/0.107/0.122	0.771/0.044/0.185
B4_63_43	0.929	0.403	0.009	0.763/0.113/0.124	0.763/0.043/0.193
B2_63_43	0.903	0.345	0.009	0.795/0.097/0.108	0.795/0.047/0.159
B3_82_62	0.951	0.460	0.011	0.684/0.148/0.168	0.684/0.044/0.272
B4_82_62	0.952	0.465	0.011	0.679/0.152/0.168	0.679/0.044/0.276
B2_82_62	0.935	0.409	0.011	0.697/0.145/0.158	0.697/0.046/0.257
B4_82_42	0.946	0.452	0.012	0.659/0.162/0.179	0.659/0.048/0.293
B3_81_61	0.955	0.479	0.011	0.669/0.154/0.177	0.669/0.043/0.289
B3_81_41	0.929	0.410	0.011	0.672/0.159/0.170	0.671/0.050/0.278
B6_81_61	0.960	0.471	0.012	0.667/0.159/0.174	0.667/0.038/0.295

Mentor (MNTR)

Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.904	0.349	0.010	0.826/0.092/0.082	0.826/0.016/0.158
B4_63_43	0.914	0.403	0.010	0.816/0.096/0.088	0.815/0.013/0.171
B2_63_43	0.879	0.324	0.009	0.849/0.079/0.072	0.849/0.017/0.134
B3_82_62	0.943	0.459	0.012	0.751/0.128/0.120	0.751/0.014/0.234
B4_82_62	0.941	0.464	0.012	0.748/0.130/0.122	0.748/0.014/0.238
B2_82_62	0.920	0.405	0.012	0.763/0.124/0.113	0.763/0.015/0.221
B4_82_42	0.934	0.457	0.013	0.728/0.139/0.133	0.728/0.014/0.257
B3_81_61	0.948	0.472	0.012	0.736/0.135/0.129	0.736/0.014/0.251
B3_81_41	0.913	0.396	0.012	0.736/0.138/0.126	0.736/0.016/0.247
B6_81_61	0.950	0.485	0.012	0.721/0.142/0.137	0.721/0.013/0.266

continued on next page

continued from previous page

New York Times (NYT)					
Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.909	0.363	0.007	0.783/0.107/0.109	0.783/0.034/0.182
B4_63_43	0.918	0.387	0.007	0.781/0.109/0.110	0.781/0.033/0.186
B2_63_43	0.887	0.316	0.007	0.814/0.095/0.090	0.814/0.033/0.153
B3_82_62	0.944	0.433	0.009	0.704/0.146/0.150	0.704/0.030/0.266
B4_82_62	0.944	0.437	0.009	0.705/0.149/0.146	0.705/0.034/0.261
B2_82_62	0.923	0.394	0.009	0.721/0.145/0.134	0.721/0.040/0.240
B4_82_42	0.939	0.414	0.009	0.684/0.158/0.158	0.684/0.034/0.282
B3_81_61	0.950	0.452	0.009	0.689/0.154/0.157	0.689/0.030/0.281
B3_81_41	0.922	0.377	0.009	0.691/0.159/0.151	0.691/0.039/0.270
B6_81_61	0.954	0.468	0.009	0.685/0.155/0.161	0.685/0.028/0.287
Provident Financial Group (PFGI)					
Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.907	0.399	0.007	0.808/0.101/0.091	0.808/0.040/0.153
B4_63_43	0.919	0.419	0.008	0.802/0.105/0.092	0.802/0.039/0.159
B2_63_43	0.885	0.351	0.007	0.827/0.092/0.081	0.827/0.041/0.132
B3_82_62	0.945	0.482	0.009	0.750/0.131/0.118	0.750/0.040/0.210
B4_82_62	0.945	0.479	0.009	0.745/0.132/0.123	0.745/0.037/0.218
B2_82_62	0.922	0.420	0.009	0.759/0.127/0.115	0.759/0.043/0.198
B4_82_42	0.941	0.469	0.009	0.728/0.143/0.129	0.728/0.039/0.233
B3_81_61	0.951	0.493	0.009	0.732/0.141/0.127	0.732/0.041/0.227
B3_81_41	0.922	0.416	0.009	0.735/0.139/0.126	0.735/0.043/0.222
B6_81_61	0.955	0.504	0.009	0.728/0.146/0.126	0.728/0.033/0.240

continued on next page

continued from previous page

Payless ShowSource (PSS)					
Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.924	0.387	0.007	0.766/0.122/0.112	0.766/0.028/0.206
B4_63_43	0.930	0.421	0.007	0.753/0.127/0.120	0.753/0.026/0.221
B2_63_43	0.906	0.357	0.007	0.777/0.117/0.106	0.777/0.030/0.193
B3_82_62	0.951	0.465	0.008	0.660/0.176/0.165	0.660/0.029/0.312
B4_82_62	0.951	0.487	0.008	0.662/0.180/0.158	0.662/0.025/0.313
B2_82_62	0.934	0.421	0.008	0.687/0.158/0.155	0.687/0.029/0.285
B4_82_42	0.948	0.461	0.009	0.640/0.189/0.170	0.640/0.027/0.333
B3_81_61	0.956	0.476	0.009	0.640/0.186/0.174	0.640/0.032/0.328
B3_81_41	0.936	0.424	0.009	0.648/0.180/0.172	0.648/0.025/0.327
B6_81_61	0.960	0.496	0.009	0.643/0.184/0.173	0.643/0.028/0.329
Exxon-Mobil (XOM)					
Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.888	0.366	0.007	0.807/0.100/0.092	0.807/0.032/0.160
B4_63_43	0.901	0.377	0.007	0.805/0.100/0.095	0.805/0.027/0.168
B2_63_43	0.864	0.312	0.006	0.824/0.091/0.085	0.824/0.034/0.143
B3_82_62	0.930	0.428	0.008	0.738/0.131/0.131	0.738/0.033/0.229
B4_82_62	0.932	0.428	0.008	0.744/0.126/0.130	0.744/0.033/0.223
B2_82_62	0.907	0.388	0.008	0.738/0.134/0.128	0.738/0.040/0.223
B4_82_42	0.925	0.419	0.008	0.707/0.148/0.145	0.707/0.037/0.256
B3_81_61	0.937	0.443	0.008	0.718/0.140/0.143	0.718/0.035/0.247
B3_81_41	0.901	0.381	0.008	0.718/0.144/0.138	0.718/0.036/0.246
B6_81_61	0.945	0.465	0.008	0.716/0.144/0.141	0.716/0.032/0.253

continued on next page

continued from previous page

Yahoo (YHOO)					
Filter	ACF	P	RMAE	Hit Ratio	Buy/Sell
B3_63_43	0.910	0.380	0.019	0.779/0.114/0.107	0.779/0.043/0.179
B4_63_43	0.919	0.398	0.019	0.780/0.118/0.101	0.780/0.038/0.182
B2_63_43	0.882	0.345	0.018	0.805/0.101/0.094	0.805/0.045/0.150
B3_82_62	0.946	0.463	0.023	0.701/0.154/0.144	0.701/0.036/0.262
B4_82_62	0.944	0.471	0.023	0.711/0.150/0.139	0.711/0.032/0.256
B2_82_62	0.916	0.414	0.022	0.713/0.146/0.141	0.713/0.045/0.242
B4_82_42	0.939	0.453	0.024	0.675/0.166/0.159	0.675/0.031/0.294
B3_81_61	0.952	0.479	0.024	0.681/0.163/0.155	0.681/0.037/0.282
B3_81_41	0.920	0.406	0.024	0.679/0.164/0.158	0.679/0.040/0.281
B6_81_61	0.956	0.484	0.024	0.686/0.162/0.151	0.687/0.034/0.280

Chapter 4

Neural Network Models And Learning Algorithms

As mentioned in Section 2.6, multi-layer perceptrons (MLPs) and radial basis functions (RBFs) are two of the most popular types of neural networks that have been shown to be universal approximators. In this thesis, we choose MLPs over RBFs for the following reasons. First, an MLP normally uses less free parameters than an RBF does, since MLP is a global approximator and RBF is a local approximator. Second, it has been suggested that there exists certain relationship between these two types of networks according to Maruyama, Girosi, and Poggio [94]. Specifically, an MLP network can always simulate a RBF network arbitrarily well, and a RBF network can only approximate reasonably well an MLP network with a sigmoid function.

In the chapter, we first overview neural network models in Section 4.1 from different aspects, including their architecture, formulation, cross-validation, and learning algorithms. We then present in Section 4.2 a new architecture, namely, the recurrent FIR (RFIR) neural networks, along with generalized epochwise backpropagation through time (EWBPTT) that is the building block for most gradient-based learning algorithms. We then present in Section 4.3 our new constrained formulation that emphasizes on violation guidance. We further develop a new cross-validation method that can be incorporated in learning. We then present in Section 4.4.2 our learning algorithm called violation-guided backpropagation that is based on the Theory of *Extended Saddle Point Condition* (*ESPC*) for continuous optimization.

4.1 Overview

Time-series predictions using ANNs have traditionally been formulated as an unconstrained optimization problem:

$$\min_{w} \quad \mathcal{E}_{av}(t_0, t_1) = \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} (o_i(t) - d_i(t))^2, \tag{4.1}$$

where N_o is the number of output nodes in the network, $o_i(t)$ and $d_i(t)$ are, respectively, the actual and desired outputs of the network at time t, w is a vector of all the weights, and the training data consist of patterns observed at $t = t_0, \cdots, t_1$. The unconstrained problem is then solved by search algorithms, such as back-propagation (BP) and its variants [134, 175, 56], simulated annealing [73] and genetic algorithms [48].

In ANN training, if the size of the network is too large, then the network does not generalize well in its predictions. On the other hand, when the size of the network is small (but still large enough to model the time series under investigation), then it is hard to train it well by local search due to a lot of deep local minimum with suboptimal quality. Extensive research has been conducted in the past on learning algorithms to design ANNs with a small number of weights. However, these algorithms have limited success because little guidance is provided in an unconstrained formulation. When a search is stuck in a

suboptimal local minimum, the sum of squared errors in (4.1) does not indicate which patterns are violated and the best direction for the trajectory to move to.

To address the issue on lack of guidance, we formulate ANN learning as a constrained optimization problem in which the *objective* is to minimize the sum-of-squared errors of all training patterns, while satisfying *constraints* that the error on each training pattern must be less than a prescribed threshold [156, 157, 158]. A constrained formulation is beneficial in difficult training scenarios because violated constraints provide additional guidance during a search, leading the trajectory towards a direction that reduces overall constraint violations. Another benefit of a constrained formulation is the ability to achieve even training over the whole training set. An unevenly trained ANN that produces very small errors in some segments of the training set and large errors in some other segments will not generalize well to new patterns.

Cross validation entails the evaluation of errors between the actual and the predicted outputs on a validation set during training. There are two classes of cross-validation schemes. In *open-loop single-step cross validation* (or in short, *single-step validation*), the external input to the ANN is always the true observed data from the validation set, and the ANN is used to predict the next output in the validation set. In contrast, in *closed-loop iterative cross validation* (or in short, *iterative validation*), the external input to the ANN is the predicted output obtained in previous iteration(s). Since the input in an iterative validation is predicted and may carry errors, a larger prediction error is generally expected because error accumulates. The final network selected is one that minimizes errors in either single-step or iterative validation or a function of the two [56, 169].

Traditional cross validations usually divide the set of training patterns into two disjoint subsets: one for training and another for validation. In general, the validation set selected must be kept small, since patterns used in training and in validations are disjoint, and it is important to use as many patterns as possible for training. However, validations using only one set of patterns are not effective because the single set cannot cover multiple regimes in multi-stationary time-series. To address this issue, we develop a new cross validation approach that defines one or multiple validation sets within the training set and that includes the error from each set as an additional constraint in a constrained formulation.

The constrained formulation yields a constrained nonlinear programming problem (NLP). We show in Section 4.4.1 that existing methods are not suitable for solving this NLP when the size of the training problem is large (in terms of the number of weights used and/or the number of patterns in the training set). To this end, we develop a new training algorithm called *violation guided back-propagation* (VGBP) that exploits some special properties of neural network training in order to improve its efficiency.

4.2 Recurrent FIR Neural Networks

In this section, we present the architecture of recurrent FIR neural networks (RFIR) used in our work. We also present generalized epochwise backpropagation through time for RFIR, which can be used to calculate the gradient of the learning objective in terms of the mean squared errors.

4.2.1 Architecture

In the literature, a variety of NN architectures have been studied for modeling time series. Oftentimes, contradictory conclusions on which architecture is better were drawn by different authors working on different experiments. For example, Horne and Giles [61] concluded that "recurrent networks usually did better than TDNN except on the finite memory machine problem". But Hallas and Dorffner stated that "The results show that recurrent neural networks do not seem to be able to do so (prediction) under the given conditions" and "a simple feedforward network significantly performs best for most of the nonlinear time series" [52]. Although a consensus on the best network may never be reached, most researchers tend to agree that different applications may require different architectures and suggest that "the efficiency of the learning algorithm is more important than the network model used"[77].

The conclusions reached in previous work tell us that neither RNNs nor non-recurrent NNs are superior to the other, and that a good training algorithm is of great importance. These motivate us to design a new architecture called *recurrent FIR neural network* (RFIR in short) that combines a recurrent structure and an explicit memory structure. In addition, we need to design an accompanying efficient training algorithm.

A simple 3-layer RFIR is shown in upper panel in Figure 4.1. It's similar to a regular fully recurrent neural network except for the connection between two nodes modeled by a multiple-tap FIR filter shown in the lower panel in Figure 4.1. The FIR structures can explicitly store history information passing through the FIR filters, and the recurrent structure is able to implicitly store history information for an indefinitely long period. We believe that RFIRs are more suitable for some problems than either recurrent networks or non-recurrent feedforward networks.

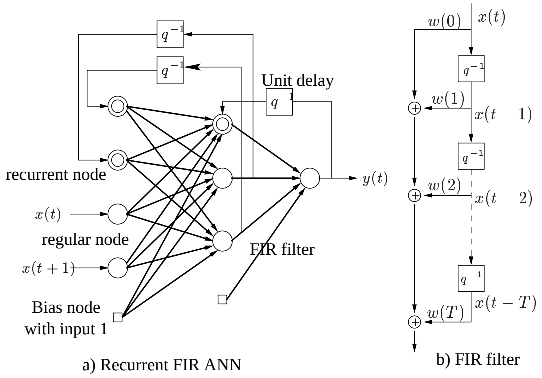


Figure 4.1: Structure of recurrent FIR neural network (RFIR) and FIR filter. Each thick line in RFIR represents a FIR filter. A single circle represents a regular node, a double circle stands for a recurrent node, and a small square is a bias node with constant input 1. A square with q^{-1} indicates a one-unit time delay.

There are different classes of recurrent neural networks, such as Elman's neural network (Elman), fully recurrent neural network (FRNN), and nonlinear autoregressive with exogenous inputs (NARX). Similarly, depending on whether there is feedback from the output layer to a previous layer and where this specific feedback goes, there are Elman-like RFIR NNs, FRNN-like RFIR NNs, and NARX-like RFIR NNs within the general structure of RFIR NNs. As a result, RFIR unifies many traditional MLPs, and a unified general back-propagation algorithm for those different MLPs can be designed for the general architecture.

The idea of combining FIR filters and a recurrent structure is not new. Aussem et al. proposed a dynamical recurrent neural network (DRNN) [11, 8] by modeling synapses as FIR filters in a fully recurrent network. There are two major differences between RFIRs and DRNNs. First, DRNNs model a *dynamic* system described by differential equations (i.e., the instantaneous change of the output of a node at time t is related to its value at time t). As a result, in a DRNN, the output of a node is fed back without any time delay. In contrast, the output of a node is fed back with at least one-unit time delay in RFIRs. This difference makes the gradient computation of DRNNs much more complicated and expensive, since it needs to find fixed equilibrium states of a series of differential equations. Second, RFIR has a much more flexible structure than DRNN.

4.2.2 Generalized epochwise backpropagation through time

For a multilayer perceptron network, a backpropagation-like algorithm is usually provided for computing the gradient of an objective function, such as the mean squared errors. The backpropagation-like algorithm for an RFIR network is called generalized epochwise backpropagation through time (EWBPTT), as the errors propagate back in time domain. Since there is no backpropagation algorithm available for RFIR networks, and the generalized EWBPTT is the basis for our new learning algorithm presented in Section 4.4.2, we present the generalized EWBPTT here in details.

For an L-layer RFIR, we include a bias node in each layer except the output layer. We index bias node as Node 1, $N(l)$ as regular nodes (see Figure 4.1) from 2 to $N(l)+1$, and recurrent nodes starting from $N(l)+2$. The number of taps for an FIR filter connecting nodes between layer l and layer $l+1$ is denoted by $T(l)$ with $T(l)+1$ coefficients. $w^l_{i,j}(m)$ denotes the weight for the m^{th} coefficient of the FIR filter that connects the i^{th} node in layer $l+1$ and the j^{th} node in layer l. The activation function for layer l is denoted by $\varphi_l(x)$. In our work, we use hyperbolic function $\tanh(\alpha x)$ with α a constant, except that

$\varphi_L(x)$ can be a linear function in the output layer. Using these activation functions, the derivative for $y = \varphi^l(x)$ is given by

$$\psi^l(y) = \begin{cases} \varphi_l'(x) = \alpha(1-y^2) & \text{if } \varphi(x) \text{ is hyperbolic,} \\ \varphi_l'(x) = 1 & \text{if } \varphi(x) \text{ is linear.} \end{cases} \tag{4.2}$$

The output value of a node indexed as i in layer l at time t is denoted by $s_i^l(t)$, and $s_i^l(t) = \psi(a_i^l(t))$ for $l \geq 2$ with $a_i^l(t)$ being the input to the i^{th} node in the l^{th} layer:

$$\begin{aligned} a_i^l(t) \quad &= w_{i,1}^{l-1} + \sum_{j=2}^{N(l-1)+1} \sum_{m=1}^{T(l-1)+1} w_{i,j}^{l-1}(m) s_j^{l-1}(t+1-m) + \\ &\sum_{j=N(l-1)+2}^{N(l-1)+N(k)+1} \sum_{m=1}^{T(k)+1} w_{i,j}^{l-1}(m) s_j^{l-1}(t-m) \end{aligned} \tag{4.3}$$

where k is the layer feeding back to layer $l-1$. Note that when there is no recurrent node in layer $l-1$, then the last term in (4.3) is dropped. The output $o_i(t)$ is same $s_i^L(t)$.

Epoch-wise back-propagation through time (EWBPTT) proposed in [175] is used for recurrent neural network training. Here, we give the generalized EWBPTT for RFIR neural networks. Define network output error for node k at time t by

$$e_k(t) = o_k(t) - d_k(t). \tag{4.4}$$

The energy function over interval $[t_0, t_1]$ is:

$$\mathcal{E}_{av}(t_0, t_1) = \frac{1}{t_1 - t_0 + 1} \sum_{t=t_0}^{t_1} \sum_{k=1}^{N(L)} e_k(t). \tag{4.5}$$

The gradient of the energy function (4.5) over $w_{i,j}^t(m)$ can be expressed as follows:

$$\frac{\partial \mathcal{E}_{av}(t_0, t_1)}{\partial w_{i,j}^l(m)} = \frac{1}{n} \sum_{t=t_0}^{t_1} \delta_i^l(t) s_j^l(t-m), \tag{4.6}$$

where $n = t_1 - t_0 + 1$. The local gradient $\delta_i^l(t)$ is computed through backpropagation as follows:

$$\delta_i^l(t) = \begin{cases} 2e_i(t)\psi^{l+1}(s_{i+1}^{l+1}(t)), & \text{if } t = t_1 \text{ and } l = L-1 \\ \left(2e_i(t) + \sum_{k=1}^{N(l+1)} \sum_{m \leq t_1, m=1}^{T(l')+1} w_{k,i+1+N(l)}^l(m) \delta_k^{l'}(t+m+1)\right) \psi^{l+1}(s_{i+1}^{l+1}(t)), & \\ & \text{if } t < t_1 \text{ and } l = L-1 \\ \left(\sum_{k=1}^{N(l+2)} \sum_{m \leq t_1, m=1}^{T(l+1)+1} w_{k,i+1}^{l+1}(m) \delta_k^{l+1}(t+m)\right) \psi^{l+1}(s_{i+1}^{l+1}(t)), & \\ & \text{if } t = t_1 \text{ and } l < L-1 \\ \left(\sum_{k=1}^{N(l+2)} \sum_{m \leq t_1, m=1}^{T(l+1)+1} w_{k,i+1}^{l+1}(m) \delta_k^{l+1}(t+m) + \right. & \\ \left. \sum_{k=1}^{N(l+1)} \sum_{m \leq t_1, m=1}^{T(l')+1} w_{k,i+1+N(l)}^l(m) \delta_k^{l'}(t+m+1)\right) \psi^{l+1}(s_{i+1}^{l+1}(t)), & \\ & \text{if } t < t_1 \text{ and } l < L-1, \end{cases} \tag{4.7}$$

where $1 \leq l \leq L-1$, and index l' is related to the recurrent node in a way that the recurrent node in layer l is fed by the regular node in layer $l'+1$. For example, in an FRNN-like RFIR, $l'+1$ equals l. When there is no recurrent node in layer l, then the term involving $\delta_k^{l'}$ is discarded.

The weights can now be updated along the negative gradient direction by means of:

$$\Delta w_{i,j}^l(m) = -\eta \frac{\partial \mathcal{E}_{av}(t_0, t_1)}{\partial w_{i,j}^l(m)}, \tag{4.8}$$

where η is a learning rate.

4.3 Constrained formulation for ANN learning

Traditional ANN training for time-series predictions is formulated as an unconstrained optimization as shown in (4.1). The unconstrained formulation can be solved by many existing optimization algorithms, such as gradient-based methods and stochastic methods.

Gradient-based methods include backpropagation and its variants, Newton's method and its variants, quadratic programming, and many others. These gradient-based methods look for local optima of the objective function (e.g. mean squared errors). Once a local optimum has been found, the search either stops or a random starting point is provided for another local search. Hence, these methods do not have a good mechanism to guide the search to find better solution once a local optimum has been reached.

Stochastic methods for neural network learning include *simulated annealing* (SA) and *genetic algorithms* (GA). These methods do provide a mechanism to find global optimal solution. But the global optimality is based on certain very restricted conditions that, in theory, requires an infinite number of samples. When time is limited, these algorithms can no longer guarantee to find optimal solutions. In practice, stochastic methods such as SA and GA get trapped in local minima when run under limited time. The difficulty of existing learning algorithms motivates us to search for a new formulation.

When the number of weights used in a network is relatively small, it is possible for good solutions to reside in very narrow deep valleys while there may be lots of local minima with poor solution quality in this very rugged search terrain. During a search, those good solutions are very easily overlooked. The search is then most likely converged to some poor local minimum and trapped there. Once the search is trapped, the objective in (4.1) itself does not provide any guidance on which direction the search should go. On

the other hand, when the number of weights in the network is too large, training may become easier because there are more local minima. But when an excessive number of parameters are used, the network tends to be overt-trained, especially when noise is present and the generalizability of the network is poor. Generalizability is more severe when the learning algorithm is so powerful that it fits the training data well enough to take noise into account. In our work, we avoid over-sized neural networks, but focus on designing efficient training algorithms.

Another disadvantage of an unconstrained formulation is uneven training. Because the search always tries to find a point with a smaller energy function value, some parts of the training data may be well trained, while other parts are not trained well enough. Figure 2.10 clearly illustrates that there are some individual patterns that are significantly under-trained when compared to most other patterns. Our experience shows that ANNs with uneven errors in their training patterns usually perform poorly, especially in multiple-regime stationary time series.

Due to the shortcomings of unconstrained formulations listed above, we develop a constrained formulation for neural network training in this thesis. Using a constrained formulation, when a search is trapped in a local minimum, those patterns that violate constraints will provide guidance for the search. Moreover, in a constrained formulation, we force individual patterns to achieve the same level of errors during a search.

4.3.1 Constrained formulation without cross-validation

During learning, we would like to identify those under-trained patterns and those that do not need further training. The natural way to achieve this goal is to set a training goal for each pattern and measure to what degree each pattern satisfies (or violates) the designated goal. For example, we can set the training goal for a learning pattern at time t as follows:

$$h_i(t) = (o_i(t) - d_i(t))^2 \leq \tau(t), \tag{4.9}$$

where $o_i(t)$ is the output of the i^{th} output node at time t, $d_i(t)$ is the corresponding desired output, and $\tau(t)$ is a small, positive predefined value. This inequality constraint requires that the squared learning error for the pattern at time t not to exceed a predefined threshold $\tau(t)$. Although $\tau(t)$ can vary with time, it is kept as a constant in most examples presented in this work.

With a constraint on each individual training pattern, we present the simplest constrained formulation on ANN training for time-series predictions as follows:

$$\begin{aligned} \min_w \quad & \mathcal{E}_{av}(t_0, t_1) = \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} \max\{(o_i(t) - d_i(t))^2 - \tau, 0\} \\ \text{s.t.} \quad & \forall\, i, t, \quad h_i(t) = (o_i(t) - d_i(t))^2 \leq \tau. \end{aligned} \tag{4.10}$$

It is easy to see that for $\tau = 0$, (4.1) and (4.10) are equivalent when training converges. Using a non-zero τ across all patterns allows training to be done to within a uniform error tolerance in case that smaller tolerances cannot be achieved. Other reasons on why a non-zero τ should be used in training is addressed in Section 4.4.2.

Unlike general constrained optimization problems, the values for the constraint $(o_i(t) - d_i(t))^2$ functions in our constrained formulation have already been computed in the objective function.

In contrast to an unconstrained formulation that provides no guidance when a search is stuck in a local minimum of the objective function, by using a constrained formulation, the search still can use the constraints to provide guidance to escape from local minima.

4.3.2 Cross-validation as additional constraints

Traditional cross-validations [144, 171] partition training data into two disjoint sets: *estimation subset* and *validation subset*. The estimation subset is used for parameter estimation or neural network learning in ANN models, whereas the validation subset is used to prevent over-fitting. When a traditional validation process is used, the set of weights that give the smallest validation error is selected as the parameters for the model. A validation error can be defined by the mean squared errors or the normalized mean squared errors in Chapter 1. For each validation set, there are two validation schemes, each giving a validation error. One scheme is called *single-step cross validation* in which the external input to the ANN is always the true observed data from the validation set. In contrast, in *iterative cross validation*, the network performs predictions multiple steps into the future. To do so, after the first iteration, the input to the ANN is the predicted output in previous iteration(s), and the network treats the predicted data the same as observed data.

The validation method with only one fixed validation set is called the *hold-out method* [144, 145]. In this validation method, training stops when the validation error stops decreasing and begin increasing. This is also called the *early stopping method of training* [107, 5]. The *hold-out method* is not desirable when training data is scarce. A variant of this validation method is the *multi-fold cross-validation* method that uses all the available training data for training [56] and that divides the available training set into K subsets with $K > 1$. Each of the K trials of training is performed over all subsets except one subset that is used for validation and a different subset is chosen as the validation subset in each trial. Although this method can fully utilize all the training patterns in training, it has the disadvantage of requiring an excessive amount of compu-

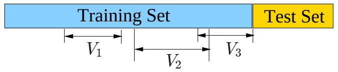


Figure 4.2: Multiple validation sets in a training set. V_1,V_2 and V_3 are three validation sets.

tation [56]. An extreme case of the multi-fold method is the leave-one-out method in which K is equal to number of training patterns. With only one pattern for validation each time, this method is only useful when training data is extremely scarce.

In traditional cross-validation methods, validation is performed after a period of training and are not performed simultaneously. As a result, cross-validation does not provide guidance on training directly, or *actively*, and it is only used as a stopping criterion for training.

Another drawback for traditional validation is the use of a single objective. When the time series studied contains multiple regimes, multiple objectives may be more desirable. For example, one may want to minimize the mean square errors across the entire training set, while at the same time one may also want to have a new objective for each regime transition region in order to make sure that regime transitions are sufficiently modeled.

To overcome the drawbacks of traditional cross-validation methods, we introduce a new cross-validation method that can utilize the whole training data for training and also use multiple cross-validation subsets during any training epoch. Overlapped validation sets are also allowed. Figure 4.2 illustrates the new cross-validation methods with three validation sets, two of which are overlapped.

In our work, we use the *normalized mean squared errors* as our validation error. The validation error for the i^{th} output node from the k^{th}, $k = 1, \cdots, v$, validation set is

defined as:

$$\varepsilon_{k,i} = \frac{1}{\sigma_i^2 N_k} \sum_{t=t_{k,0}}^{t_{k,1}} (o_i(t) - d_i(t))^2, \tag{4.11}$$

where σ_k^2 is the variance of the true time series in the period of $[t_{k,0}, t_{k,1}]$, and N_k is the number of patterns in the k^{th} set.

The cross-validation errors are incorporated as constraints in (4.10):

$$\begin{aligned}
&\min_w \mathcal{E}_{av}(t_0, t_1) = \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} \max\{(o_i(t) - d_i(t))^2 - \tau, 0\} \\
&\text{s.t.} \quad h_i(t) = (o_i(t) - d_i(t))^2 \le \tau, \quad i \in [t_0, t_1] \\
&\qquad h_{k,i}^I(w) = \varepsilon_{k,i}^I \le \tau_{k,i}^I, \quad k \in \{1, 2, \cdots, v\} \text{ and } i \in \{1, 2, \cdots, N_o\} \\
&\qquad h_{k,i}^S(w) = \varepsilon_{k,i}^S \le \tau_{k,i}^S, \quad k \in \{1, 2, \cdots, v\} \text{ and } i \in \{1, 2, \cdots, N_o\}
\end{aligned} \tag{4.12}$$

where $\varepsilon_{k,i}^I$ (resp. $\varepsilon_{k,i}^S$) is the normalized mean squared errors of the iterative (resp. single-step) validation errors on the i^{th} validation set, and $\tau_{k,i}^I$ and $\tau_{k,i}^S$ are predefined small positive constants similar to τ.

Note that the iterative validation error is not in closed form; hence, there is no closed-form gradient available for the iterative validation error. As a result, one cannot use the corresponding gradient information to adjust neural-network weights, and (4.12) cannot be solved by gradient-based optimization methods.

The constrained formulation also provides the flexibility of incorporating *a priori* knowledge about the time series. For example, if we know in advance that the time series under modeling has multiple regimes, then we can select multiple validation sets, each corresponding to one regime.

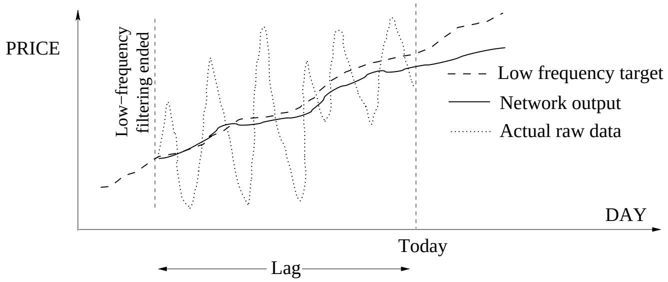


Figure 4.3: Illustration of a constraint in the lag period for financial time-series predictions. The raw data (stock price) should center around the low-pass data as well as the ANN outputs.

4.3.3 Constraints on predictions in the lag period in noisy time series

When low-pass filtering is applied in preprocessing noisy financial time series, the resulting low-pass data lags behind the original raw time series. However, in the lag period, the raw data is available for learning. To take advantage of the available raw data and to improve the prediction accuracy on preprocessed low-pass data, we include a special constraint in the lag period. Let $R(t)$ be the raw data, $S(t)$ be the low-pass data, and $\hat{S}(t)$ be the ANN output at time t (where t can be time during learning or time during prediction). Since the low-pass curve is a smoothed version of the raw data curve, the raw data in the lag period generally centers around the true low-pass curve. As the desired output for ANN should be the low-pass data, the curve of the ANN outputs (Figure 4.3) should also be centered by the raw data in the lag period. This observation motivates us to add a new constraint in (4.12) on the difference between the raw data and the ANN

outputs in the lag period [160]:

$$h^{\text{lag}} = \sum_{t=t_0-m+1}^{t_0} \hat{S}(t) - R(t) \le \tau^{\text{lag}}, \tag{4.13}$$

where m is number of lag period data to be predicted. Note that $R(t)$ is different from $d(t)$ in this problem, as $d(t)$ is in fact a low-passed version of $R(t)$. When h^{lag} is small enough, the predicted low-pass data is centered approximately around by the raw data in the same way that the true low-pass data should be.

The constrained formulation for noisy time-series prediction is stated as follows.

$$\begin{aligned} &\min_w \mathcal{E}_{av}(t_0, t_1) = \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} \max\{(o_i(t) - d_i(t))^2 - \tau, 0\} \\ &\text{s.t.} \quad h_i(t) = (o_i(t) - d_i(t))^2 \le \tau, \\ &\qquad h^I_{k,i}(w) = \varepsilon^I_{k,i} \le \tau^I_{k,i}, \\ &\qquad h^S_{k,i}(w) = \varepsilon^S_{k,i} \le \tau^S_{k,i}, \\ &\qquad h^{lag} = \sum_{t=t_0-m+1}^{t_0} \hat{S}(t) - R(t) \le \tau^{lag}. \end{aligned} \tag{4.14}$$

Eq. (4.14) is a continuous nonlinear optimization problem with highly nonlinear constraints, where some of constraints do not have closed-form gradients, some constraints are quadratic with closed-form expressions, and other constraints are in non-differentiable closed-form. As a result, the traditional backpropagation algorithm, such as epochwise backpropagation through time (EWBPTT), cannot be used for this constrained formulation. To solve this problem, we have developed the violation guided backpropagation algorithm in Section 4.4.2.

4.4 Learning algorithm for constrained formulations

The learning algorithm presented in this thesis is based on the theory of *Extended Saddle Point Condition* recently developed in our group [163, 154, 23, 164, 153, 155]. We first summarize the theory for continuous constrained optimization in this section.

4.4.1 Theory of Extended Saddle Points for Continuous Nonlinear Programming

The constrained formulation (4.12) is a continuous constrained nonlinear programming problem (NLP) with *non-differentiable functions*. The formulation, when applied to large time-series predictions, cannot be handled by existing Lagrangian methods that require the differentiability of functions.

Methods based on penalty formulations have difficulties in convergence when penalties are not chosen properly. A recently developed Theory of Extended Saddle Point Condition for continuous constrained optimization [153] can be applied even when constraint functions are not differentiable.

Since (4.12) is not differentiable, and may contain thousands of variables and tens of thousands of constraints, the traditional backpropagation algorithm (and its variants) and other gradient-based methods cannot be applied; and sample-base search algorithms, such as genetic algorithms (GA) and simulated annealing, are too expensive in terms of computation time when applied to solve (4.12). On the other hand, the newly developed Theory of Extended Saddle Point Condition [153] can be used to solve continuous constrained optimization problem with non-differentiable functions. A more general con-

tinuous nonlinear programming problem than (4.12) is defined as follows.

$$\begin{aligned} (P_c): \qquad & \min_x \quad f(x) \qquad \text{where} \quad x = (x_1, \cdots, x_w)^T \in \mathcal{R}^w \\ & \text{subject to} \quad h(x) = 0 \qquad \text{and} \qquad g(x) \le 0, \end{aligned} \tag{4.15}$$

where $\mathcal{R}^w$ is a w-dimensional continuous space, $f(x)$ is continuous and differentiable, $h(x)$ consists of m continuous functions that may be non-differentiable, and $g(x)$ consists of r continuous functions that may be non-differentiable. The goal of solving P_c is to find a constrained local minimum $x^\star$ with respect to $\mathcal{N}_c(x^\star) = \{x' : ||x' - x^\star|| \le \epsilon \text{ and } \epsilon \Rightarrow 0\}$, the continuous neighborhood of $x^\star$.

Traditional Lagrangian multiplier methods first transform (4.15) into following Lagrangian function:

$$L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x), \tag{4.16}$$

where $\lambda = (\lambda_1, \cdots, \lambda_m)^T \in \mathcal{R}^m$ and $\mu = (\mu_, \cdots, \mu_r)^T \in \mathcal{R}^r$. The original saddle point condition is stated as follows ([79, 12, 85]): $x^\star$ is a saddle point of P_c if there exist unique $\lambda^\star$ and $\mu^\star$ such that:

$$L(x^\star, \lambda, \mu) \le L(x^\star, \lambda^\star, \mu^\star) \le L(x, \lambda^\star, \mu^\star) \tag{4.17}$$

for all x that satisfies $||x - x^\star|| < \epsilon$ and all $\lambda \in \mathcal{R}^m$ and $\mu \in \mathcal{R}^r$.

The original saddle point condition (4.17) is too restricted as it requires to find the unique λ and μ. The *Theory of Extended Saddle Point Condition* for (4.15) provides a much more relaxed approach to solving (4.15). The theory can be summarized in four definitions and one theorem.

Definition 1. Point w is a CLM_c, a constrained local minimum, of P_c if $x^\star$ is feasible and $f(x^\star) \le f(x)$ for all $x \in \mathcal{N}_c(x^\star)$.

Note that in (4.12), a CLM_c is the same as a feasible point.

Definition 2. The l_1-penalty function for P_c in (4.15) is defined as follows:

$$L_c(x, \alpha, \beta) = f(x) + \alpha^T |h(x)| + \beta^T \max(0, g(x)), \tag{4.18}$$

where $|h(x)| = (|h_1(x), \cdots, |h_m(x)|)^T$ and $\max(0, g(x)) = (\max(0, g_1(x)), \cdots, \max(0, g_r(x)))^T$; and $\alpha \in \mathcal{R}^m$ and $\beta \in \mathcal{R}^r$ are penalty vectors.

Definition 3. *Subdifferential* [153], $D_x(\phi(x), \vec{p})$, of function ϕ along direction $\vec{p} \in X$ is defined as:

$$D_x(\phi(x), \vec{p}) = \lim_{\epsilon \to 0} \frac{\phi(x + \epsilon\vec{p}) - \phi(x)}{\epsilon}. \tag{4.19}$$

Definition 4. *Constraint qualification* for the *extended saddle-point condition* (ESPC) [153]. Solution $x^\star$ meets the constraint qualification if there is no direction $\vec{p} \in X$ along which the subdifferentials of continuous equality and continuous active inequality constraints are all zero. That is,

$$\nexists \vec{p} \in X \text{ such that } D_x(h_i(x^\star), \vec{p}) = 0 \text{ and } D_x(g_i(x^\star), \vec{p}) = 0 \forall i \in C_h \text{ and } j \in C_g, \tag{4.20}$$

where C_h and C_g are, respectively, the sets of indices of continuous equality and continuous active inequality constrains.

Theorem 1. *Necessary and sufficient extended saddle point condition on CLM_c* [153] of (P_c). Suppose $x^\star \in \mathcal{R}^w$ is a point in the continuous search space of P_c and satisfies the constraint qualification condition (4.20), then $x^\star$ is a CLM_c of P_c if and only if there exist finite $\alpha^\star \geq 0$ and $\beta^\star \geq 0$ such that, for any $\alpha^{\star\star} > \alpha^\star$ and $\beta^{\star\star} > \beta^\star$, the following condition is satisfied:

$$L_c(x^\star, \alpha, \beta) \leq L_c(x^\star, \alpha^{\star\star}, \beta^{\star\star}) \leq L_c(x, \alpha^{\star\star}, \beta^{\star\star}) \tag{4.21}$$

for all $x \in \mathcal{N}_c(x^\star)$ and all $\alpha \in \mathcal{R}^m$ and $\beta \in \mathcal{R}^r$. Point $(y^\star, \alpha^{\star\star}, \beta^{\star\star})$ is called an *extended saddle point*. In the following sections, we simply call an *extended saddle point* (ESP_c) a saddle point.

As the objective function in (4.12) is continuous and differentiable, and constraints are all continuous, there is no problem in applying Theorem 1 to (4.12). To apply the Theorem 1 to solve (4.12), we transform it into an an l_1-penalty function:

$$L(w, \beta) = \mathcal{E}_{av}(t_0, t_1) + \sum_{t=t_0}^{t_1} \sum_{i=1}^{N_o} (\beta_i(t)\phi(h_i(t) - \tau)) + \sum_{j=I,S} \sum_{k=1}^{v} \sum_{i=1}^{N_o} \left(\beta_{k,i}^j \phi(h_{k,i}^j - \tau_{k,i}^j)\right), \tag{4.22}$$

where $\phi(x) = \max\{0, x\}$, and $\beta_i(t)$ and $\beta_{k,i}^j$ are penalties for individual pattern constraints and other constraints, such as cross-validation constraints.

In the next section, we present an efficient algorithm to solve the l_1-penalty function (4.22) with non-differentiable terms.

4.4.2 Violation-Guided Back-Propagation

The nice feature of ESPC over the original saddle-point condition (4.17), is that, there exist relaxed $\alpha^{\star\star}$ and $\beta^{\star\star}$ instead of a unique pair as in the original saddle-point condition (4.17). Since (4.12) only has inequality constraints, we only need to find $\beta^{\star\star}$. In this section we describe an efficient algorithm to look for ESP_c for (4.22). The general framework to look for ESP_c can be found in [153] and is highlighted in the shaded box in Figure 4.4. It consists of two parts: one performing descents in w space and another performing ascents in the penalty β space. In updating β, we start from $\beta = 0$ and perform incremental updates in order to find $\beta^{\star\star}$ as discussed in [153, 155, 23]. As indicated earlier, random sampling on continuous w is too inefficient to be applied in a problem with a large number of weights/patterns. To overcome the inefficiency, we

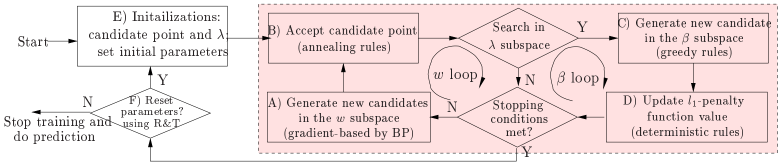


Figure 4.4: An iterative learning procedure using a constrained formulation for ANN time-series prediction. The shaded box represents the routine to look for ESP_c. R&T stands for our *relax-and-tighten* strategy.

propose in Section 4.4.2.1 to use BP to compute an approximate gradient direction for guidance. Since gradient descents may lead to infeasible local minima (i.e., points that are local minima of the objective function but do not satisfy all the constraints), we present a new annealing strategy in Section 4.4.2.2 to help escape from such points. Last, we exploit special properties in the constrained formulation for ANN time-series predictions and present in Section 4.4.3 a new relax-and-tighten strategy to successively tighten constraints as more relaxed constraints are satisfied. The strategy is depicted in the two boxes in the left in Figure 4.4. The strategy greatly accelerates the search performance in practice.

4.4.2.1 Framework to look for ESP_c

The w loop in Figure 4.4 performs descents in w space by generating candidates in Box (A) and by accepting the candidates generated using deterministic or annealing rules in Box (B). Occasionally, the β loops carries out ascents in β space by generating candidates in β space in Box (C) and by accepting them using deterministic or annealing rules in

Box (D). In this subsection we present the functions of Boxes (A), (C) and (D) and leave the discussion of Box (B) to the next subsection.

For a learning problem with a large number of weights and training patterns, it is expensive to test each point generated in Box (A). Hence, it is essential that the point generated be a likely candidate to be accepted. Since (4.22) is not differentiable, we choose to implement an approximate gradient direction by setting output error $e_i^{'}(t) \leftarrow \beta_i(t)e_i(t)$, applying (4.6) and (4.7) to compute the gradient of the sum-of-squared errors of $e_i^{'}(t)$, and generating a trial point by an approximate gradient and step size η. In this way, a training pattern with a large error (through its corresponding penalties) will contribute more in the overall gradient direction, leading to an effective suppression of constraint violations.

Step size η used in deriving a candidate point must be dynamic because the same candidate point will be generated using a fixed η and a deterministic gradient algorithm if the previous point is rejected. This strategy is essentially the same as a line search used in descent algorithms such as Newton's methods. In our algorithm, we generate η uniformly in $(0, \eta_0)$ each time a candidate point is derived, and adapt η_0 dynamically based on the acceptance ratio a of candidate points generated. The reason for the latter strategy is that a high a indicates that the current search region is promising, leading to increases in η_0 and larger step sizes. On the other hand, a low a indicates that the step size is too large for the current search terrain, leading to decreases in η_0 and smaller step sizes. After extensive experiments, we choose to change η_0 as follows:

$$\eta_0 \longleftarrow \begin{cases} \eta_0 * \left(1 + \frac{2(a-0.7)}{1-0.7}\right) & \text{if } a > 0.7 \\ \eta_0 \div \left(1 + \frac{2(0.5-a)}{0.5}\right) & \text{if } a < 0.5. \end{cases} \tag{4.23}$$

This rule tries to keep the acceptance ratio between 0.5 and 0.7. In this way the search performs descent with a high probability and avoids performing pure local descent all the time.

Box (C) in Figure 4.4 increases β as follows:

$$\beta \longleftarrow \beta + 1 \text{ if true violation } > 1.1\tau, \tag{4.24}$$

where τ is the violation tolerance defined in (4.12) and (4.22). This rule penalizes a violated constraint relative to τ. We do not generate β probabilistically because we like their effects on guidance to take place as soon as possible. For the same reason, Box (D) accepts every β update deterministically in order to allow the search to perform ascents in β space.

4.4.2.2 Probabilistic acceptances in w space

Since the gradient direction computed by BP does not consider constraints on cross validation and its step size is chosen heuristically, it is possible for a search to get stuck in infeasible local minima. In previous studies, restarts are often used to help escape from such points. However, our experimental results have shown that uncontrolled restarts may lead to a loss of valuable local information collected during a search. In this paper, we study stochastic methods to accept candidate points that lead to descents in w space.

In Box (B), we introduce an annealing strategy that decides whether to go from the current point (w, β) to a new trial point (w', β) according to the Metropolis probability:

$$A_T(\mathbf{w}', \mathbf{w})|_\beta = exp\left\{\frac{(L(\mathbf{w}) - L(\mathbf{w}'))^+}{T}\right\}, \tag{4.25}$$

where $x^+ = \min\{0, x\}$, and T is a temperature parameter introduced to control the acceptance probability. We discuss schemes for setting T in Section 4.4.5 after presenting the violation-guided backpropagation.

4.4.3 Relax-and-tighten (R&T) strategy

In this section, we discuss methods for setting violation tolerance τ in (4.12) and (4.22). It is undesirable to set τ to 0 initially in a search because we do not know whether such a violation tolerance can be achieved by the search. Moreover, setting τ to 0 will result in considerably large violations in each pattern, leading to large β's, a rugged search space, and a more difficult search. On the other hand, if we set a loose $\tau > 0$ initially, then most constraints can be satisfied easily, and the algorithm can focus on the few patterns with large constraint violations and increase their corresponding β's.

Another observation is that the progress of a search differs substantially for different fixed τ's. These differences are illustrated in Figure 4.5 that shows the average maximum violations for different N (number of evaluations) over five independent runs. When N is small, there is little difference in the maximum violations. The violation-guided backpropagation algorithm presented in Section 4.4.4 is used in all the experiments, and the temperature parameter is fixed as described in Section 4.4.5. As N is increased, runs with larger τ's have faster decreases in the maximum violation than those with smaller τ's. Eventually, all the curves level off when either all the constraints are almost satisfied using the specified τ or further improvement is impossible using the given network topology. The figure also shows a steeper rate of decrease of maximum violations with larger τ's.

Our R&T strategy exploits the different convergence behavior due to the different τ's by dynamically adjusting τ during a search in order to achieve fast convergence in

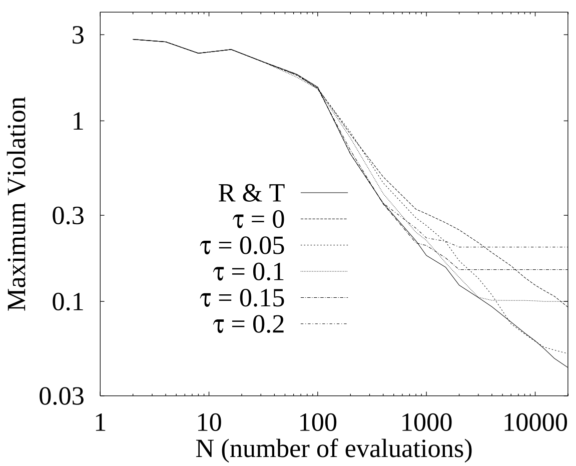


Figure 4.5: Decreases of the maximum violation over all training patterns for the MG17 time series when different initial violation tolerance τ's (broken lines) are used and when our relax-and-tighten (R&T) strategy (solid line) is used.

the search. This is done by choosing a loose τ initially and by tightening $\tau \leftarrow \beta\tau$ when the true maximum violation of constraints satisfies $\max_i\{h_i(t)\} \leq (1+\gamma)\tau$, where $0 < \gamma < \beta < 1$. In this way, the search will try to use the largest possible τ at any time and will switch to a smaller τ as the convergence behavior using the original τ levels off.

Figure 4.5 illustrates the behavior of our R&T algorithm. Initially, we set $\tau = 0.2$, leading to the steepest convergence behavior. When the convergence behavior levels off, we switch to $\tau = 0.15$ by tightening the constraints, again leading to the steepest convergence behavior for the range of N used. By repeatedly tightening the constraints,

the convergence behavior of R&T is effectively the envelope of the best convergence behavior at any time.

The choice of the initial τ is not critical to convergence as long as it is large enough. This is true because the larger the τ is, the steeper will be the curve and the shorter the amount of time before it levels off. In this case, R&T will tighten τ quickly. In our implementation, we set our initial $\tau = 0.8 \max_i\{h_i(t)\}$ over all the constraints, $\beta = 0.95$, and $\gamma = 0.1$. Around those values set, convergence is not sensitive to the different β's and γ's.

The R&T strategy works in a constrained formulation of ANN learning because all the constraints are defined in the same range (limited by the activation function) and all the constraints have similar magnitudes. In a general constrained NLP in which constraint violations may vary in large ranges, it will be necessary but difficult to define different amount of relaxations for different constraints. As a result, R&T does not work well in solving general constrained NLPs. Research on applying R&T to solving general constrained NLPs is under investigation in our group.

4.4.4 Violation-guided backpropagation algorithm

Figure 4.6 shows the violation-guided backpropagation algorithm (VGBP) with the R&T strategy included. A detailed explanation for this algorithm is presented below.

Line 2 initializes the parameters used in VGBP. Weights w can be generated randomly or assigned beforehand. β's are set to zeros by default. η_0 is set to be 1 and adjusted dynamically. The default setting for T is discussed in Section 4.4.5. N_S is the ratio of the frequencies of updating w and β. The default value for N_S is 20. The initial τ is set

```
1  procedure VGBP
2      set initial w = (w, β), η_0, T, N_S, τ;
3      run one pass of the feed-forward process;
4      while stopping condition is not satisfied do
5          for k ← 1 to N_S do
6              for t ← t_0 to t_1
7                  for i ← 1 to N_o
8                      e_i(t) ← β_i(t)e_i(t)
9                  end_for
10             end_for
11             run BP to obtain δw;
12             accept w' ← w + δw using (4.25);
13             set τ ← 0.95τ if max_i{h_i} ≤ 0.1τ
14         end_for
15         adjust β according to constraint violations;
16         adjust η_0 according to acceptance ratio a
17     end_while
18 end_procedure
```

Figure 4.6: Procedure of violation-guided back-propagation algorithm. Refer to Section 4.4.4 for a detailed explanation for each line. $e_i(t)$ is error between the network output and the desired output defined in (4.4).

to be 0.8 times of the maximal violation of constraints on individual patterns. Other τ's (e.g. τ^I) can be set in a similar way. This step relaxes the constraints first.

Line 3 runs the feedforward process once in order to gather information on the objective and the constraints before testing the stopping criterion in Line 4. If the stopping criterion is met, then training is done; otherwise, proceed to Line 5.

Lines 6 to 10 penalize the network-output error $e_i(t)$ by penalty $\beta_i(t)$ according to the constraint violation of the i^{th} pattern at time t. Lines 11 and 12 generate a new sample in the weight subspace by using backpropagation and then determine whether to accept the generated sample using the Metropolis probability.

Line 13 tightens the constraints if all the constraints are close to be satisfied. This is the stage for tightening the constraints in the relax-and-tighten strategy.

After repeating the process in Lines 6 to 13 N_S times, the procedure updates penalty β's in Line 15 according to constraint violations. It also updates step size η_0 in Line 16 according to the sample acceptance ratio.

The process is repeated until the stopping criterion is met in Line 4.

4.4.5 Parameters in VGBP

In this section, we summarize the values of parameters used in VGBP.

Figure 4.7 plots the progress of the mean squared errors (MSE) defined in (4.1) of training an ANN in order to predict the Mackey-Glass-17 time series (in short MG17) by using two fixed temperatures: $T = 1$ and $T = \infty$, respectively. (The MG17 is used as a running example throughout this section unless otherwise specified.)

When $T = \infty$ is combined with restarts, the algorithm accepts every trial point generated in the same way as traditional BP. Figure 4.7 illustrates this behavior that

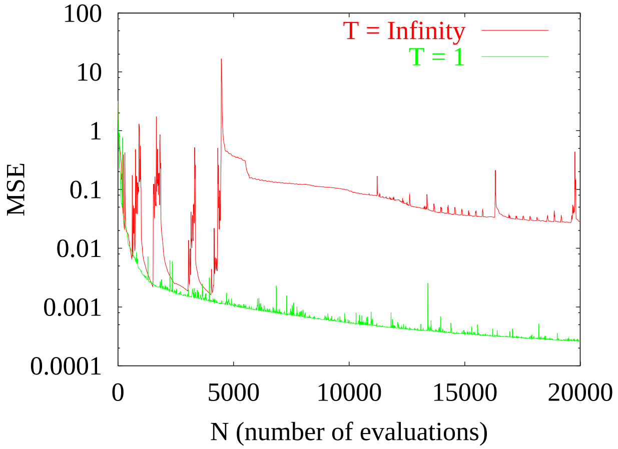


Figure 4.7: Progress of MSE defined in (4.1) for $T = 1$ and $T = \infty$ during the learning of an ANN to predict the MG17 time series.

shows the search exploring a local region in the first 4000 evaluations, getting stuck in an infeasible local minimum, and restarting to a new point without keeping any history information.

On the other hand, using $T = 1$ allows the search to accept trial points according to (4.25) and rejects poor points with high probability. Consequently, the algorithm keeps implicitly the history information of points searched in the past and progresses smoothly without escaping into poor regions blindly.

In contrast to conventional annealing schedules that start a search at high temperatures and that decrease the temperature to zero as time runs out, we use a fixed temperature throughout the search. Our strategy was chosen in order to de-emphasize the effect

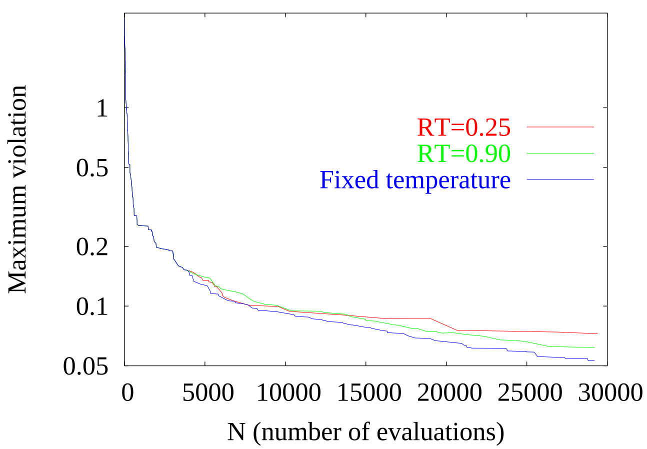


Figure 4.8: Decreases of the maximum violation over all constraints for the MG(17) time series between using a fixed temperature ($T = 5$) and using a schedule of decreasing temperatures ($T_0 = 5$, $T_{i+1} = 0.25T_i$ and $T_{i+1} = 0.95T_i$ every 4000 evaluations, $T_\infty = 0.0003$, and η_0 was adjusted every 50 evaluations).

of local searches at low temperatures. At low temperatures, annealing rejects almost every trial point with worse quality. Since backpropagation performs local descents all the time, if it is performed at low temperatures, then the search will over-emphasize on local searches and may miss the opportunity to escape from a local minimum. Using a fixed temperature (but not a fixed low temperature) allows a search to always have an opportunity to explore better regions.

Figure 4.8 compares the progress in maximum constraint violations when a fixed temperature is used as compared to the progress when a dynamically decreasing temperature schedule is used. At high temperatures, the maximum violation drops quickly, and there is little difference in the performance in both two cases. This is because both a global search and a local search are enabled at those temperatures. The difference comes at low temperatures in a dynamic temperature schedule. In comparing the two cases in which a dynamic temperature schedule is used, there is almost no progress since iteration 21000 for the case when $T_{i+1} = 0.25T_i$, but there is still some progress for the case when $T_{i+1} = 0.95T_i$. Clearly, the one using a fixed temperature outperforms the other cases. This is due to the fact that at low temperatures, a local search is over emphasized, and the search has little chance to explore other regions.

In VGBP, we set

$$T = \alpha N_p R, \tag{4.26}$$

where N_p is the number of training patterns and is known when training begins, R is the range in which ANN outputs are normalized and is set to a default value of one, and α is a constant. T should be proportional to N_p because (4.22) is proportional to N_p when all the patterns have approximately the same level of violations. Likewise, T should be proportional to R because R affects (4.22) in a similar manner.

Figure 4.9 shows the average $nMSE$'s over 10 runs of VGBP under different α's for MG17, MG30, and sunspots. Since VGBP is robust over a wide range of $\alpha \in [10^{-6}, 10^{-2}]$, we set the default α to be 10^{-3} in our implementation.

We further set η_0 in (4.23) to be 1.0. Since η_0 is adjusted dynamically, its initialization has no significant impact on performance. The setting of τ, β and γ in R&T has been discussed in Section 4.4.3.

In short, all the parameters in VGBP are set either by default or automatically, with no tuning required by users during training.

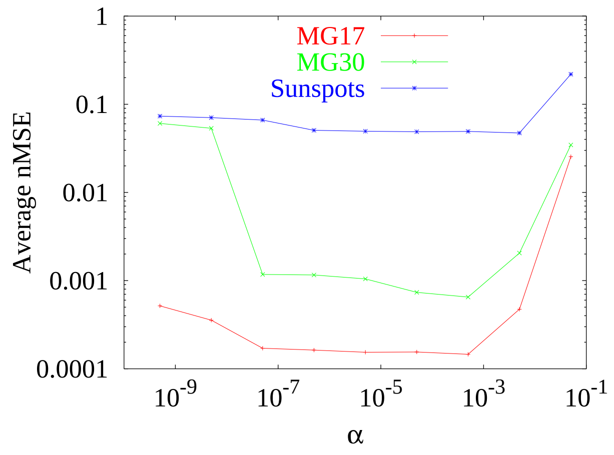


Figure 4.9: Robustness of VGBP with respect to α in predicting the MG-17, MG-30, and sunspots time series.

4.5 Summary

In this chapter, we have presented a recurrent FIR neural network that unifies a recurrent neural network and an FIR neural network. We have also introduced a constrained formulation for artificial neural network models. In the constrained formulation, we have shown schemes for setting constraints on individual patterns, cross-validation sets, and other user-defined constraints. By using constraints, we can optimize multiple objective functions simultaneously. We have further developed a new learning algorithm called violation-guided backpropagation based on the theory of extended saddle points. Finally, we have studied the selection of parameters and a relax-and-tighten strategy in order to make the search more efficient.

Chapter 5

Experimental Results On Near Noiseless Chaotic Time Series

In this chapter, both real-world time series, such as laser intensity and yearly sunspots, and artificially generated chaotic time series, including Mackey-Glass-17 (MG17), Mackey-Glass-30 (MG30), Henon map, Lorenz attractor, and Ikeda attractor, are studied using various architectures trained by VGBP. The performance is measured by both the nMSE of predictions and the number of ANN weights.

5.1 Real world time-series

The sunspots time series and the laser intensity time series are two sets of widely studied standard real world time-series benchmark in the literature. In order to give a good comparison with previous work, we study these two sets of benchmarks in our work.

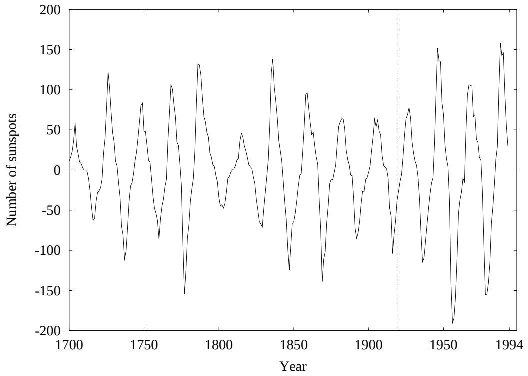


Figure 5.1: *Sunspots* time series. Dashed vertical lines divide the training and testing sets.

5.1.1 Sunspots time series

Sunspots contains yearly averaged sunspots numbers from 1700-1994. In order to compare our prediction results with previous work, in the experiments, we use data from 1700-1920 for training and evaluate single-step predictions on four durations (1921-1955, 1956-1979, 1980-1994, and 1921-1994). The architecture used in our work is an Elman recurrent neural network with only 2 hidden units. We use only two hidden units for sunspots time series, which may be the smallest number of hidden nodes one can use in this problem. For the same problem, Wan used 113 weights [167] in a feedforward FIR network, and Aussem used 30 weights [8] in a dynamical recurrent network. In those studies, the performance before 1955 is rather accurate, but the performance degrade significantly after 1955.

To compare the performance of constrained formulations and our proposed cross-validation methods with traditional formulation and traditional cross-validation, we trained an ANN for an application using a fixed amount of time (roughly 23-25 seconds on a Pentium III 450MHz machine running Solaris 7) by each of the following five methods.

1. SA: Unconstrained formulation trained by SA (using the SIMANN [47] package with its sample generation replaced by EWBPTT) without cross-validation,
2. SA&V1: Unconstrained formulation trained by SIMANN [47] with traditional cross-validation, using *sunspot* patterns from 1901 to 1920 as the validation set,
3. VGBP(NV): Constrained formulation trained by VGBP but without cross-validation,
4. VGBP(V1): Constrained formulation trained by VGBP with traditional cross-validation, using *sunspot* patterns from 1901 to 1920 as the validation set, and
5. VGBP(V2): Constrained formulation trained by VGBP with proposed cross-validation, using *sunspot* patterns from 1881 to 1920 as the validation set.

For each method, we run the program multiple times in order to reach a 95% confidence of the mean behavior within 10% of the values presented; i.e., the program is repeated (with different initial weights) until the true mean nMSE a has 95% confidence to fall into the interval of $[\bar{a} - \bar{a}/10, \bar{a} + \bar{a}/10]$ where $\bar{a}$ is the measured average nMSE of the prediction. The less number of runs needed to reach such a confidence interval, the more consistence the method exhibits.

Table 5.1 compares the average test performance of the five methods. The much better results of VGBP(NV) over SA on all four test durations (without cross-validation)

Table 5.1: Average test performance by different formulations and cross-validation methods on *sunspot*. (Each entry in the second through the fifth columns represents a mean value with 95% confidence and $\pm 10\%$ of the value indicated. The last column shows the number of runs needed to achieve the required level of confidence. Bold numbers indicate the best results. Each run took approximately 24 to 26 seconds on a 450-MHz Pentium-III computer under Solaris 7. The ANN used has one input, two hidden units, one output, and 11 weights.)

Method	1921-1955	1956-1979	1980-1994	1921-1994	Runs
BP	0.043286	0.085110	0.045041	0.058779	27
BP&V1	0.057859	0.106697	0.058096	0.075161	37
SA	0.052306	0.113958	0.055074	0.076735	28
SA&V1	0.081943	0.138510	0.086798	0.102199	76
VGBP(NV)	0.035385	0.061361	0.039559	0.045554	10
VGBP(V1)	0.042079	0.086607	0.051244	0.060784	18
VGBP(V2)	**0.034288**	**0.053549**	**0.034236**	**0.040634**	4

demonstrate the merits of using constraints in the problem formulation. The table also shows that traditional cross validation does not work well in both unconstrained and constrained formulations. This may be attributed to the facts that the test set (20 patterns) takes up patterns in the already small training set (220 patterns), and that the choice of the test set may not be the best. The last column of Table 5.1 illustrates the rather erratic training behavior of SA and SA&V1 in which each method needs a large number of runs in order to reach the specified confidence of the mean behavior within 10% of the values presented. Finally, the table shows that VGBP(V2) has the best test performance among the five methods over the four test durations, and that it requires the least number of runs to reach the given confidence interval. The stability of VGBP(V2) has also been observed in 100 runs of the algorithm. In the following, VGBP stands for VGBP(V2) which is the VGBP algorithm based on a constrained formulation using the proposed cross-validation method.

Table 5.2 compares the prediction results of the VGBP algorithms and some previous results. The table shows that VGBP achieves much better predictions of all prediction periods of all four prediction durations. The prediction is especially more accurate than predictions from previous work in the last stage (duration 1980-1994). This indicates that the ANN we have obtained captures the underlying chaotic behavior more accurately than models found in previous work. Several factors may contribute to this achievement. First, we have used less weights than previous designs (especial for nonlinear models). This may effectively prevent over-fitting and increase generalizability. Second, due to our proposed cross-validation method, we were able to train the network using the whole training data, without using the scarce training data only for cross-validations. Last, the network is well trained by VGBP as shown in Table 5.2.

Table 5.2: Single-step test performance in nMSE on *sunspot* for AR(12) [167], WNet [174], SSNet [117], DRNN [8], COMM [167], ScaleNet [44], and VGBP. Boxed numbers indicate the best results. N/A stands for data not available. n represents the number of weights used. The second column shows the number of weights/free variables in each method.)

Method	n	Training	Single-Step Testing			
		1700-1920	1921-55	1956-79	1980-94	1921-94
AR(12)	12	0.128	0.126	0.36	0.306	0.238
WNet	113	0.082	0.086	0.35	0.313	0.219
SSNet	N/A	N/A	0.077	N/A	N/A	N/A
DRNN	30	0.105	0.091	0.273	N/A	N/A
COMM	N/A	0.079	0.065	0.24	0.188	0.148
ScaleNet	N/A	0.086	0.057	0.13	N/A	N/A
VGBP	11	$\boxed{0.0559}$	$\boxed{0.0337}$	$\boxed{0.0524}$	$\boxed{0.0332}$	$\boxed{0.0397}$

5.1.2 Laser intensity time series

The *Laser* time-series, introduced in the *Sante Fe Institute Time Series Prediction and Analysis Competition* during the fall of 1991 [173], is a set of chaotic intensity pulsation of an NH_3 laser in a clean experimental environment. The *laser* time series is plotted in Figure 2.11. The first 1000 data items are to be used as training data, and the next 100 are to be predicted in the competition.

There is little noise in the measurement. But as clearly seen from Figure 2.11, it contains two regimes: the regular regime and some corruptions alternate. This *a priori* knowledge can be embedded in training by choosing two validation sets: one cross-validation set, consisting of data 590 to data 639, covers the corruption point, and the other cross-validation set, containing the last 100 data in the training set, ensures that the short-term prediction is reliable. The architecture we have used in this experiment is an Elman recurrent neural network with a 20-node hidden layer. There are 461 weights in this ANN.

In the *Sante Fe Competition*, Wan's FIR-NN [169] took the first place. In the competition, Wan reserved the last 100 data (data 901-1000) for cross-validation. Wan also pointed out that the last 100 data does not contain intensity corruption, and he ran iterative predictions starting near point 550 through point 600 in order to determine how well the network predicted the known corruptions. Hence, Wan used multiple cross-validation sets as we did in our study. The difference lies in how to handle multiple cross-validation errors. In our case, we treat each validation error as a constraint. While in Wan's approach, the "training was halted when an acceptable iterated prediction was achieved" [169]. We believe that our method of handling multiple cross-validation sets is

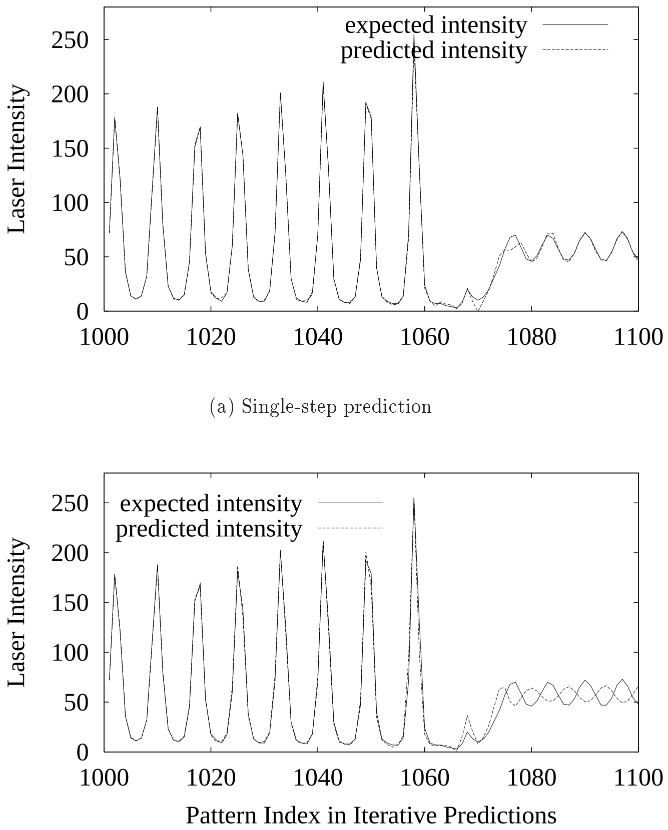
Table 5.3: Single-step and iterative test performance in nMSE on *laser*. (The test set consists of patterns from 1001 to 1100. As a comparison, we also computed nMSE on single-step as well as iterative predictions for patterns 1001-1050 in the test set. Boxed numbers indicate the best results; N/A stands for data not available.)

Method	Number of weights	Training	Single-step predictions		Iterative predictions	
		100-1000	1001-1050	1001-1100	1001-1050	1001-1100
FIR network [169]	1105	0.00044	0.00061	0.023	0.0032	0.0434
ScaleNet [44]	N/A	0.00074	0.00437	0.0035	N/A	N/A
VGBP (Run 1)	461	0.00036	$\boxed{0.00043}$	$\boxed{0.0034}$	0.0054	$\boxed{0.0194}$
VGBP (Run 2)	461	0.00107	$\boxed{0.00030}$	$\boxed{0.00276}$	$\boxed{0.0030}$	$\boxed{0.0294}$

more systematic. After Wan's work, Geva used multiple feedforward neural networks to train and predict the laser time series, but he was not as successful as Wan was.

Table 5.3 lists the prediction results of the VGBP algorithm based on a constrained formulation and our proposed cross-validation method. Wan's result along with another recent results are also listed in the table for comparison. Table 5.3 shows that VGBP improves over the previous algorithms in terms of prediction quality. In this time series, Wan used 1105 weights (on 1000 available training data) in order to capture the multiple regimes (mainly the corruption position). In our network, only 461 weights were used, yet we achieved better results. By embedding the turning points into cross-validation sets, we were able to take care of multiple regimes more effectively.

Figure 5.2a shows the single-step prediction achieved by VGBP, and Figure 5.2b plots the iterative prediction results. In single-step predictions, the prediction is remarkably accurate until it comes to the corruption point; at that point our ANN drifts away from



(a) Single-step prediction

(b) Iterative prediction

Figure 5.2: Single-step (a) and iterative prediction results of an ANN trained by VGBP for the *laser* time-series. The solid line plots the actual data and the dashed line represents the prediction data.

the actual data a little bit; after the corruption point, our ANN regains high accuracy quickly. For iterative predictions, before the corruption point, the predictions are very accurate before the network successfully predicts the corruption point. The predictions then recover from corruption but with some phase shift. Note that it is inevitable that iterative predictions diverge from the actual time series when the prediction horizon becomes long.

5.2 Artificial chaotic time series

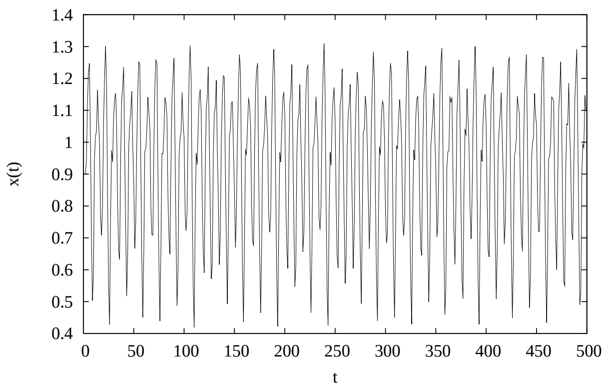
In this subsection, we compare the performance of our proposed neural-network model trained by VGBP and those of previous work, especially Wan's work [169] and Aussem's work [8]. Four sets of well-known chaotic time series are studied: Machey-Glass, Henon map, Lorenz attractor, and Ikeda attractor.

5.2.1 Mackey-Glass

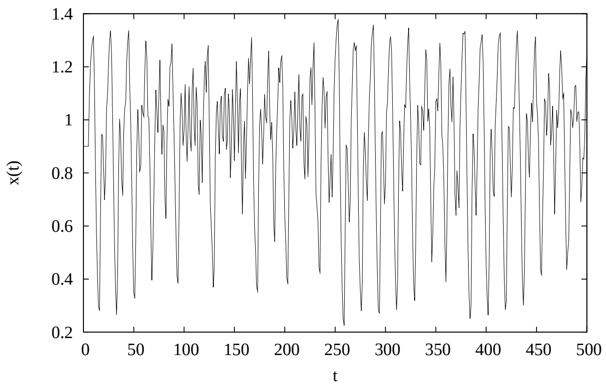
The Mackey-Glass chaotic time series comes from the delay-differential equation:

$$\frac{dx}{dt} = -0.1x(t) + \frac{0.2x(t-\Delta)}{1+x(t-\Delta)^{10}}. \tag{5.1}$$

A fourth order Runge-Kutta technique was used to simulate the chaotic series with the delay Δ set to 17 and 30, respectively. The initial conditions for the differential equation in (5.1) used in the simulation was $x(t) = 0.9$ for $0 \leq t \leq \Delta$, and the sampling rate was set to $\tau = 6$. The parameters and initial conditions used here are the same as those used in [169, 8]. The resulting two series are called MG17 and MG30 and are plotted in Figure 5.3.



(a) Mackey-Glass (17) with $\Delta = 17$



(b) Mackey-Glass (30) with $\Delta = 30$

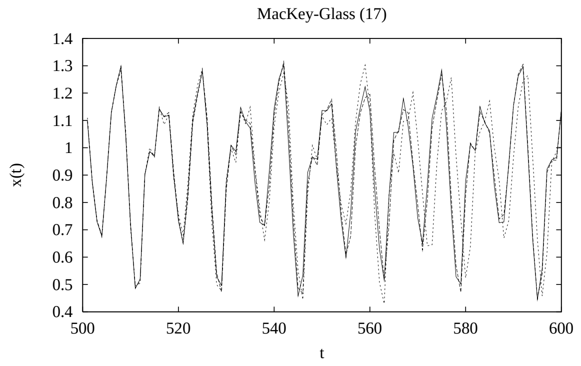
Figure 5.3: MacKey-Glass time series with (a) $\Delta = 17$ and (b) $\Delta = 30$.

To train the Mackey-Glass time series, Wan used 196 weights [169], and Aussem used 197 weights [8]. The training set consists of the first 500 data; iterative predictions are performed starting from data 501 to 600; and single-step predictions are carried out from data 501 to 2000.

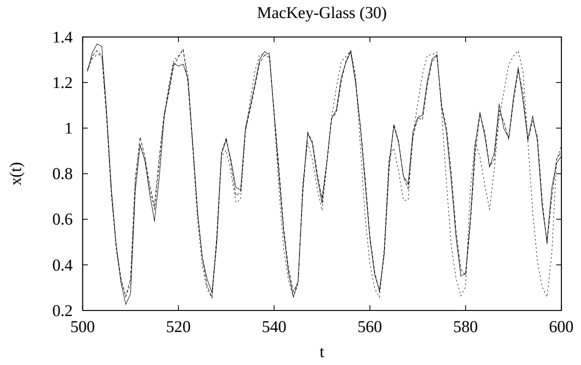
In our work, we use RFIR neural networks with a much smaller number of weights to train the two sets of Mackey-Glass time series. A small network will lead to better generalizability. In both time series, the same RFIR neural network architecture was used. The node configuration is 1 : 6 : 1 (1 input node, 6 hidden nodes, and 1 output node), and filter orders are 4 : 1 : 0 (4-tap filter connecting to the input node, 1-tap filters for the connections between the input node and the hidden nodes, 1-tap filters in the connections between the hidden nodes and the output node). Indeed, our VGBP achieves much more accurate iterative predictions as compared to those of [167] and [8]. The iterative prediction nMSE for the duration 501-600 can be as small as 0.018 for MG17 and 0.0064 for MG30. The iterative prediction results are plotted in Figure 5.4 for both MG17 and MG30. The figures show that our iterative predictions are very accurate for as long as 100 steps. Aussem pointed out in [8] that "The less information presented at the input, the more perturbed is the input/output mapping" and he used this reasoning to explain why his DRNN (with 5 past inputs) had better short-term prediction and worse long-term prediction than Wan's FIR NN (with 9 past inputs). Aussem might be right if none of the two networks (Wan's FIRNN and Aussem's DRNN) was trained much better than the other one. We used only 5 past data (due to the use of a 4-tap FIR filter connected to the input node) in our network but still can achieve much better long term prediction than Wan's network with 9 inputs. The success can be attributed to the facts that our network was well trained with our training algorithm and that a smaller network was used. These lead to better long term generalizability.

Table 5.4 compares the single-step prediction performance of our VGBP for both sets of Mackey-Glass time series with previous work. Wan's FIR NN and Aussem's DRNN performed reasonably well on these two time series. Our networks were able to achieve an nMSE near 200 times smaller than either of their results for MG17 and near 40 times better for MG30 even with less than two third of the weights they used.

For iterative predictions, it may be hard in general for chaotic time series, since they are unpredictable by nature. For this reason, DRNN [8] did not emphasize on its iterative-prediction performance [9]. However, our results show that it is possible to predict at least the first 100 steps for the *Mackey-Glass* time series (starting from data 501 up to data 600). Figure 5.4 plots the iterative predictions of the ANN found by VGBP and that by Wan's algorithm for MG17 and MG30. The figure shows that our iterative predictions are accurate for as many as 100 steps, while Wan's iterative prediction after 70 steps diverged from the actual data. In Aussem's work, significant divergence can be observed ([9]) in the iterative prediction from roughly the 40^{th} step. Numerically, VGBP was able to achieve iterative-prediction $nMSE$'s of 0.018 for MG17 and 0.0064 for MG30, respectively. In contrast, applying Wan's training algorithm on FIR-NN [167] leads to iterative-prediction $nMSE$'s of 0.3832 for MG17 and 0.1487 for MG30. Unfortunately the software for Aussem's work is not available for us to repeat his experiments (per our personal communication with Aussem [9]).



(a) Mackey-Glass-17 ($\Delta = 17$)



(b) Mackey-Glass-30 ($\Delta = 30$)

Figure 5.4: A comparison of 100-step iterative predictions on two sets of *Mackey-Glass* time series. Solid lines represent the actual data; long dashed lines indicate the predicted data using VGBP; and short dashed lines are prediction results by running Wan's FIR-NN training algorithm in [169]. Our predictions (long dashed lines) are almost indistinguishable from the actual data for the most part of the two graphs.

Table 5.4: Normalized mean square errors for single-step prediction errors for the MacKey-Glass time series (1500 steps) using different prediction models.

Model	$nMSE$ $\Delta = 17$	$nMSE$ $\Delta = 30$	network nodes	filter taps	Number of weights
Linear	0.320	0.375	–	–	–
Poly	0.020	0.061	–	–	–
Rational	0.102	0.0699	–	–	–
Loc(1)	0.0518	0.0837	–	–	–
Loc(2)	0.0228	0.432	–	–	–
RBF	0.0194	0.0408	–	–	–
MLP	0.0183	0.0498		–	–
FIR Network	0.00985	0.0279	1:15:1	8:2	196
DRNN	0.00947	0.0144	1:7:1	4:2:0	197
VGBP	**0.000057**	**0.000374**	1:6:1	4:1	121

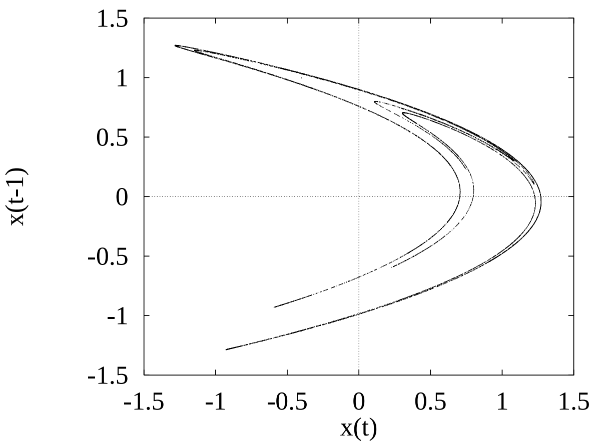


Figure 5.5: Phase plot of Henon map: $x(t)$ versus $x(t-1)$.

5.2.2 Henon map

In this example, consider the bi-variate *Hénon* equations:

$$\begin{aligned} x(t+1) &= 1.0 - 1.4x^2(t) + y(t) \\ y(t+1) &= 0.3x(t). \end{aligned} \tag{5.2}$$

The phase plot of $x(t)$ versus $x(t-1)$ in Figure 5.5 reveals a remarkable structure called the *strange attractor*.

We trained this time series by several architectures. One architecture used is a 3-layer Elman recurrent neural network with 1 input, 1 output and 13 hidden nodes. There are 209 weights in this network. We also trained the series by a 185-weight RFIR network with nodes configuration of $1:8:1$ (1 input node, 8 hidden nodes, and 1 output node) and filter taps of $3:1:0$ (a 3-tap filter connecting input signals with the input node, 1-

Table 5.5: Normalized single-step prediction errors for the Hénon map time series by using different prediction models. The number of weights includes bias weights. Both AR and DRNN results are from [8], and FIR Network result is taken from [169].

Model	Autoregressive	FIR Network	DRNN	VGBP
nMSE	0.874	0.0017	0.0012	**0.000034**
Number of weights	–	385	261	209
Nodes	–	1:12:12:1	1:10:1	1:13:1
Taps	–	3:1:1	3:1:0	1:1

tap filters connecting the input node to hidden nodes, and 0-tap filters in the connections from hidden layer to the output node). Again, the number of weights in both networks is considerably smaller than the ones used by Wan [169] and Aussem [8] (385 and 261 weights, respectively).

Table 5.5 shows that, with less weights, we can achieve about 35 times better prediction performance in terms of nMSE with the Elman network and about 10 times better performance with the smaller RFIR network. Note that, with more weights, we can achieve even better prediction performance. Here, we want to demonstrate that, even with less weights, we can still achieve much better prediction quality using our VGBP algorithm based on a constrained formulation.

As to iterative prediction, we can achieve an $nMSE$ of 0.1369 for the first 20 steps, whereas Wan's algorithm achieves an $nMSE$ of 0.6252. No iterative prediction error is available for Aussem's work on Henon map.

5.2.3 Lorenz attractor

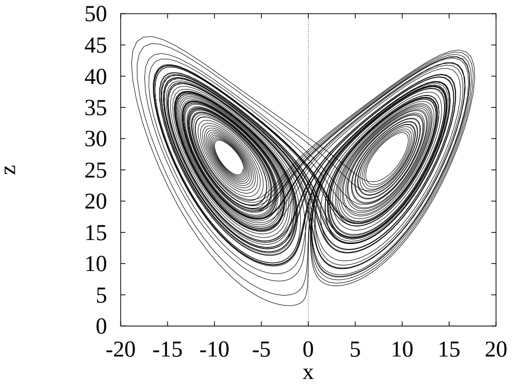
A Lorenz system is described by the solution of the following three differential equations:

$$\begin{aligned} \tfrac{dx}{dt} &= -\sigma x + \sigma y \\ \tfrac{dy}{dt} &= -xz + rx - y \\ \tfrac{dz}{dt} &= xy + bz. \end{aligned} \tag{5.3}$$

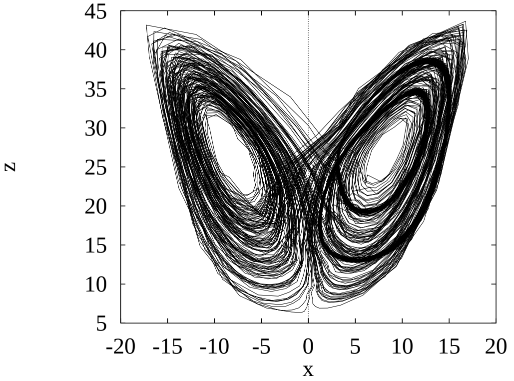
Parameters $\sigma = 10, r = 28$, and $b = \frac{8}{3}$ are used to generate the x and z time series. Although there are two sets of data streams, we only used one data stream as input and both data streams as outputs, just as were done in [167, 8]. In this way, the network needs to learn the relationship between stream x and stream z, in addition to predicting the future data for both streams. This is considered to be harder than when both streams are used as inputs [169].

Like in the case of training the Henon map time series, we used two networks. One network with a slightly smaller number of weights than the ones used in previous work (Table 5.6) is an Elman RNN with 1 input, 2 outputs and 21 hidden nodes. Wan's FIR network uses over 1000 weights, which is about twice as many weights as we have used. Aussem's DRNN uses slightly more weights than we do.

For the single-step prediction result, Table 5.6 shows that the ANN found by VGBP can achieve over 100 times smaller nMSE's for both sets of data streams, while using less weights. Figure 5.6b shows the phase plot of the iterative predictions. We can see that our iterative predictions were able to capture the relationship between the two data streams successfully.



(a) Actual series



(b) Predicted series

Figure 5.6: Phase plot of Lorenz time-series: x versus z. (a) Actual time series, and (b) iteratively predicted time series. The parameters used to generate (b) are listed in Table 5.6.

Table 5.6: Normalized single-step prediction errors for Lorenz equations by using different prediction models. The results on autoregressive and DRNN are taken from [8], and the results on FIR Netowork are taken from [169].

Model		Autoregressive	FIR Network	DRNN	VGBP
nMSE	x	0.036	0.0070	0.0055	**0.000033**
	z	0.090	0.0095	0.0078	**0.000039**
Number of weights		–	1070	542	**527**
Nodes		–	1:12:12:2	1:10:2	1:21:2
Taps		–	2:5:5	30:1:0	0:0:0

5.2.4 Ikeda attractor

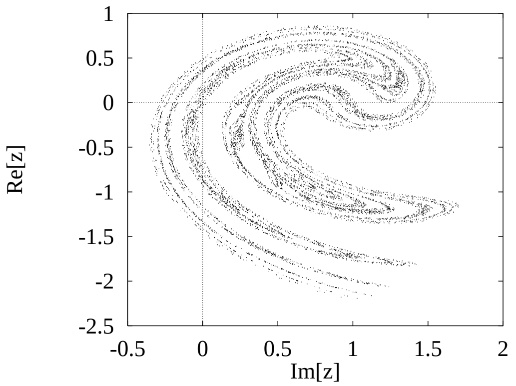
Ikeda attractor is described by the following complex plane equation:

$$z(t+1) = a + bz(t)exp\{i[\phi - c/(1+|z(t)|^2)]\}, \tag{5.4}$$

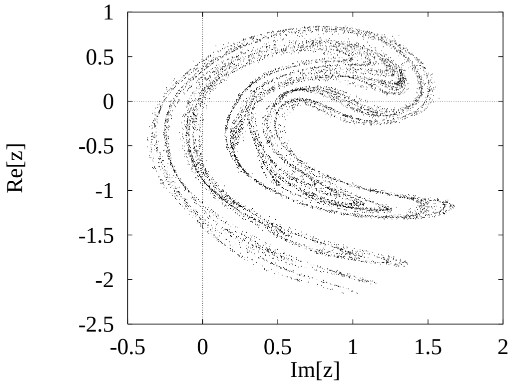
with $a = 1.0$, $b = 0.9$, $c = 6$, and $\phi = 0.4$. The Ikeda map corresponds to the plane-wave interactivity in an optical ring laser. The real and the imaginary parts of the Ikeda series are shown in Figure 5.7a.

Just as in the Lorenz attractor time series, there are two data streams to be predicted in this time series. Here, we need only one stream (the real part) as input, and the prediction was made on both real and the imaginary streams for the same reason stated in the Lorenz series.

Again, in existing work, we used the linear AR model, DRNN, and FIR Networks. One is an DRNN that contains less number of weights in the network than the FIR



(a) Actual series



(b) Predicted series

Figure 5.7: Phase plot of the Ikeda map time series: real part $Re[z]$ versus imaginary part $Im[z]$. (a) Original Ikeda attractor phase plot. (b) Predicted Ikeda attractor phase plot. Parameters used to generate (b) are listed in Table 5.7.

Table 5.7: Normalized single-step prediction errors for the Ikeda map by using different prediction models. Results for Autoregressive and DRNN are taken from [8], and results for the FIR Netowork are taken from [169].

Model		Linear Model	FIR Network	DRNN	VGBP
nMSE	$Re[z]$	0.640	0.0080	0.0063	**0.00023**
	$Im[z]$	0.715	0.0150	0.0134	**0.00020**
Number of weights		–	2227	587	**558**
Nodes		–	1:25:25:2	1:15:2	1:22:2
Taps		–	5:2:2	5:1:0	0:0

network shown in Table 5.7. Our network is a RFIR network with the smallest size as compared to DRNN and FIRNN used for this problem.

For Ikeda, Table 5.7 shows that RFIR trained by VGBP can achieve about 30 times better nMSE in the real part and more than 60 times better in the imaginary part, while using slightly less number of weights than DRNN, and about one fourth of the number of weights used by Wan's FIR network.

Figure 5.7b shows the phase plot (predictions on the real part versus predictions on the imaginary part) of our iterative prediction. It is hard to distinguish Figure 5.7a (which is the phase plot on real data) from Figure 5.7b (which is the phase plot on predictions). Hence, we conclude that our iterative predictions capture the relationship between the real and the imaginary parts remarkably well.

5.2.5 Summary of chaotic time-series benchmarks

In all the four chaotic time-series benchmarks studied, we have used less number of weights than both Wan's FIR network and Aussem's DRNN network. Table 5.8 compares the single-step performance of our RFIR network trained by VGBP for all four sets of time series with respect to results in previous work. For a more complete comparison, we also include the results of carbon-copy (C.C.) in Table 5.8. Carbon-copy simply predicts the next data value to be the same as the current data value. Throughout all the experiments conducted, the three sophisticated predictors (Wan's, Aussem's, and our predictors) perform much better than the two simple predictors (AR and C.C.) (also provided by Wan [169]). Our results show that it is worthwhile to use more sophisticated models in nonlinear time-series predictions. Wan's FIR NN and Aussem's DRNN did reasonably well on these two sets of time series. But with significantly less weights, our proposed network can consistently achieve much smaller nMSE than both of their results.

The extremely encouraging results on noiseless time-series benchmarks gave us great confidence in applying our proposed neural network in more challenge financial time-series predictions.

Table 5.8: Comparison of single-step-prediction performance in $nMSE$ using five methods: Carbon copy (C.C), a linear model (AR), FIR [169], DRNN [8], and VGBP. Carbon copy simply predicts the next time-series data to be the same as the proceeding data ($x(t+1) = x(t)$). The training (resp. testing) set indicates patterns used for learning (resp. testing). *Lorenz attractor* has two data streams labeled by x and z, respectively, whereas *Ikeda attractor* has two streams – real ($Re(x)$) and imaginary ($Im(x)$) parts of a plane wave. Boxed numbers indicate the best results. N/A stands for data not available.

Bench-Mark	Training Set	Testing Set	Performance Metrics		Design Methods				
					C.C.	AR	FIR	DRNN	VGBP
MG17	1-500	501-2000	$nMSE$		0.6686	0.320	0.00985	0.00947	$\boxed{0.000057}$
			# of weights		0	N/A	196	197	$\boxed{121}$
MG30	1-500	501-2000	$nMSE$		0.3702	0.375	0.0279	0.0144	$\boxed{0.000374}$
			# of weights		0	N/A	196	197	$\boxed{121}$
Henon	1-5000	5001-10000	$nMSE$		1.633	0.874	0.0017	0.0012	$\boxed{0.000034}$
			# of weights		0	N/A	385	261	$\boxed{209}$
Lorenz	1-4000	4001-5500	$nMSE$	x	0.0768	0.036	0.0070	0.0055	$\boxed{0.000034}$
				z	0.2086	0.090	0.0095	0.0078	$\boxed{0.000039}$
			# of weights		0	N/A	1070	542	$\boxed{527}$
Ikeda	1-10000	10001-11500	$nMSE$	$Re(x)$	2.175	0.640	0.0080	0.0063	$\boxed{0.00023}$
				$Im(x)$	1.747	0.715	0.0150	0.0134	$\boxed{0.00022}$
			# of weights		0	N/A	2227	587	$\boxed{574}$

Chapter 6

Predicting Stock-Market Time Series

In the past twenty years, there were three Makridakis competitions (also known as M-Competitions [88], M2-Competitions [89] and M3-Competitions [90]) held to test forecasting accuracy, including financial and economic time series. A variety of models were tested in those competitions. In those three competitions, results from many different models were submitted. Models employed include ARIMA, exponential smoothing, *kNN*, GARCH, ANNs, and also expert systems. An expert system may employ a collection of models, including exponential smoothing methods, ARIMA methods and moving average, develop a set of rules for selecting a specific method [39, 103], and select one for activation when certain conditions are satisfied. The conclusions reached were rather consistent. We list below some related conclusions drawn from these competitions and the literature [51, 59, 100].

(a) No single method is clearly superior to other methods in most time series tested.

(b) Existing methods do not out-perform random-walk models significantly and statistically in terms of both prediction accuracy and prediction directions (up/down trends). In some cases, they are even worse than random walks.

In this chapter, we apply our neural network model, which has been successful in predicting chaotic and piecewise chaotic time series, to financial time series to see whether it can predict financial market better than random walk.

6.1 Experiments setup

In this chapter, we first show the effect of our proposed preprocessing approaches. To do that, we conduct experiments based on traditional low-pass preprocessing and experiments based on our proposed channel-specific preprocessing. Next, to show that our proposed ANN model out-performs traditional statistical models, we also conduct experiments using an autoregressive model (served as a benchmark predictor).

6.1.1 Predictors compared

We compare our proposed ANN model-based channel-specific preprocessing approach to ANN model based on traditional low-pass filtering preprocessing and autoregressive model.

1. Predictor AR. We apply the autoregressive model discussed in Section 2.1.1.1 to predict future stock prices. The software used to train the AR model was developed by Hegger and Schreiber in the $TISEAN$ software package [58]. We tried a variety of AR models with different orders n in order to find the best $AR(n)$ for predicting

stock-price time series directly. The performance of AR model (Predictor AR) will be served as the benchmark for experiments conducted in this chapter.

2. Predictor CNN-LP. It is an ANN model trained by VGBP based on traditional low-pass preprocessing. We first perform preprocessing on stock-price time series using low-pass filters designed in Section 3.2. We then apply the ANN model developed in this work on each set of preprocessed signals and train the ANN model by VGBP. Three low-pass filters considered are listed in Table 6.1: a filter with 6 taps and cut-off frequency of 0.15, a filter with 8 taps and cut-off frequency of 0.10, and a filter with 10 taps and cut-off frequency of 0.05.

3. Predictor CNN-CSP. ANN trained by VGBP based on channel-specific preprocessing. We first perform preprocessing on stock-price time series using the new channel-specific preprocessing approach presented in Section 3.3, which combines wavelet decomposition and channel-specific preprocessing on the resulting individual channels. The alternatives on different combinations of wavelet and low-pass filters to be used in our preprocessing are given in Section 3.3.4.2. We then apply ANN learning on each preprocessed channel and train the ANN model by VGBP.

Table 6.1: Three representative traditional low-pass filters and their corresponding combinations of wavelets and low-pass filters used in channel-specific preprocessing.

(a) Target low-pass filter 6/0.15

ID	Mother Filter	Filter for Channel H		Filter for Channel LH	
		Taps	f_{cutoff}	Taps	f_{cutoff}
B3_63_43	G_{B3}	6	0.15	4	0.15
B4_63_43	G_{B4}	6	0.15	4	0.15
B2_63_43	G_{B2}	6	0.15	4	0.15

(b) Target low-pass filter 8/0.10

ID	Mother Filter	Filter for Channel H		Filter for Channel LH	
		Taps	f_{cutoff}	Taps	f_{cutoff}
B3_82_62	G_{B3}	8	0.10	6	0.10
B4_82_62	G_{B4}	8	0.10	6	0.10
B2_82_62	G_{B2}	8	0.10	6	0.10
B4_82_42	G_{B4}	8	0.10	4	0.10

(c) Target low-pass filter 10/0.05

ID	Mother Filter	Filter for Channel H		Filter for Channel LH	
		Taps	f_{cutoff}	Taps	f_{cutoff}
B3_81_61	G_{B3}	8	0.05	6	0.05
B3_81_41	G_{B3}	8	0.05	4	0.05
B6_81_61	G_{B6}	8	0.05	6	0.05
B6_81_41	G_{B6}	8	0.05	4	0.05

6.1.2 Performance measures

We continue to use hit ratios defined in Eq. (3.1) and (3.2) to measure the performance of predictions, but with real predictors (AR, CNN-LP, and CNN-CSP) replacing the ideal (perfect) predictor. In CNN-CSP, we apply different combinations of parameters on the IBM stock-price time series and determine which combination is to be used. The selected combination will be generalized to the remaining nine sets of stock-price time series. As the spectra of the ten stock-price time series are very similar (see Section 3.2), we expect similar performance to hold on other nine benchmarks.

We present a simple trading strategy based on the predicted time series and measure the cumulative returns obtained when different predictors are used. The simple trading strategy is described as follows.

Initially, we have a certain amount of cash in our portfolio. We assume that the daily low/high prices are already known a few minutes before market closes at day t_0. Right before the market closes, the prediction on the stock price is performed. If the predicted price for t_0+1 is higher than the daily high price for t_0, then we buy the stock if cash is available; if the predicted price for t_0+1 is lower than the daily low price, then we sell the stock if there are shares in the portfolio. Then, right at the moment the market opens in day t_0+1, we immediately buy (resp. sell) as many shares of stocks as possible if the predicted price for t_0+1 is higher (resp. lower) than the opening price. We perform this simple trading simulation over the entire time period used in our experiments and compute the annual return of the portfolio according to (1.21) in Section 1.3.3.1. Table 6.2 illustrates the trading strategy over a period of 7 days.

In our experiments for stock-price time-series predictions, we only predict one day into the future, although we need to predict multiple days in the lag period when low-pass

Table 6.2: Illustration of the simple trading strategy used in our study. Column labeled by "Tx" represents the buy/sell transaction, and $P_L/P_H/P_O/P_C$ stand for daily low/high/open/close prices.

Prices			At t_0 opening			At t_0 closing			
t_0	$P_L/P_H/P_O/P_C$	$P_t(t_0+1)$	Tx	Cash	Shares	Tx	Cash	Shares	Portfolio
1	9.8/10.2/10.1/9.9	10.1	–	10000	0	–	10000	0	10000
2	9.7/10.1/10.0/9.9	10.2	Buy	0	1000	–	0	1000	10050
3	10.0/10.4/10.1/10.2	10.3	–	0	1000	–	0	1000	10200
4	10.0/10.3/10.2/10.1	10.2	–	0	1000	–	0	1000	10100
5	9.8/10.2/10.0/10.1	9.9	–	0	1000	Sell	10050	0	10050
6	10.0/10.2/10.0/10.1	10.2	–	10050	0	Buy	5	995	10050
7	10.1/10.4/10.4/10.2	10.2	Sell	10348	0	–	10348	0	10348

filtering is used. Due to nonstationarity, in predicting longer horizons, it is important to ensure that prediction errors do not propagate in the iterative predictions.

6.2 Experimental results on preprocessing

In order to identify the effect of preprocessing, we compare the prediction performance using traditional low-pass preprocessing represented by low-pass filters to the prediction performance using the corresponding channel-specific preprocessing approach. We list in Table 6.1 the representative traditional low-pass filters selected in Section 3.2, along with the alternative combinations of wavelet and low-pass filters selected for their corresponding channel-specific preprocessing in Section 3.3.

6.2.1 ANN Prediction on individual channels using channel-specific preprocessing

When traditional low-pass preprocessing is used, we need overcome the lags incurred by low-pass filtering and predict the preprocessed low-pass signals into the future. On the other hand, when channel-specific preprocessing is used, we need to predict three sets of preprocessed signals and combine three sets of predictions in order to obtain the predictions for the original time series. Due to the shift-invariant property of the redundant wavelet decomposition, we obtain the prediction of original signal $P_h(t)$ by applying the following equation

$$P_h(t) = P_h^1(t) + P_h^2(t) + P_h^3(t), \tag{6.1}$$

where $P_h^i(t)$ is the prediction for horizon h of the i^{th} channel.

As mentioned in Section 3.3, the prediction on the low-pass channel is rather accurate. However, the high-frequency channels account for a large portion of prediction errors when channel-specific preprocessing is used. As an example, consider the prediction of the IBM stock-price time series using the parameter combination of B3_82_62, i.e., using a mother filter of the B_3-spline filter, the low-pass filter in Channel H with 8 taps and cut-off frequency of 0.10 (along with 4-day lag), and the low-pass filter in Channel LH with 6 taps and cut-off frequency of 0.10 (along with 3-day lag).

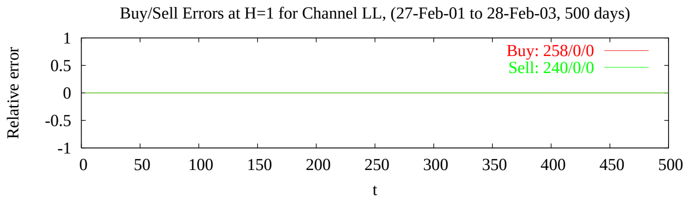
To show the channel that pose the biggest challenge in overall prediction, we present the prediction results in an additive way. Specifically, we first present the prediction results for Channel LL (i.e., $P_h^{LL}(t)$). We then add the prediction results for both Channels LL and LH (i.e., $P_h^{LL}(t) + P_h^{LH}(t)$). Finally, we add up the prediction results of all three channels (i.e., $P_h^{H}(t) + P_h^{LH}(t) + P_h^{LL}(t)$).

6.2.1.1 Predictions for Channel LL

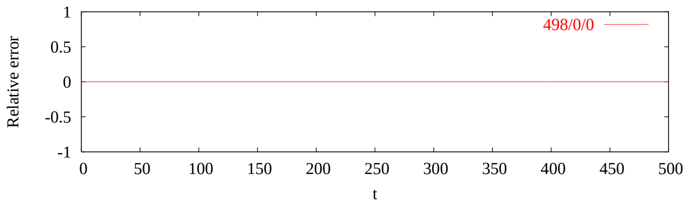
The predictions of the low-pass channel (Channel LL) is rather accurate as mentioned in Section 3.3. Consider the predictions of Channel LL of the wavelet decomposition using the B_3-spline wavelet. As Channel LL is a low-pass channel obtained directly from an appropriate low-pass filter. We can obtain the corresponding daily low/high prices by performing the same wavelet decomposition on the daily low/high prices.

To predict Channel LL, we use an FIR neural network with a structure of 1:20:1 (i.e., 1 input node, 20 hidden nodes, and 1 output node), and the filter structure of 15:0:0 (i.e., 15-tap filter coming into the input node, and no FIR filters for connections between the input and the hidden layers and between the hidden and the output layers). The activation function for the hidden nodes are the tanh(x) functions, and the output node uses a linear function of $y = x$. To predict future price values, the 200 most recent prices are used for training.

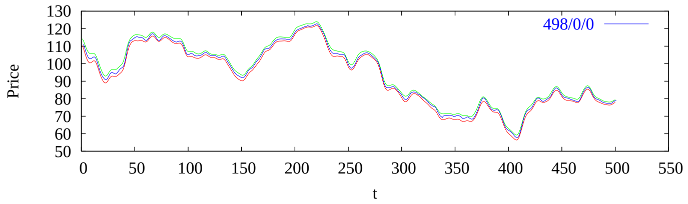
Figures 6.1(a) and 6.1(b) show the one-step prediction performance in terms of errors w.r.t. potential buy/sell signals and errors w.r.t. daily low/high prices. Figure 6.1(c) shows the actual prediction values. The graphs show that the one-step prediction on Channel LL is very accurate w.r.t. Channel LL of the daily low/high prices. In fact, there is no error w.r.t. daily low/high prices and potential buy/sell signals from Feb 27, 2001 up to Feb 28, 2003. As a result, there is no data point shown in these two graphs. Figure 6.1(c) clearly shows that the predicted values are well confined by the low/high prices.



(a) Prediction errors w.r.t. potential buy/sell signals



(b) Prediction errors w.r.t. the Channel LL of daily low/high prices



(c) Prediction values when compared with daily low/high prices

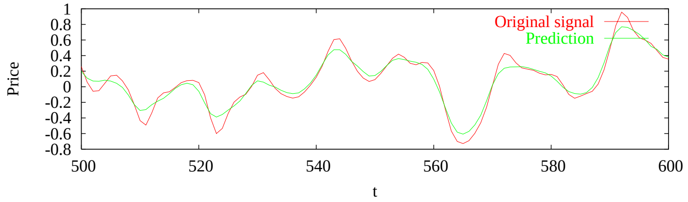
Figure 6.1: Prediction performance on Channel LL of IBM stock-price time series after 3-channel wavelet decomposition with a mother filter of B_3-spline filter. The relative error is the prediction error divided by the actual price.

6.2.1.2 Predictions on Channel L (a combination of Channel LH and LL)

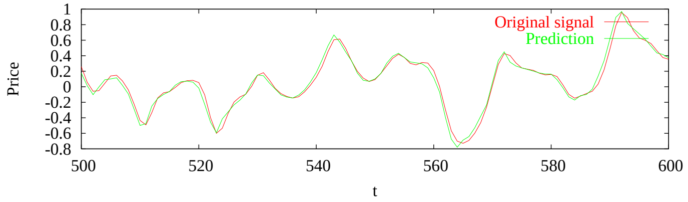
Channel LH's daily low (resp. high) price time series is the Channel LH of the wavelet decomposition of daily low (resp. high) price time series using the same wavelet decomposition. Since Channel LH's daily low and high prices cross each other, there are no meaningful hit ratios for the predictions on this channel alone. As a result, we will not evaluate predictions on Channel LH directly. Instead we combine Channels LL and LH into a single Channel L and present the predicted values in the lag period along with horizon 1 for Channel LH.

Note that the combination of Channels LH and LL is in fact the low-frequency channel (Channel L) of the same wavelet decomposition but with only 2 channels. Therefore, the corresponding low/high prices for the combination of Channels LH and LL is Channel L of the wavelet decomposition on the original daily low/high prices. When low-pass filter 6/0.10 is applied to Channel LH, a 3-day lag is incurred. To predict Channel LH, we have to overcome the 3-day lag. Here we give the prediction results in Lags 1 to 3 to show the prediction performance in the lag period.

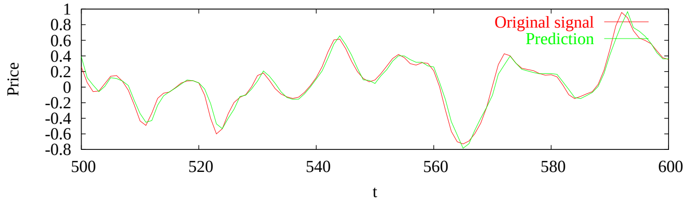
To predict Channel LH, we use an FIR neural network with a node structure of 1:12:1 (i.e., 1 input node, 12 hidden nodes, and 1 output node), and a filter structure of 15:0:0 (i.e., 15-tap filter coming into the input node, and no FIR filters used for connections between the input and the hidden layers and connections between the hidden and the output layers). The activation function for hidden nodes is the tanh(x) function, and the output node uses a linear function of $y = x$. To predict future price values, the 120 most recent data are used for training. We have tried window sizes between 100 and 200, and found that there is no single window size that performs much better than other.



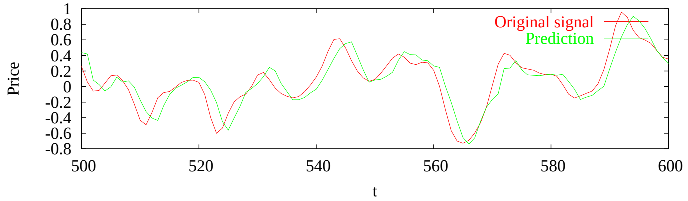
(a) Prediction for Channel LH at Lag=1



(b) Prediction for Channel LH at Lag=2



(c) Prediction for Channel LH at Lag=3



(d) Prediction for Channel LH at Horizon of 1

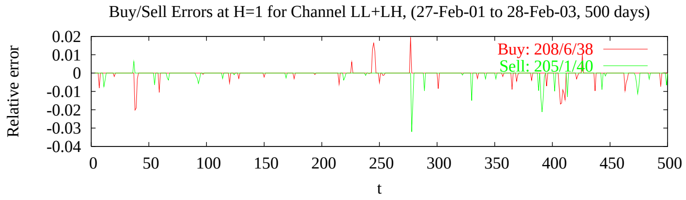
Figure 6.2: Predicted values for 3 lags and horizon 1 of Channel LH.

Figures 6.2(a) to 6.2(c) show that the predictions for all three lags are fairly accurate. Figure 6.2(d) shows that the predictions at horizon of 1 are not synchronized with the original Channel LH data; i.e. there are lags in the predictions of horizon 1. This is the price we have to pay because we have introduced 3 lags by applying a 6-tap non-causal filter to perform denoising in Channel LH.

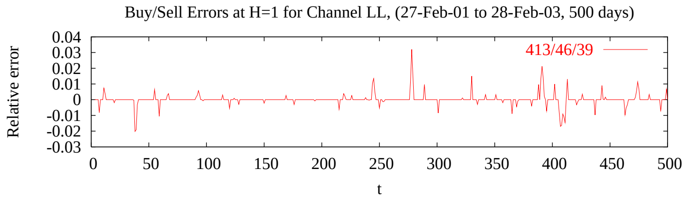
Figure 6.3(a) shows there are 208 predictions without errors w.r.t. potential buy signals, as compared to 6 positive prediction errors and 38 negative prediction errors. Meanwhile, there are 205 predictions without error w.r.t. potential sell signals as compared to 1 positive prediction error and 40 negative prediction errors. Overall, the ratio of predictions without error is $\frac{208+205}{208+6+38+205+1+40} = 82.9\%$. Figure 6.3(b) shows the number of zero/positive/negative prediction errors w.r.t. daily low/high prices are 413/46/39, which also confirms that there are 82.9% predictions without error. Figure 6.3(c) shows that the predictions generally fall into the daily low/high price range. However, the predictions may fall out of the low/high price range in the turning points of the time series. This is understandable as the patterns around the turning points might not be anticipated by the ANN model trained. Overall, the predictions on the combined LH and LL channels are still rather accurate.

6.2.1.3 Predictions on all 3 channels combined

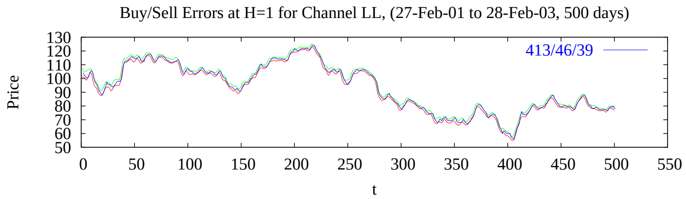
Channel H's daily low (resp. high) price time series is the Channel H of the wavelet decomposition on the daily low (resp. high) price time series using the same wavelet decomposition. Also, for the same reason as discussed for Channel LH, there are no meaningful hit ratios for the predictions on Channel H alone. Hence, we will not evaluate the predictions on Channel H directly. But instead we always combine all three



(a) Prediction errors w.r.t. potential buy/sell signals



(b) Prediction errors w.r.t. daily low/high prices



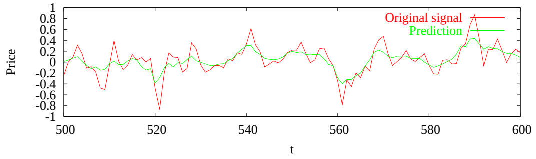
(c) Prediction values compared with daily low/high prices

Figure 6.3: Prediction performance on combined Channels LH and LL of IBM stock-price time series after 3-channel wavelet decomposition with a B_3-spline mother filter.

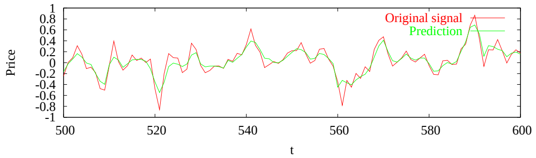
channels (LL, LH, and H) when evaluating Channel H. Since Channel H contains a lot of high-frequency noise, we apply a low-pass filter 8/0.10 to perform denoising, thereby introducing a 4-day lag. To predict Channel H, the same network used for training Channel LH is employed. Also, a window of 120 most recent data available is used for training. In fact, in experiments we have conducted, the prediction results are not very sensitive to different ANN node structures, FIR filter structures, and different window sizes around the parameters used. Here we give the prediction results in Lags 1 to 4 to show the prediction performance in the lag period.

Figures 6.4(a) to 6.4(d) show that the predictions for all four lags are fairly accurate. Figure 6.4(e) shows that the predictions at horizon of 1 are not synchronized with the original Channel H data. This is also the price paid for the 4-day lags incurred when using the 8-tap non-causal filter for denoising. To correct the lag, we performed postprocessing described in Section 6.2.2. In the following, the results are obtained after postprocessing has been performed.

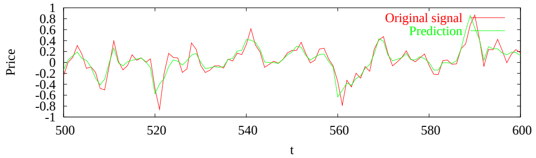
When all three channels are added together, the daily low/high prices are just the original daily low/high prices. Figure 6.5(a) shows that there are 160/16/66 zero/positive/negative prediction errors w.r.t. potential buy signals, and 155/21/80 zero/positive/negative prediction errors w.r.t. potential sell signals. Overall, the number of zero/positive/negative prediction errors w.r.t. daily low/high prices are 315/96/87 (Figure 6.5(b)). This means there are about 63.3% predictions having zero prediction errors w.r.t. daily low/high prices, or w.r.t. potential buy/sell signals. Compared to the 68.4% ratio of zero prediction errors by ideal predictor (see IBM results in Table 3.4), the ratio of 63.3% is quite respectable. Figure 6.5(c) further shows that a significant number of predictions are inside the daily low/high price range.



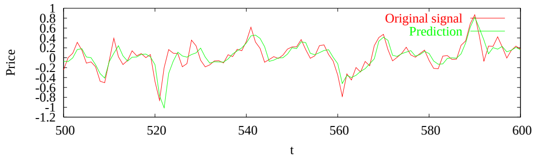
(a) Predictions for Channel H at Lag=1



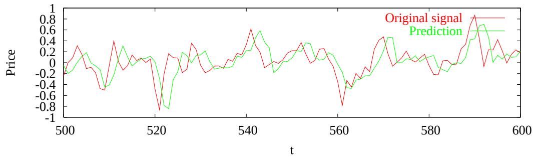
(b) Predictions for Channel H at Lag=2



(c) Predictions for Channel H at Lag=3

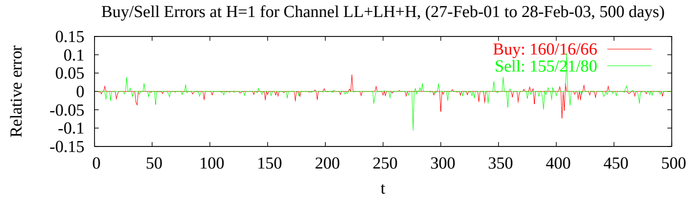


(d) Predictions for Channel H at Lag=4

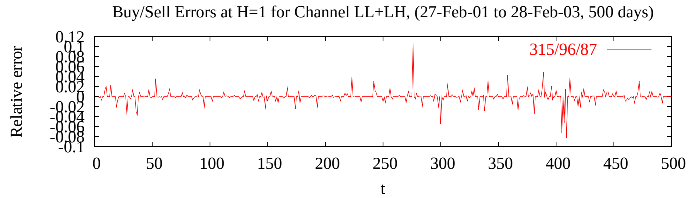


(e) Predictions for Channel H at Horizon of 1

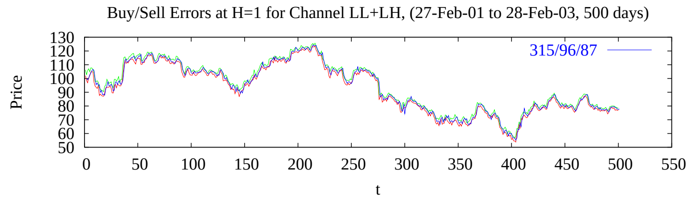
Figure 6.4: Predicted values for 4 lags and horizon 1 of Channel H.



(a) Prediction errors w.r.t. potential buy/sell signals



(b) Prediction errors w.r.t. daily low/high prices



(c) Prediction value compared with daily low/high prices

Figure 6.5: Prediction performance on all three channels combined for IBM stock-price time series after 3-channel wavelet decomposition with a B_3-spline mother filter.

Based on the above experimental results, we can draw following conclusions. Channel LL is easy to predict when compared to Channels H and LH. Currently, the prediction performance is mainly dictated by Channels H and LH.

6.2.2 Postprocessing

As stated in Sections 6.2.1.2 and 6.2.1.3, it is hard to overcome the 3-day lag when predicting Channel LH and to overcome the 4-day lag when predicting Channel H. As a result, predictions on horizon 1 also exhibit lags. To correct these lags, we have developed a very simple heuristic postprocessing technique. The postprocessing procedure at current day t_0 is as follows.

Step 1 Detection at Day t_0

- Set $pp = 1$ if $P_{t_0-1}(t_0) > R_H(t_0)$ or $P_{t_0-1}(t_0) < R_L(t_0)$.
- If $pp = 1$ and $R_L(t_0) \leq P_{t_0-1}(t_0) \leq R_H(t_0)$, then set $pp = 0$.

Step 2 Modification for $P_{t_0}(t_0 + 1)$

- If $pp = 1$, then set current day's prediction to be $P_{t_0}(t_0+1) = 0.25(R_L(t_0) + R_H(t_0)) + 0.5R_C(t_0)$; otherwise leave $P_{t_0}(t_0 + 1)$ unchanged.

The main idea of the postprocessing is to detect if the previous prediction $P_{t_0-1}(t_0)$ at Day t_0 is out of the actual daily low/high price range at Day t_0. If it is out of range, then a correction is applied to current day's prediction of $P_{t_0}(t_0 + 1) = 0.25(R_L(t_0) + R_H(t_0)) + 0.5R_C(t_0)$. If we detect that the previous prediction is within desired low/high price range, then no correction is performed.

6.2.3 Parameters used in channel-specific preprocessing

As discussed in Section 3.3, for each target low-pass filter, there are several combinations of mother filters (for wavelet decomposition) and low-pass filters for each individual channel. Since it takes about four days to finish the experiments on one set of parameters, we were not able to conduct experiments on all the combinations for all ten stock-price time-series benchmarks. Instead, we select one parameter set based on the evaluation results of various parameter sets on the IBM stock-price time series.

The same three ANN architectures described in Section 6.2.1 are used for the three channels for all different parameters sets combined.

Table 6.3 presents the prediction performance (including the predictions on individual channels, reconstructions from individual channel's predictions, and postprocessing) when different parameter combinations are used for channel-specific preprocessing. We also list the prediction performance when the target low-pass filters are used. Table 6.3 shows that there is virtually no difference between the prediction performances in terms of various hit rates among different parameter alternatives when channel-specific preprocessing is used. Further, they all have better prediction performance than traditional low-pass preprocessing approaches. The fact that there is little performance difference among different parameter alternatives is somewhat surprising. This may be caused by two reasons. First, we do not perform low-pass filtering on Channel LL, which is highly predictable and dominates Channels H and LH. When compared with traditional low-pass preprocessing, our channel-specific preprocessing preserves more information in the low-frequency end that is the dominant part in the spectrum. Second, as long as the low-pass filters used in Channels H and LH do not have a large number of taps or a high cut-off frequency, the improved predictability of the resulting signals trades off well with the degree of information loss. When a longer filter is used, the ease of training trades off well with the need in overcoming longer lags.

Based on the prediction results here and the simple trading strategy in Sector 6.1.2, we build a portfolio with initial cash value of $1000000. The annual returns achieved when different preprocessing is used are listed in the last column of Table 6.3. Figure 6.6 shows the evolution of portfolio values with time. Here, we only compare the performance of CNN-LP and CNN-CSP using different parameters in order to decide on the best parameters to be used. We include results of Predictor AR and Buy-and-Hold strategy in the next section when we present experimental results on stock market predictions. Unlike the performance measured by hit ratios, the portfolios based on channel-specific preprocessing behave differently, and the one using $B3\_63\_43$ is better than the cases where other parameters or traditional low-pass preprocessing are used. As a result, we will use $B3\_63\_43$ for channel-specific preprocessing, and low-pass filter 6/0.15 to represent traditional preprocessing.

6.3 Experimental results on stock market predictions

In this section, we compare the prediction performance using three predictors described in Section 6.1.1: AR model, ANN model with traditional low-pass filtering, and ANN model with our proposed channel-specific preprocessing. All ANNs are trained by violation guided backpropagation presented in Chapter 4. For stocks with a long history, we evaluate the prediction performance roughly from the beginning of November, 1992 up to the end of February, 2003. For stocks with a short trading history, such as Yahoo and PFGI, we will use all the historical data through February 28, 2003 for training and predictions.

Table 6.3: Prediction performance on IBM stock-price time series using ANN models with different channel-specific preprocessing alternatives. The parameter combinations in the column labeled "Parameter" can be found in Table 6.1. LP T/f represents a T-tap low-pass filter with cut-off frequency f. The tuple (a, b) stands for a ratio of $a\%$ and average relative error of b, where relative error is defined as the prediction error divided by the actual price. The annual return is calculated using the simple strategy described in Section 6.1.2.

Parameters	Error w.r.t. low/high price range			Error w.r.t. potential buy/sell			Annual
	Zero	Positive	Negative	Zero	Positive	Negative	Return
Filter 6/0.15	0.6357	(0.1644,0.0084)	(0.1998,-0.0086)	0.6352	(0.0620,0.0127)	(0.3029,-0.0086)	15.68%
B3_63_43	0.6709	(0.1490,0.0080)	(0.1801,-0.0078)	0.6703	(0.0803,0.0125)	(0.2494,-0.0080)	24.16%
B4_63_43	0.6704	(0.1486,0.0081)	(0.1810,-0.0078)	0.6704	(0.0803,0.0124)	(0.2493,-0.0080)	19.00%
B2_63_43	0.6713	(0.1486,0.0081)	(0.1801,-0.0078)	0.6709	(0.0800,0.0125)	(0.2491,-0.0080)	21.05%
Filter 8/0.10	0.6311	(0.1702,0.0093)	(0.1987,-0.0091)	0.6304	(0.0655,0.0119)	(0.3041,-0.0097)	19.16%
B3_82_62	0.6709	(0.1490,0.0080)	(0.1801,-0.0078)	0.6703	(0.0803,0.0125)	(0.2494,-0.0080)	17.76%
B4_82_62	0.6704	(0.1486,0.0081)	(0.1810,-0.0078)	0.6704	(0.0803,0.0124)	(0.2493,-0.0080)	16.84%
B2_82_62	0.6713	(0.1486,0.0081)	(0.1801,-0.0078)	0.6709	(0.0800,0.0125)	(0.2491,-0.0080)	21.92%
B4_82_42	0.6704	(0.1486,0.0081)	(0.1810,-0.0078)	0.6704	(0.0803,0.0124)	(0.2493,-0.0080)	19.10%
Filter 10/0.05	0.6284	(0.1710,0.0098)	(0.2006,-0.0097)	0.6279	(0.0682,0.0118)	(0.3039,-0.0104)	20.97%
B3_81_61	0.6703	(0.1496,0.0080)	(0.1801,-0.0078)	0.6703	(0.0803,0.0125)	(0.2494,-0.0080)	17.28%
B3_81_41	0.6703	(0.1496,0.0080)	(0.1801,-0.0078)	0.6703	(0.0803,0.0125)	(0.2494,-0.0080)	16.43%
B6_81_61	0.6694	(0.1493,0.0080)	(0.1813,-0.0078)	0.6689	(0.0803,0.0124)	(0.2509,-0.0080)	22.07%

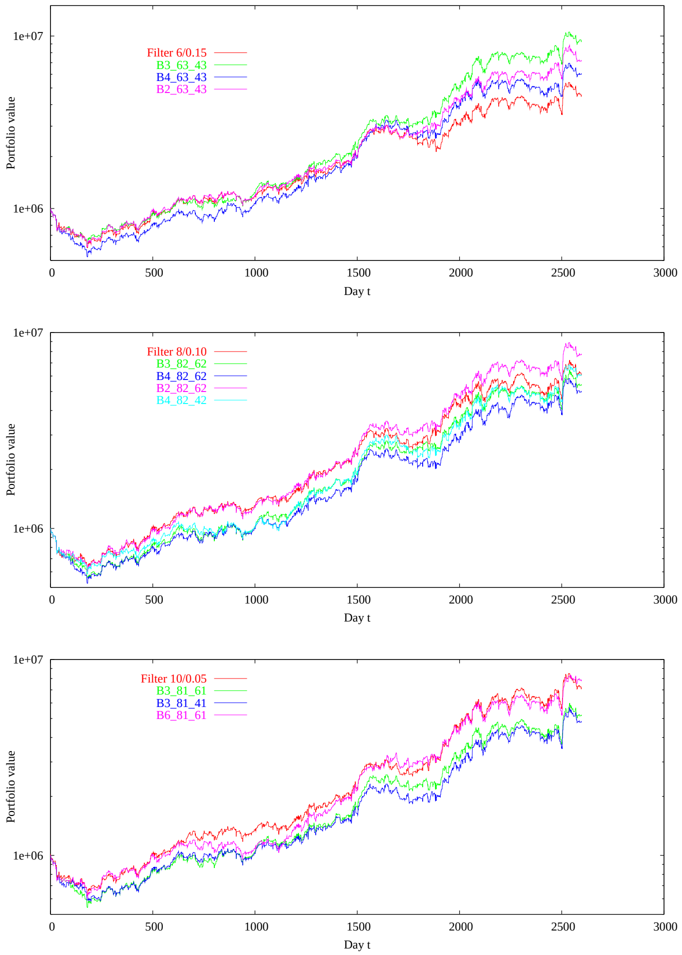


Figure 6.6: Portfolio value (with initial value of $1000000) grows with time. Each panel plots the evolution of portfolio values with time when a target filter and its corresponding channel-specific preprocessing are used. The upper, middle, and lower panels are for target filters of 6/0.15, 8/0.10, and 10, 0.05 respectively.

For the AR model, we have tried different orders and have found that $AR(10)$ is slightly better than $AR(5)$, $AR(20)$, and $AR(30)$. Therefore, we use $AR(10)$ in Predictor AR in our experiments. The ANN structure for Predictor CNN-LP is exactly the same as the ANN used for Channel H of Predictor CNN-CSP in Section 6.2.1, and a training window size of 200 is used for Predictor CNN-LP as well. For Predictor CNN-CSP, the ANNs used in the three decomposed channels are the same as those in Section 6.2.1.

Figures 6.7 to 6.16 plot the prediction errors w.r.t. daily low/high prices for the ten stock benchmarks. In each figure, there are three panels for the three predictors. The legend in the form $a/b/c$ in each panel represents the number of predictions with zero/positive/negative errors. In all the experiments we have conducted here, the autoregressive model has the worst performance, whereas CNN-CSP outperforms CNN-LP consistently. Since the only difference between Predictor CNN-LP and CNN-CSP is in their preprocessing, it is easy to conclude that channel-specific preprocessing used in CNN-CSP works better with respect to range errors.

Figures 6.17 to 6.26 also show the prediction errors w.r.t. the potential buy/sell signals discussed in Section 3.3. A similar conclusion on the performance of the three predictors is also reached.

Table 6.4 summarizes the hit ratios along with their average magnitudes. For hit ratios w.r.t. daily low/high prices. It is desirable to have a high hit ratio on zero errors and a low magnitude in average positive/negative prediction errors. For hit ratios w.r.t. potential buy/sell decisions, it is desirable to have a low hit ratio and a small magnitude on negative errors, since those negative errors are the ones that cause the loss of money if they are executed. Based on these measures, it is very clear that CNN-CSP is the winner.

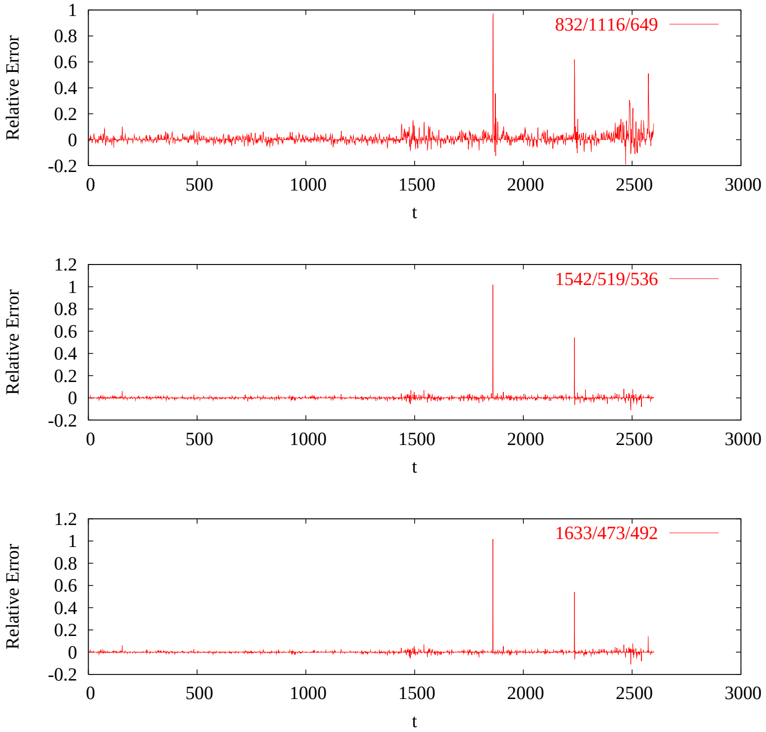


Figure 6.7: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for AMR Inc using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

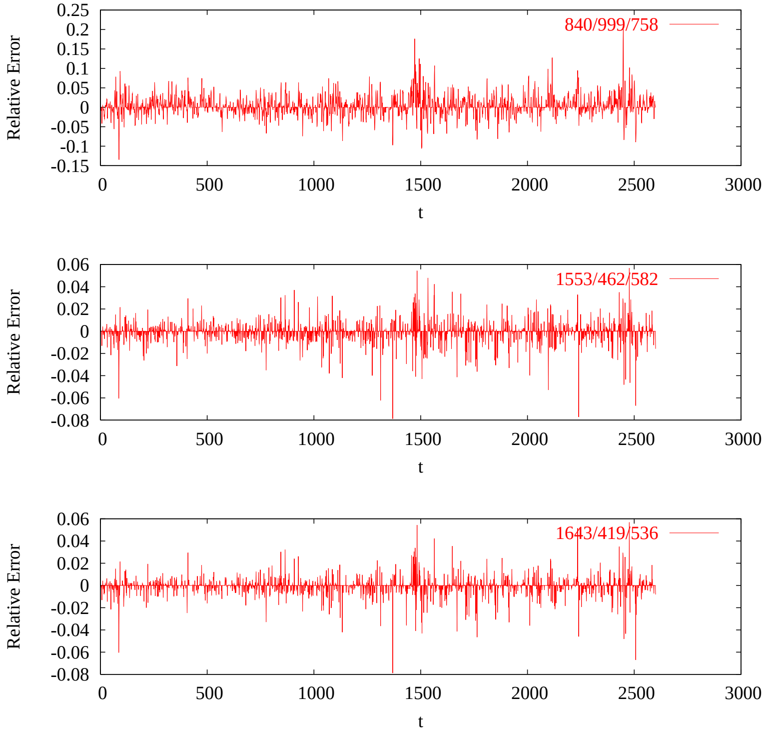


Figure 6.8: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Citigroup using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

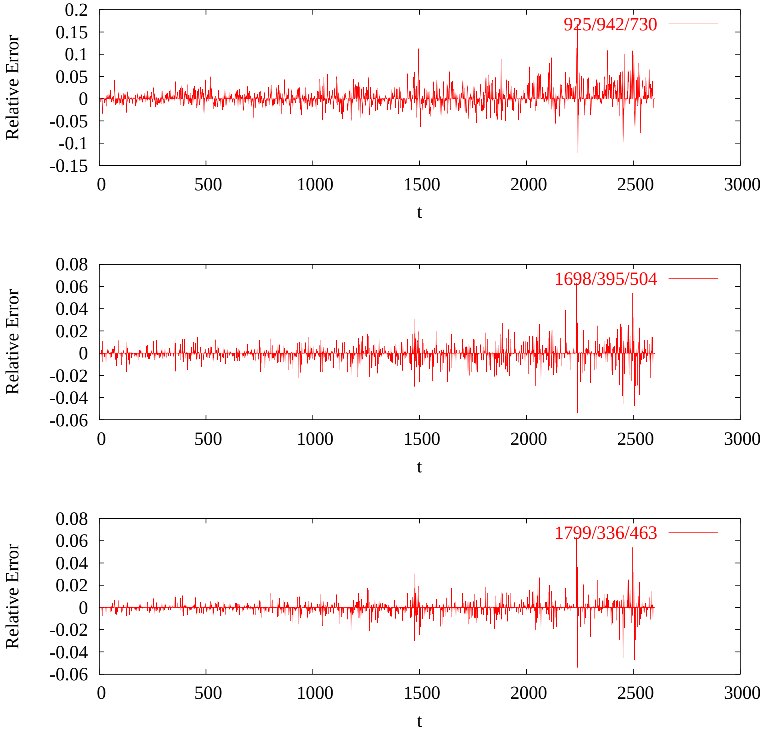


Figure 6.9: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for General Electric using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

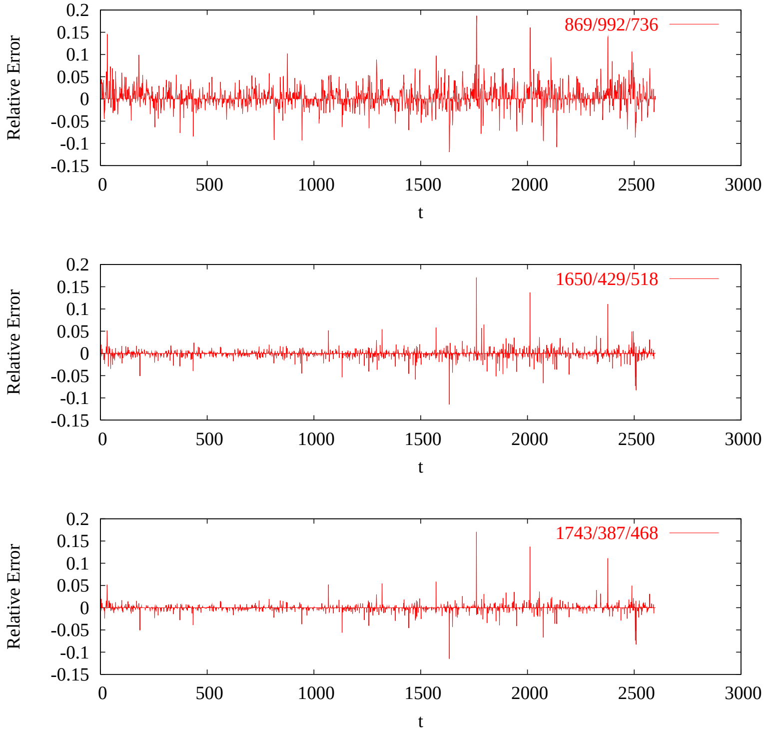


Figure 6.10: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for IBM stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

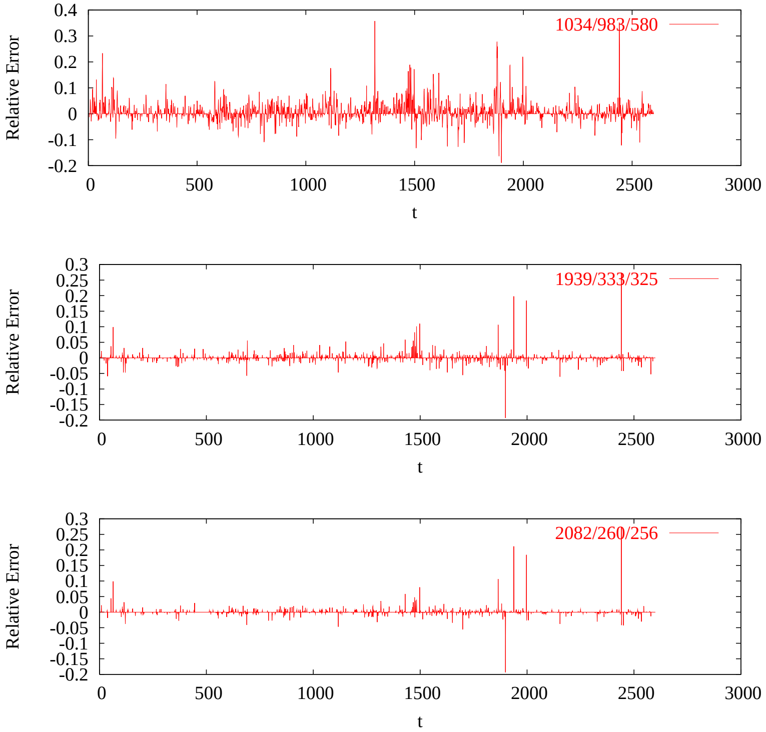


Figure 6.11: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Mentor using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

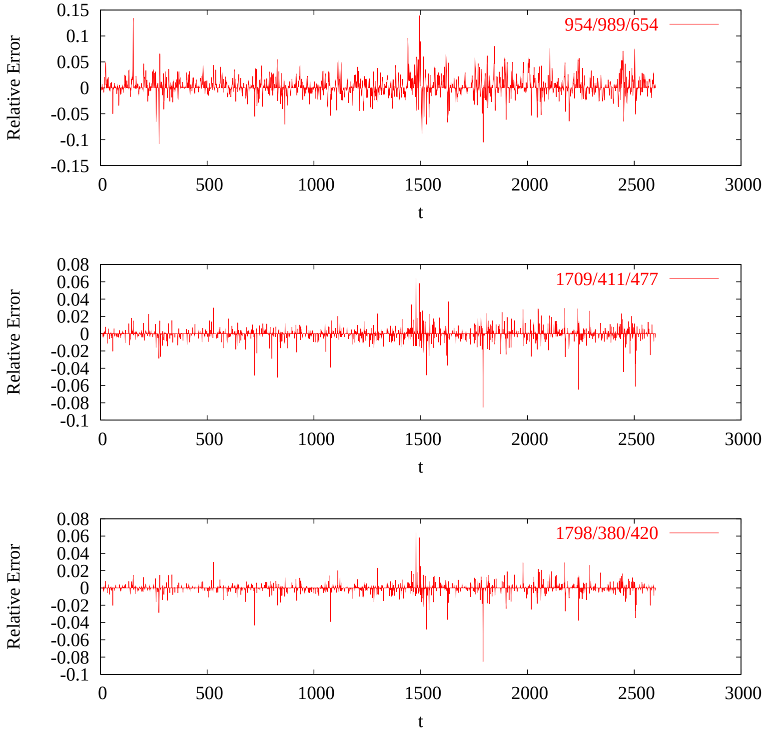


Figure 6.12: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for New York Times using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

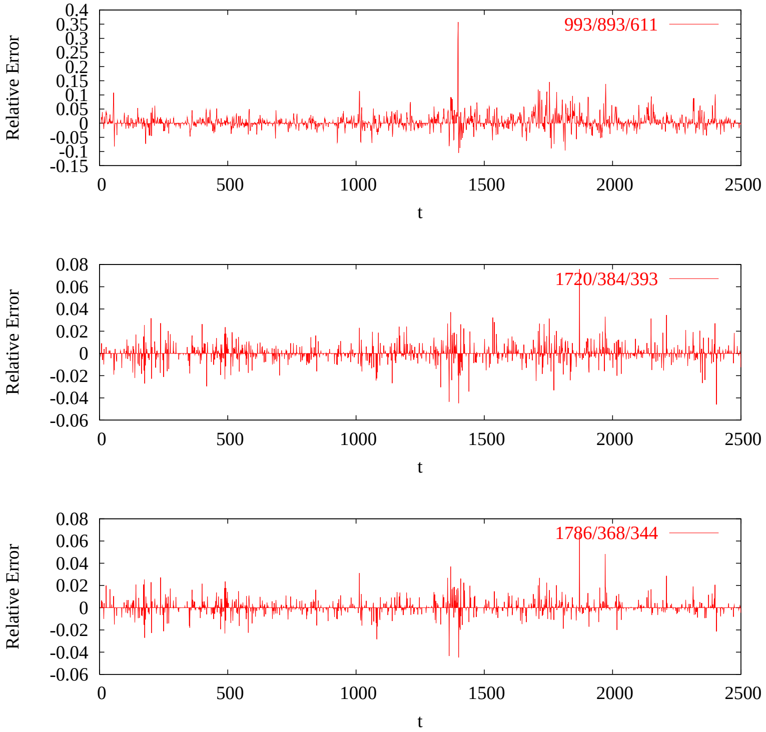


Figure 6.13: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Provident Financial Group using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

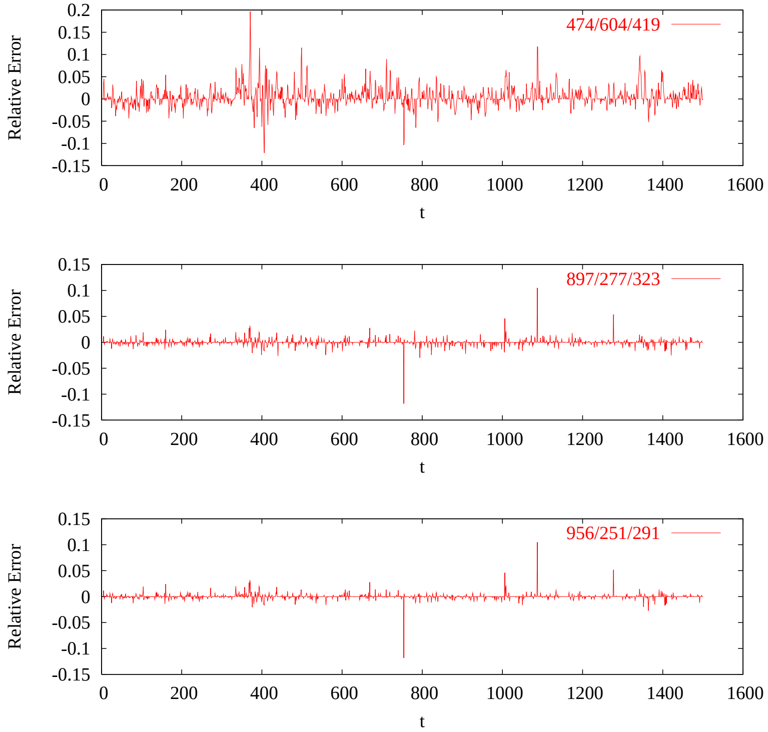


Figure 6.14: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Payless ShoeSource using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

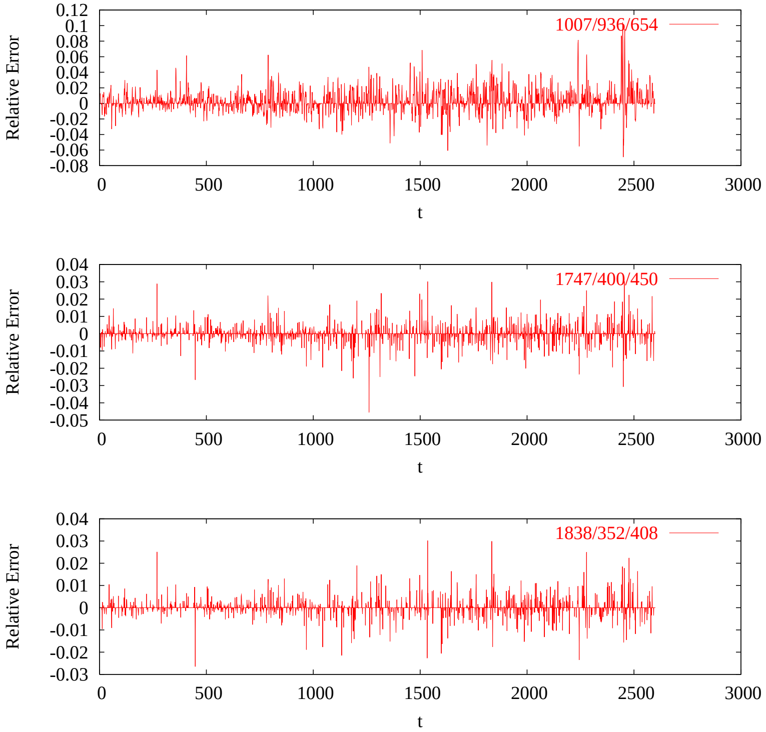


Figure 6.15: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Exxon-Mobil using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

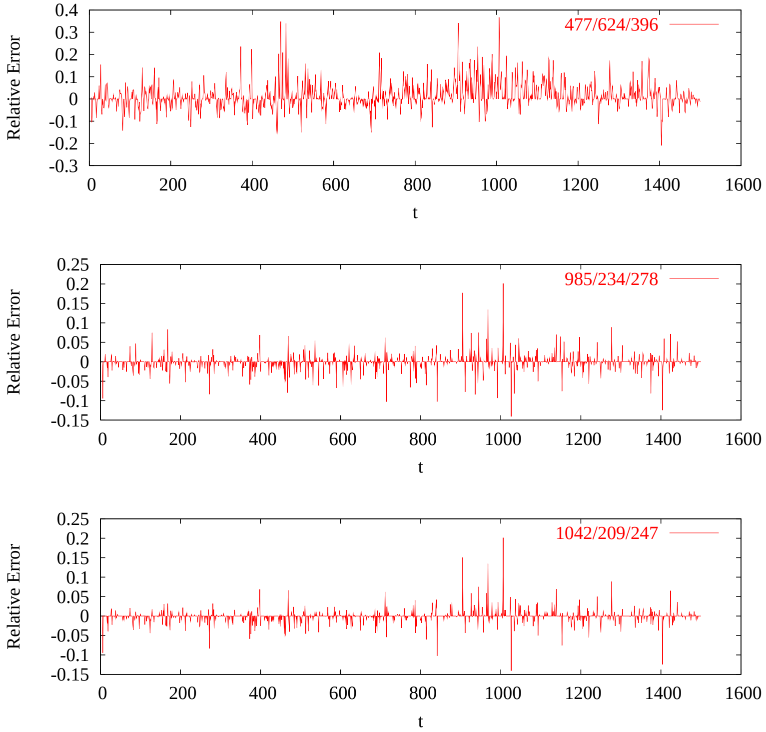


Figure 6.16: Prediction errors w.r.t. daily low/high stock prices (from November 4, 1992 to February 28, 2003) for Yahoo using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

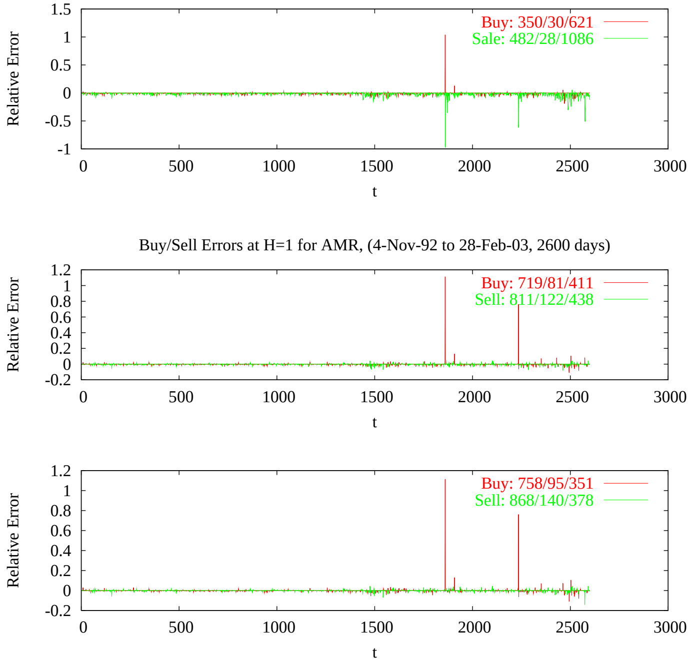


Figure 6.17: Prediction errors w.r.t. potential buy and sell signals for AMR Inc stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

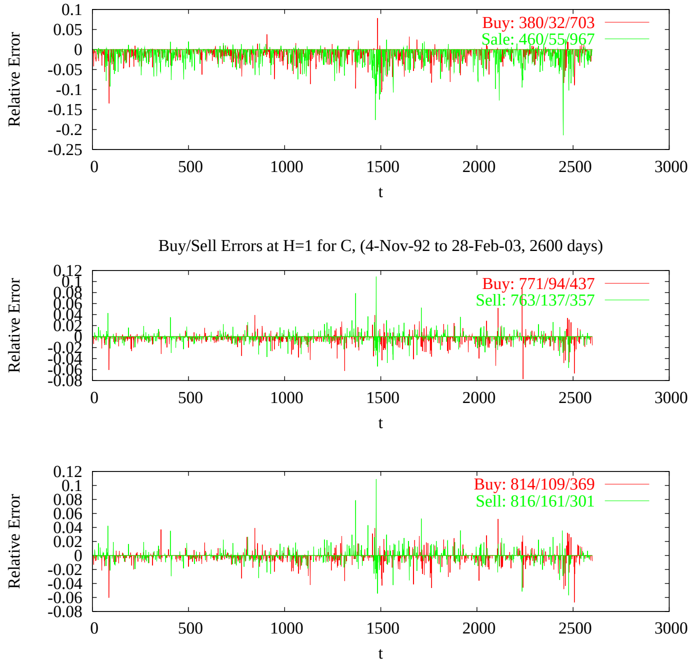


Figure 6.18: Prediction errors w.r.t. potential buy and sell signals for Citigroup stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

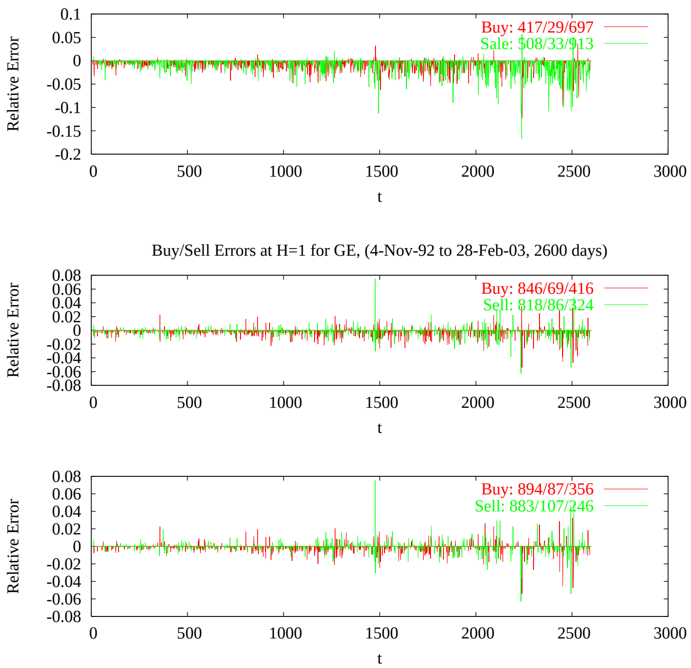


Figure 6.19: Prediction errors w.r.t. potential buy and sell signals for General Electric stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

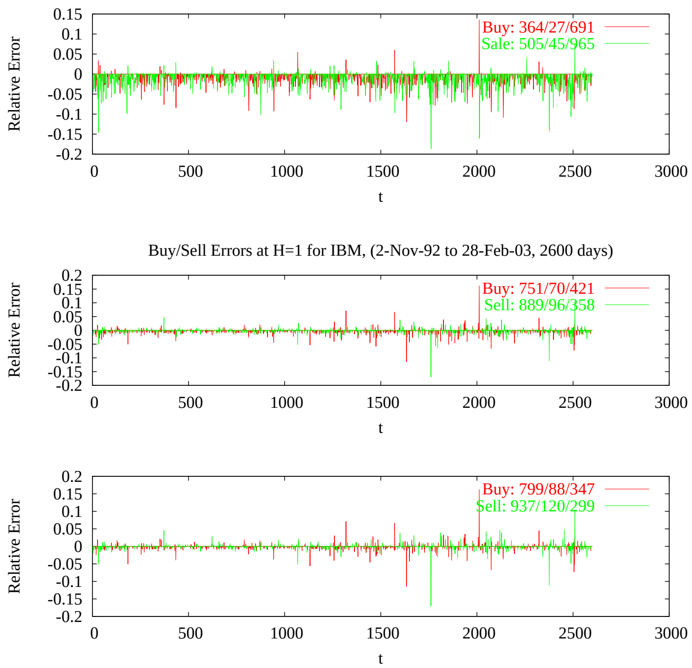


Figure 6.20: Prediction errors w.r.t. potential buy and sell signals for IBM stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

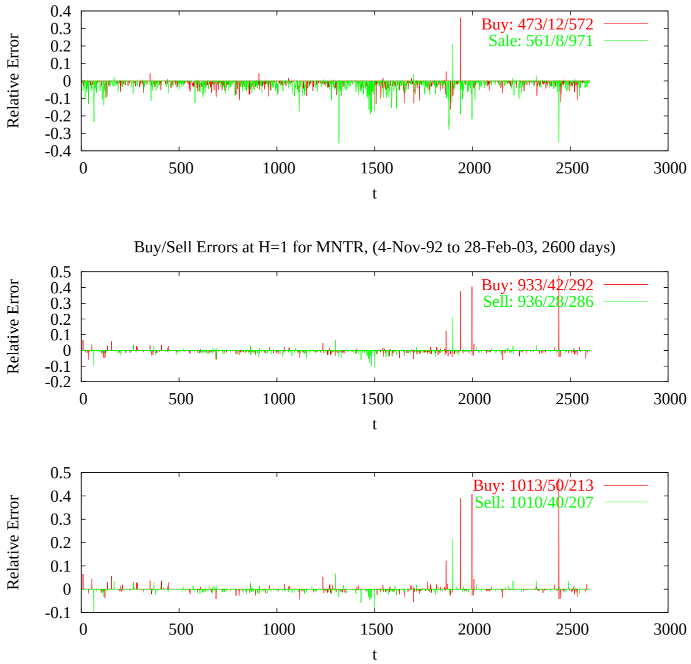


Figure 6.21: Prediction errors w.r.t. potential buy and sell signals for Mentor stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

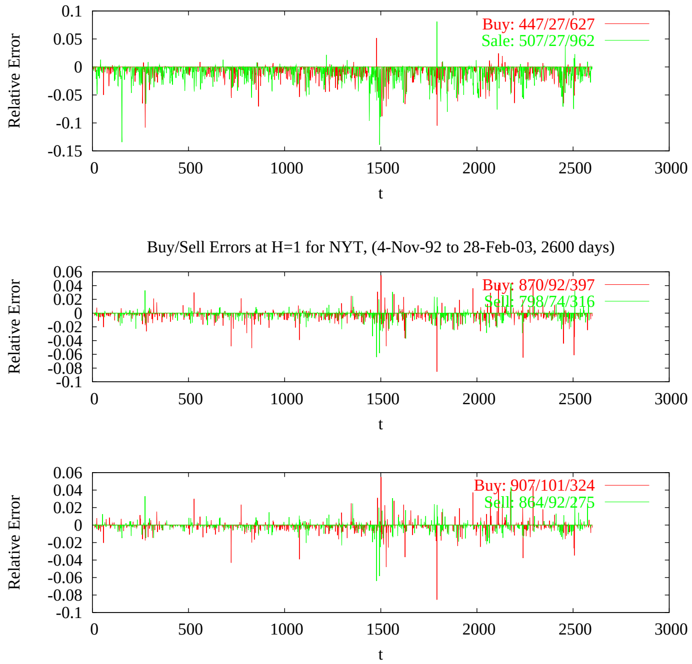


Figure 6.22: Prediction errors w.r.t. potential buy and sell signals for New York Times stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

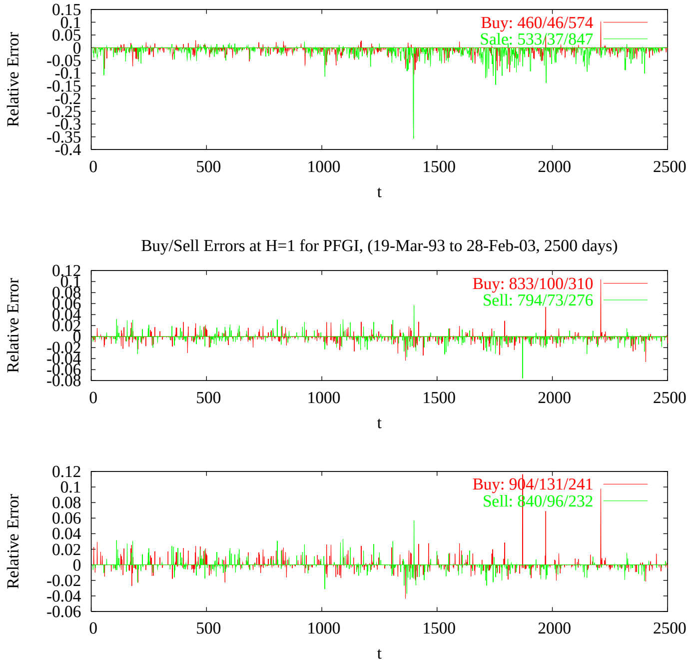


Figure 6.23: Prediction errors w.r.t. potential buy and sell signals for Provident Financial Group stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

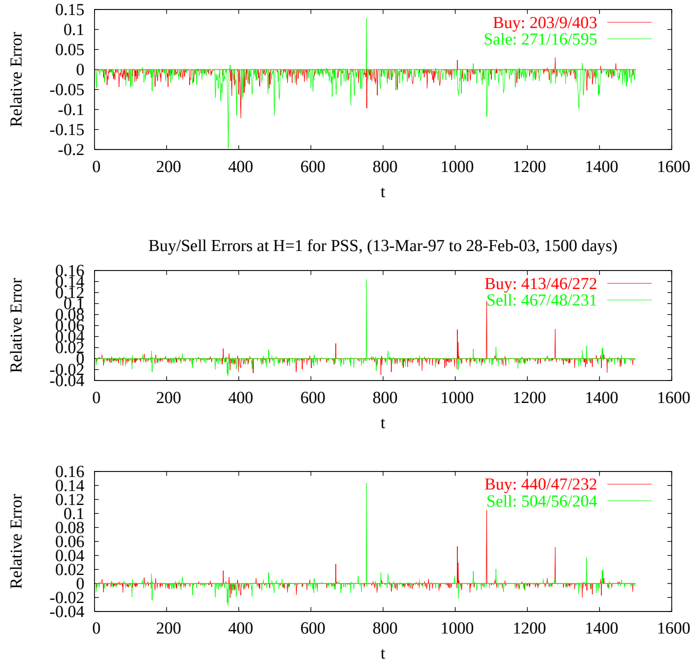


Figure 6.24: Prediction errors w.r.t. potential buy and sell signals for Payless ShoeSource stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

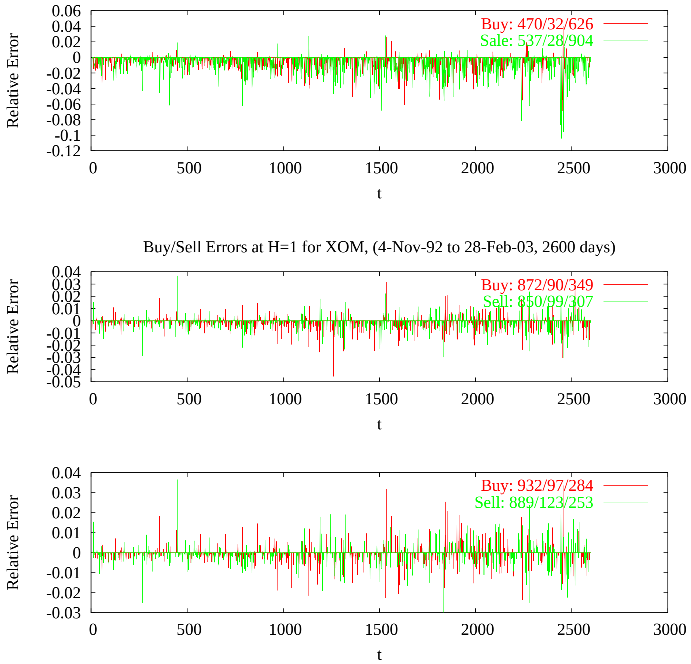


Figure 6.25: Prediction errors w.r.t. potential buy and sell signals for Exxon-Mobil stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

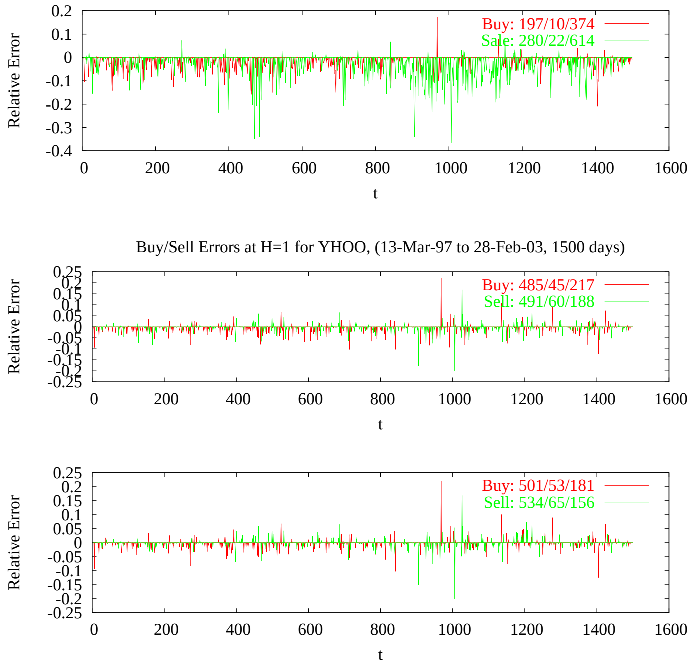


Figure 6.26: Prediction errors w.r.t. potential buy and sell signals for Yahoo stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

Table 6.4: Prediction performance on the ten stock-price time series benchmarks for different predictors described in Section 6.1.1.

AMR Inc (AMR)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.320	(0.429,0.033)	(0.250,-0.020)	0.320	(0.022,0.032)	(0.657,-0.028)
CNN-LP	0.594	(0.200,0.013)	(0.206,-0.009)	0.593	(0.079,0.022)	(0.329,-0.010)
CNN-CSP	0.629	(0.182,0.012)	(0.189,-0.008)	0.628	(0.091,0.020)	(0.282,-0.008)

Citigroup (C)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.324	(0.385,0.023)	(0.292,-0.019)	0.324	(0.034,0.010)	(0.643,-0.022)
CNN-LP	0.598	(0.178,0.008)	(0.224,-0.009)	0.600	(0.090,0.013)	(0.310,-0.009)
CNN-CSP	0.632	(0.161,0.007)	(0.206,-0.008)	0.634	(0.105,0.013)	(0.260,-0.008)

General Electric (GE)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.356	(0.363,0.018)	(0.281,-0.013)	0.356	(0.024,0.009)	(0.620,-0.016)
CNN-LP	0.654	(0.152,0.007)	(0.194,-0.007)	0.650	(0.061,0.009)	(0.289,-0.007)
CNN-CSP	0.693	(0.129,0.006)	(0.178,-0.005)	0.691	(0.075,0.009)	(0.234,-0.006)

International Business Machine (IBM)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.335	(0.382,0.022)	(0.283,-0.017)	0.335	(0.028,0.016)	(0.638,-0.020)
CNN-LP	0.635	(0.165,0.009)	(0.200,-0.009)	0.634	(0.064,0.013)	(0.301,-0.009)
CNN-CSP	0.671	(0.149,0.008)	(0.180,-0.008)	0.670	(0.080,0.012)	(0.249,-0.008)

continued on next page

continued from previous page

Mentor (MNTR)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.398	(0.379,0.032)	(0.223,-0.022)	0.398	(0.008,0.046)	(0.594,-0.029)
CNN-LP	0.747	(0.128,0.013)	(0.125,-0.011)	0.743	(0.028,0.039)	(0.230,-0.011)
CNN-CSP	0.801	(0.100,0.011)	(0.099,-0.008)	0.799	(0.036,0.035)	(0.166,-0.008)

New York Times (NYT)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.367	(0.381,0.017)	(0.252,-0.015)	0.367	(0.021,0.010)	(0.612,-0.016)
CNN-LP	0.658	(0.158,0.007)	(0.184,-0.007)	0.655	(0.065,0.009)	(0.280,-0.007)
CNN-CSP	0.692	(0.146,0.006)	(0.162,-0.006)	0.691	(0.075,0.009)	(0.234,-0.006)

Provident Financial Group (PFGI)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.398	(0.358,0.022)	(0.245,-0.015)	0.398	(0.033,0.012)	(0.569,-0.020)
CNN-LP	0.689	(0.154,0.008)	(0.157,-0.007)	0.682	(0.073,0.013)	(0.246,-0.008)
CNN-CSP	0.715	(0.147,0.007)	(0.138,-0.006)	0.714	(0.093,0.013)	(0.194,-0.006)

Payless ShoeSource (PSS)

Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.317	(0.403,0.021)	(0.280,-0.015)	0.317	(0.017,0.011)	(0.667,-0.019)
CNN-LP	0.599	(0.185,0.006)	(0.216,-0.006)	0.596	(0.064,0.009)	(0.341,-0.006)
CNN-CSP	0.638	(0.168,0.006)	(0.194,-0.005)	0.637	(0.070,0.008)	(0.294,-0.005)

continued on next page

continued from previous page						
Exxon-Mobil Stock (XOM)						
Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.388	(0.360,0.014)	(0.252,-0.010)	0.388	(0.023,0.008)	(0.589,-0.013)
CNN-LP	0.673	(0.154,0.005)	(0.173,-0.005)	0.671	(0.074,0.007)	(0.256,-0.006)
CNN-CSP	0.708	(0.136,0.005)	(0.157,-0.004)	0.706	(0.085,0.007)	(0.208,-0.005)
Yahoo! (YHOO)						
Predictor	Errors w.r.t. low/high price range			Errors w.r.t. potential buy/sell		
	Zero	Positive	Negative	Zero	Positive	Negative
AR	0.319	(0.417,0.056)	(0.264,-0.038)	0.319	(0.021,0.034)	(0.660,-0.050)
CNN-LP	0.658	(0.156,0.020)	(0.186,-0.022)	0.657	(0.071,0.028)	(0.272,-0.023)
CNN-CSP	0.696	(0.139,0.017)	(0.165,-0.018)	0.695	(0.079,0.029)	(0.226,-0.019)

Figures 6.27 to 6.36 illustrate the predicted values along with daily low/high prices. A close look would show that the predictions from AR go out of daily low/high price boundaries more frequently than CNN-LP and CNN-CSP. Note that it may be hard to tell from these graphs the difference between different predictors, as they does not show the details as well as the errors w.r.t. price ranges and potential buy/sell signals. To take a better look at the predicted values w.r.t. daily low/high prices, we plot the last 200 predictions generated by the three predictors for C, IBM, and XOM stocks in Figures 6.37 to 6.39. Clearly, we can see that that CNN-LP and CNN-CSP perform considerably better than AR in this measure.

We have also built a portfolio for each stock with an initial value of $1 million according to the simple trading strategy described in Section 6.1.2. Each portfolio contains

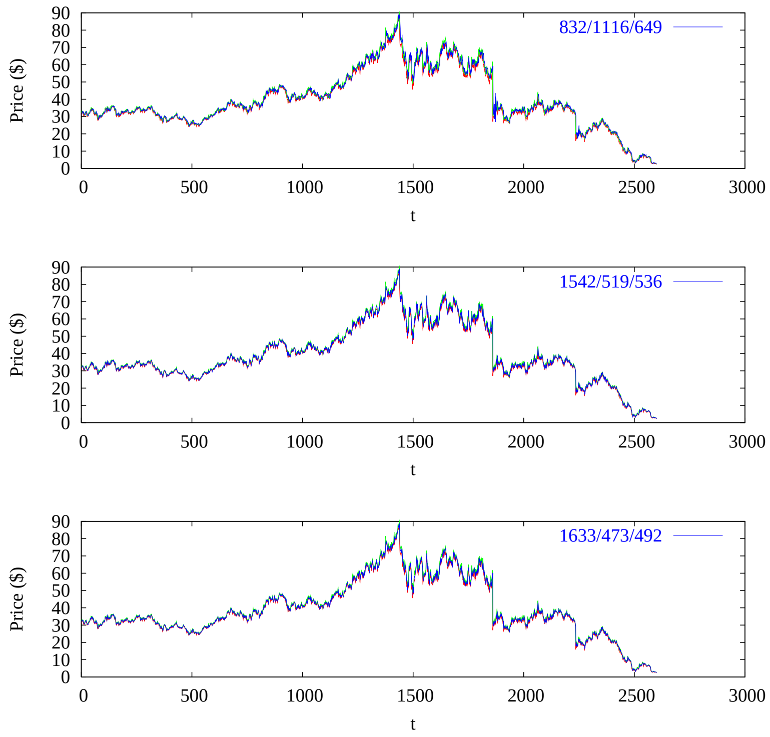


Figure 6.27: Actual predictions for AMR Inc stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

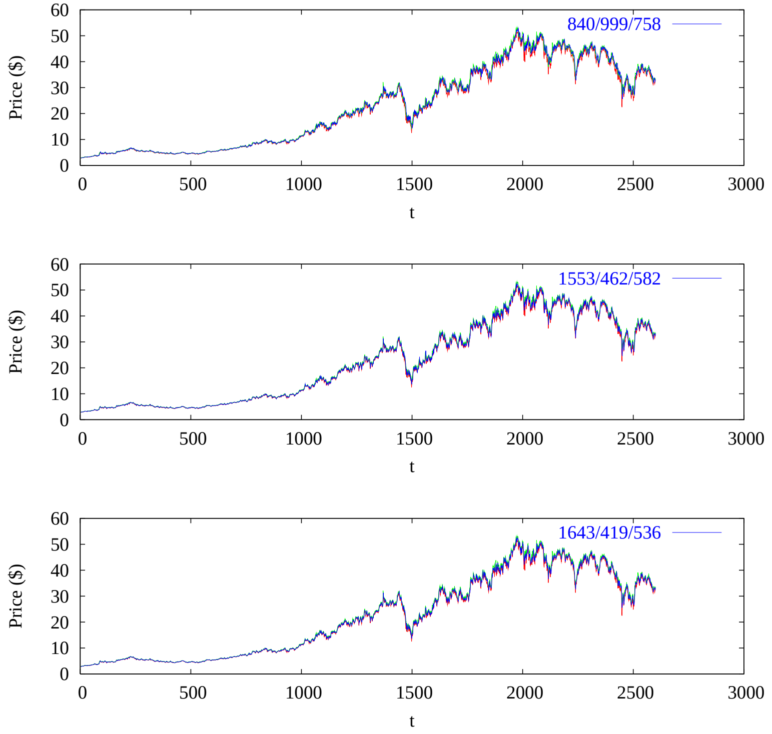


Figure 6.28: Actual predictions for Citigroup stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

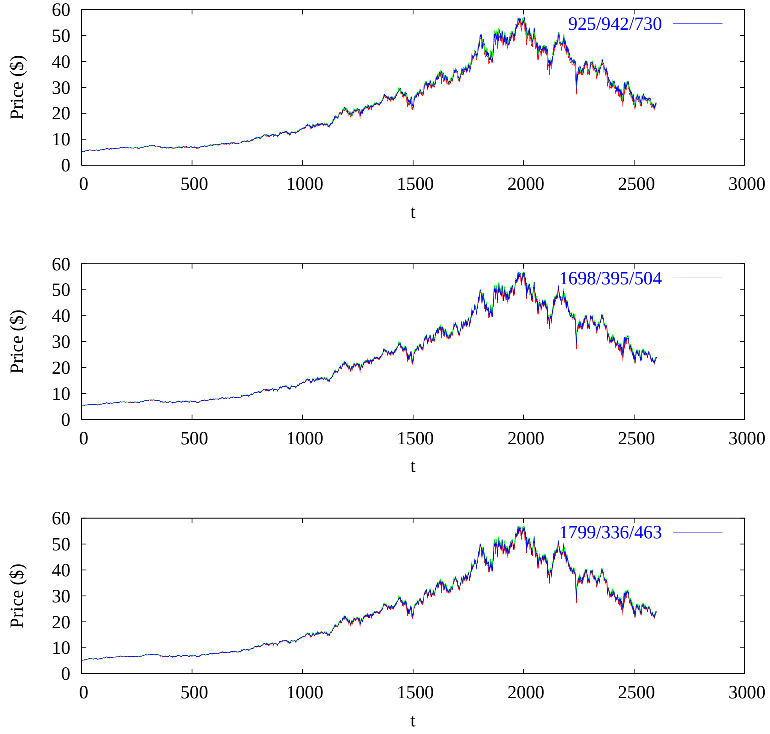


Figure 6.29: Actual predictions for General Electric stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

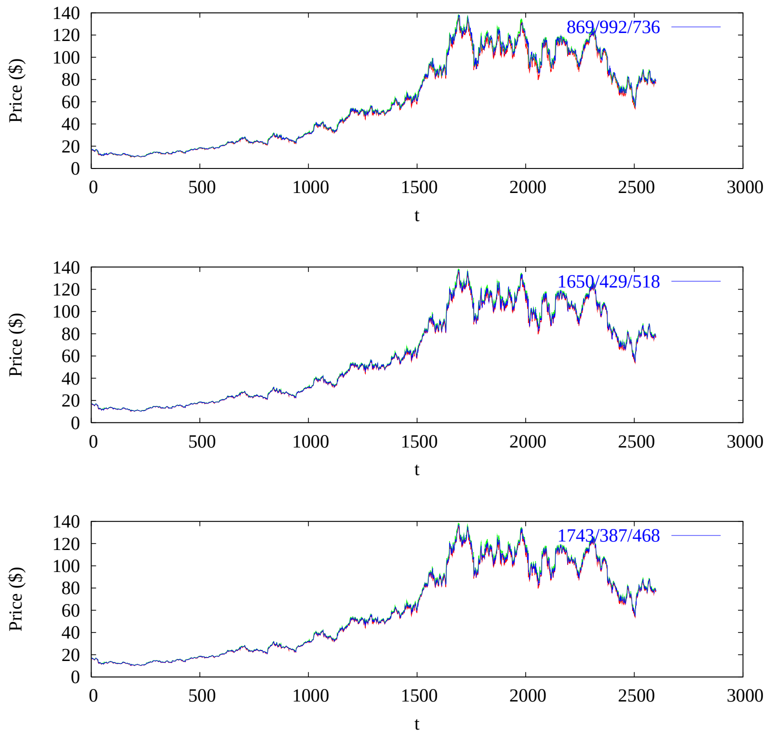


Figure 6.30: Actual predictions for IBM stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

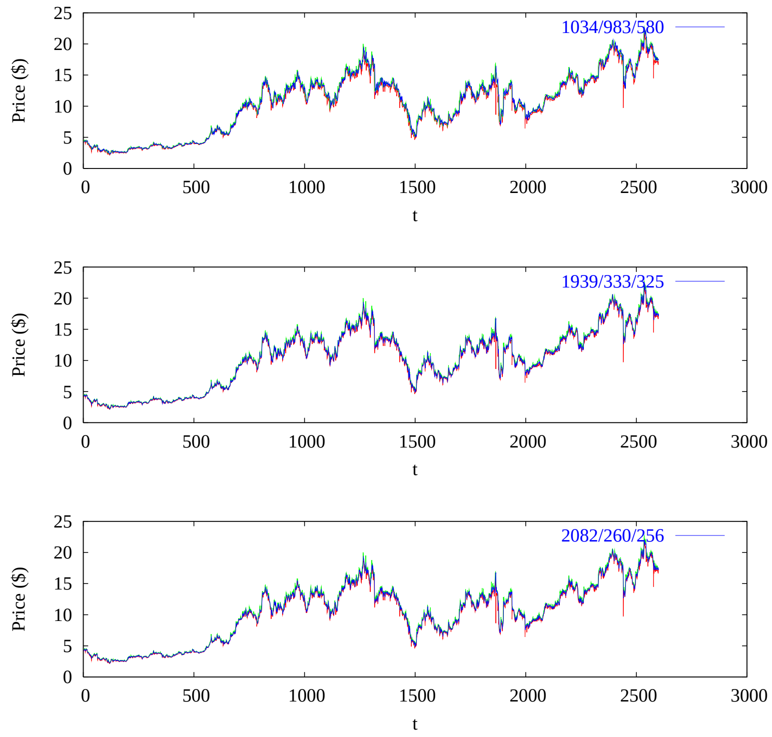


Figure 6.31: Actual predictions for Mentor stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

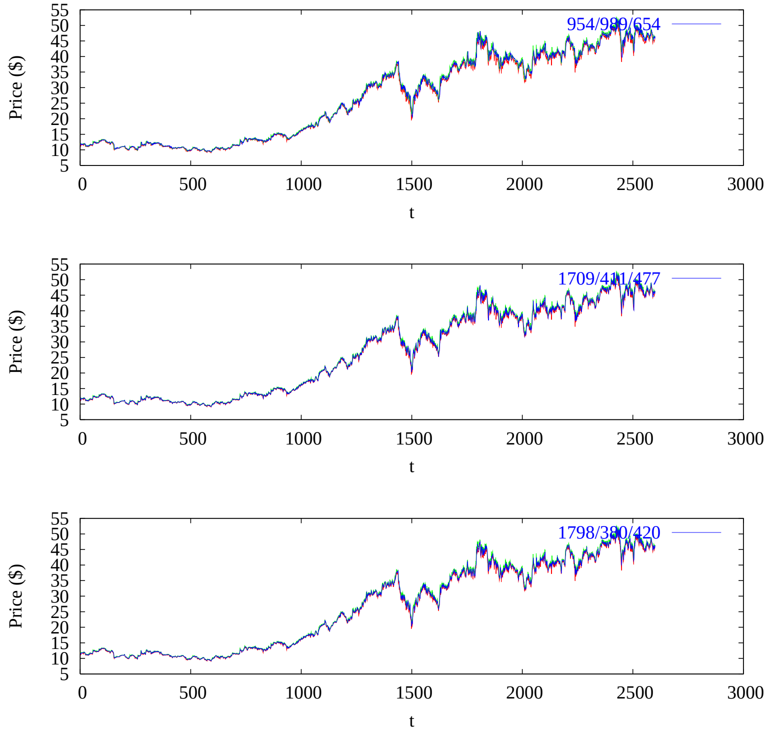


Figure 6.32: Actual predictions for New York Times stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

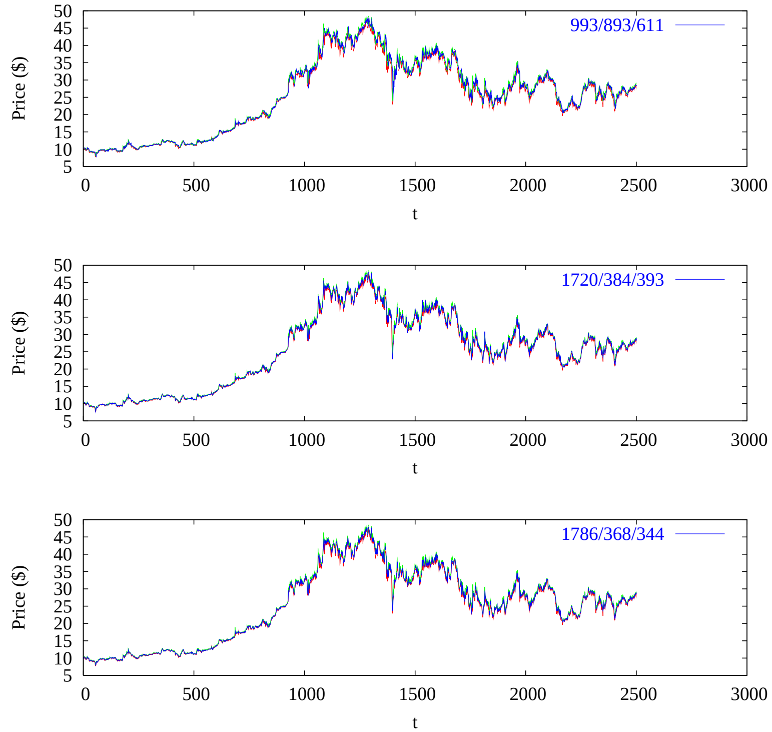


Figure 6.33: Actual predictions for Provident Financial Group stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

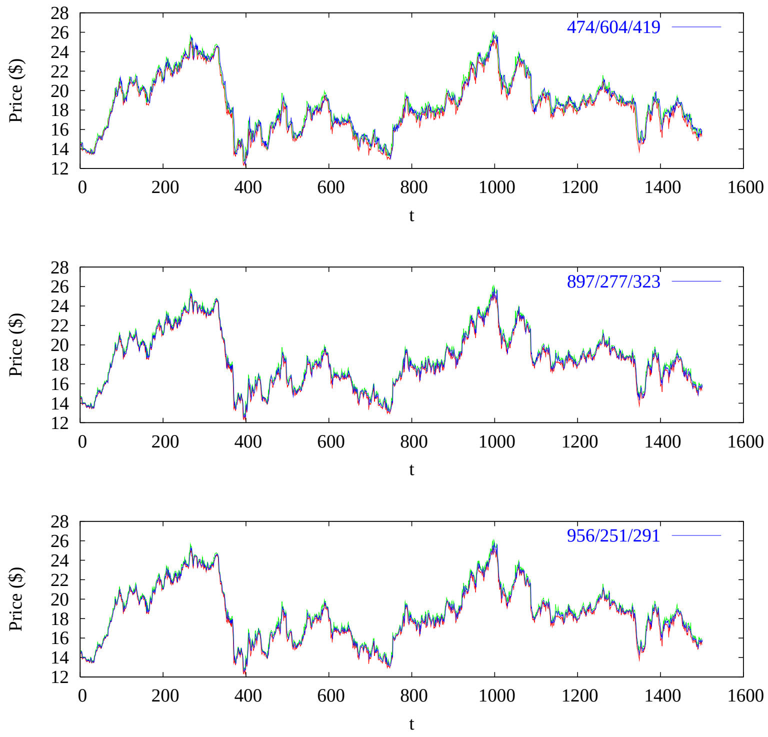


Figure 6.34: Actual predictions for Payless ShoeSource stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

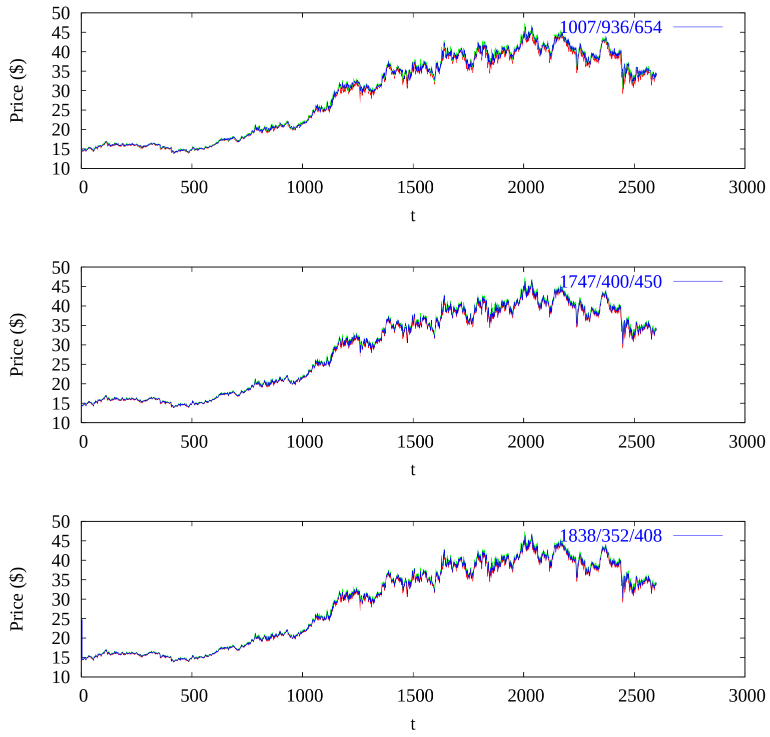


Figure 6.35: Actual predictions for Exxon-Mobil stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

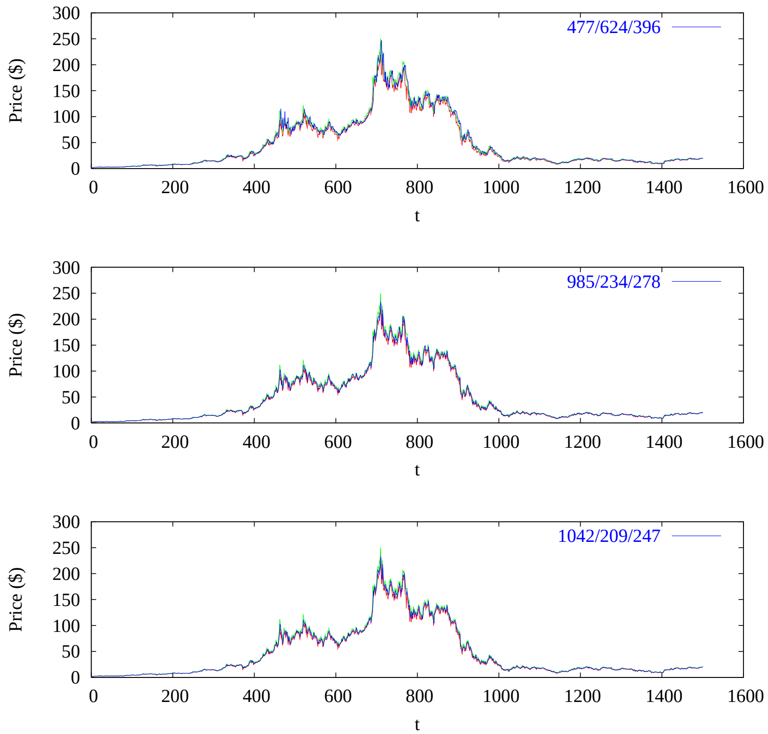


Figure 6.36: Actual predictions for Yahoo stock using Predictor AR (upper panel), Predictor CNN-LP (middle panel), and Predictor CNN-CSP (lower panel).

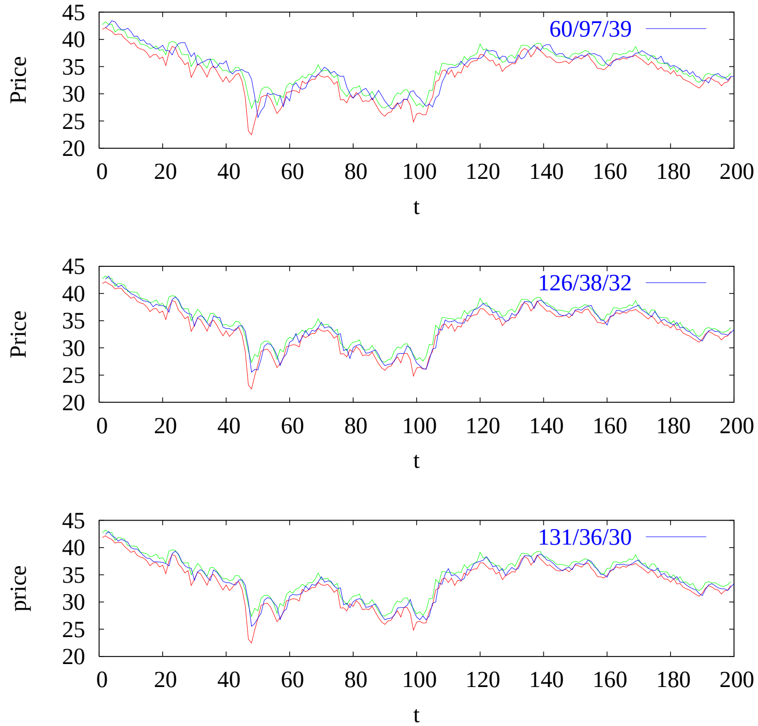


Figure 6.37: Last 200 predictions of Figure 6.35.

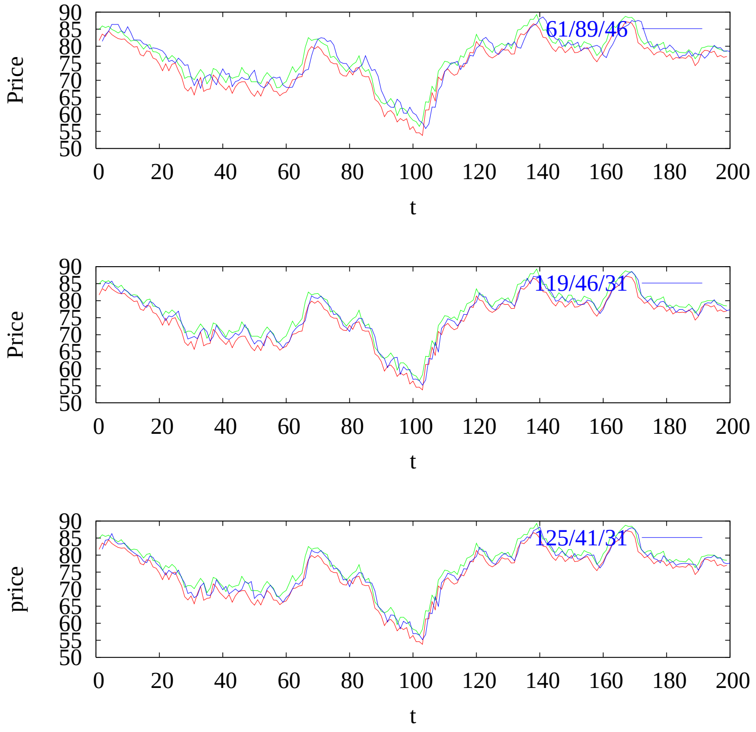


Figure 6.38: Last 200 predictions of Figure 6.35.

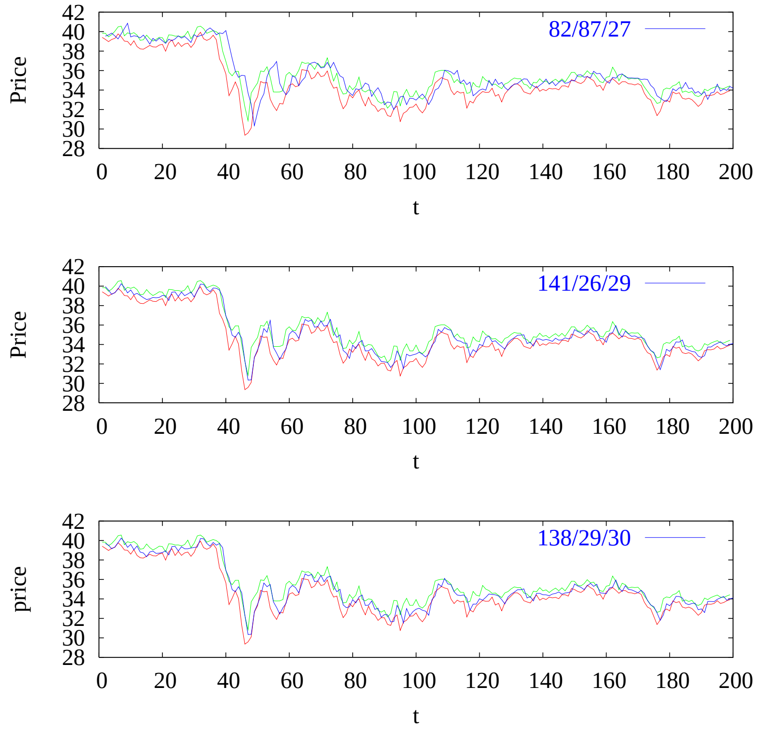


Figure 6.39: Last 200 predictions of Figure 6.35.

only one stock. We don't combine the prediction results on different stocks to form a multi-stock portfolio because we want to test the performance of the three predictors.

Figures 6.40 to 6.49 plot the evolution of the portfolio values with time when using the simple trading strategy described in Section 6.1.2 and the prediction results from the three predictors. We compute annual return for each portfolio and present the results in Table 6.5. The table shows one interesting fact, which also confirms what was found in [88, 89, 90], that a lot of predictors (Such as AR and CNN-LP) do not outperform the simple Buy-and-Hold strategy or random work in the long term. In a Buy-and-Hold strategy, a purchase is made at the beginning of the test period and then hold the position during the whole test period without making any transaction.

It is interesting that the portfolio performance of Predictor AR is not too much better than Buy-and-Hold. Five out of ten portfolios based on AR are better than Buy-and-Hold, whereas another five perform worse. There is also little difference between AR and Buy-and-Hold in terms of their average annual return over ten portfolios. For the strategy based on CNN-LP, it beats Buy-and-Hold in only 3 out of 10 portfolios. On the other hand, the strategy based on CNN-LP has higher average annual returns than Buy-and-Hold. It significantly underperforms Buy-and-Hold in only three portfolios (C, PSS, and YHOO) but performs much better than Buy-and-Hold in the portfolios for MNTR and PFGI. When compared with AR, CNN-LP performs better in 6 out of 10 portfolios and has higher average annual returns. Based on above observations, we conclude that AR and CNN-LP do not perform better than Buy-and-Hold.

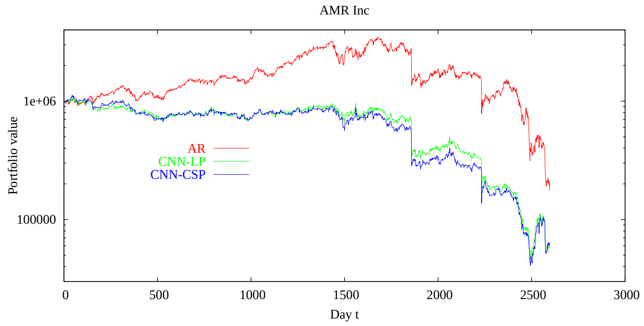


Figure 6.40: Portfolio value grows with time for trading AMR Inc stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$31.560 and \$2.400, respectively.

On the other hand, CNN-CSP is almost consistently better than CNN0-LP (in 8 of 10 portfolios) and leads to significantly higher average annual returns than Buy-and-Hold, AR and CNN-LP, although it performs considerably worse than Buy-and-Hold in the portfolio of PSS. Since CNN-CSP is almost consistently better than CNN-LP and their only difference is the preprocessing, we conclude that the channel-specific preprocessing in CNN-CSP is the key to its improved portfolio performance. Further, CNN-CSP outperforms both AR and Buy-and-Hold in 6 out of 10 portfolios. When CNN-CSP underperforms, it normally only underperforms slightly; and when it outperforms, it usually outperforms significantly. The overall average annual return over 10 portfolios for CNN-CSP is significantly better than any of the other three strategies.

Next, we consider why some portfolios fail to perform during some time periods. For example, the daily closing price for AMR increases from \$51.50 on March 7, 2000 to \$60.63

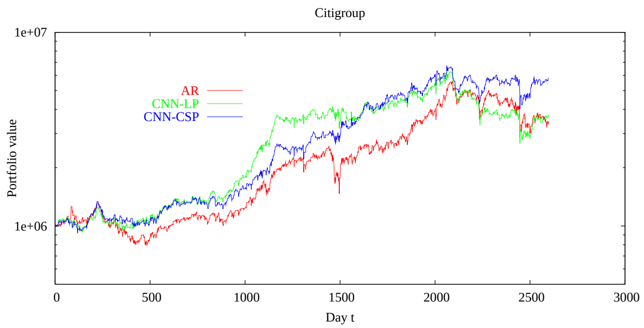


Figure 6.41: Portfolio value grows with time for trading Citigroup stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$2.880 and \$33.390, respectively.

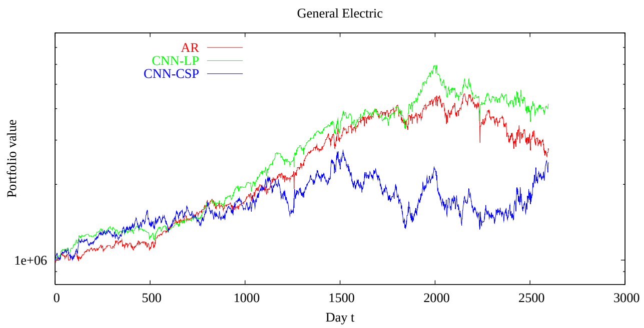


Figure 6.42: Portfolio value grows with time for trading General Electric stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$5.285 and \$23.965, respecitively.

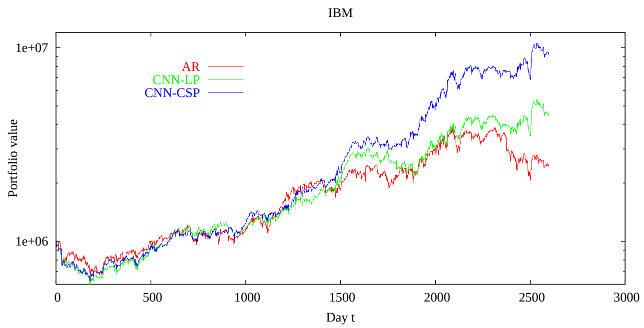


Figure 6.43: Portfolio value grows with time for trading IBM stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$16.895 and \$77.735, respectively.

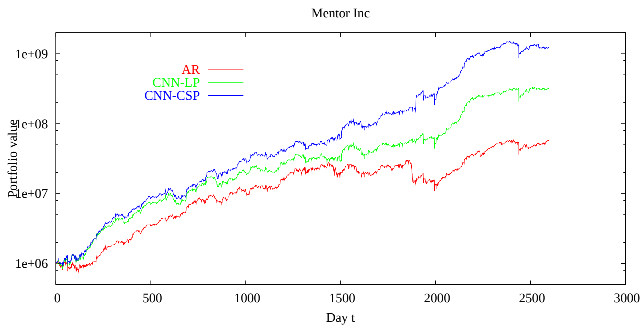


Figure 6.44: Portfolio value grows with time for trading Mentor stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$4.200 and \$17.375, respectively.

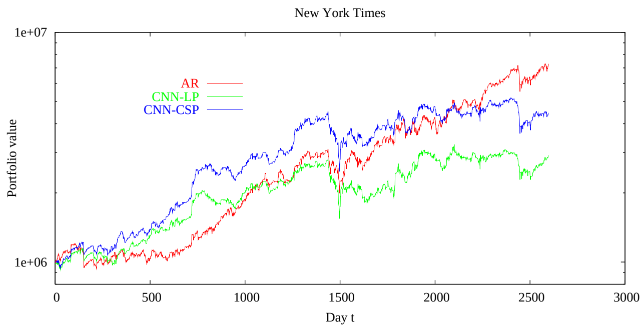


Figure 6.45: Portfolio value grows with time for trading New York Times stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$12.005 and \$46.230, respectively.

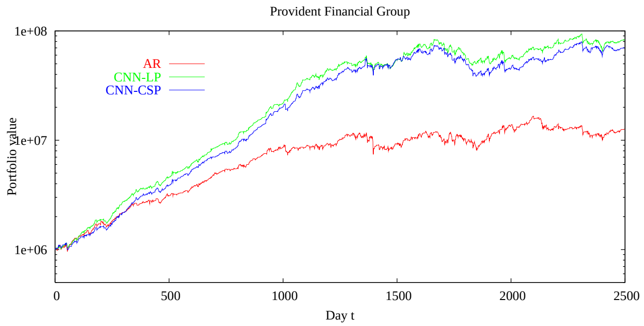


Figure 6.46: Portfolio value grows with time for trading Provident Financial Group stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending day days of the investment duration are \$10.245 and \$28.925, respectively.

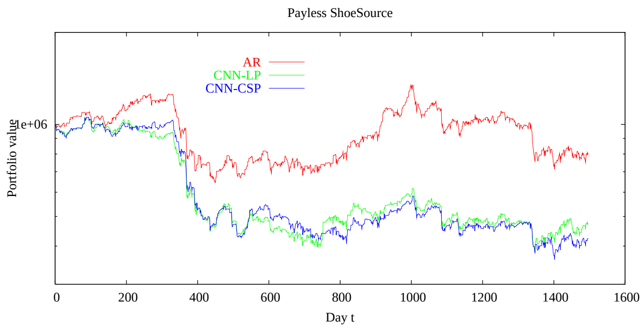


Figure 6.47: Portfolio value grows with time for trading Payless ShoeSource stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$14.540 and \$15.635, respectively.

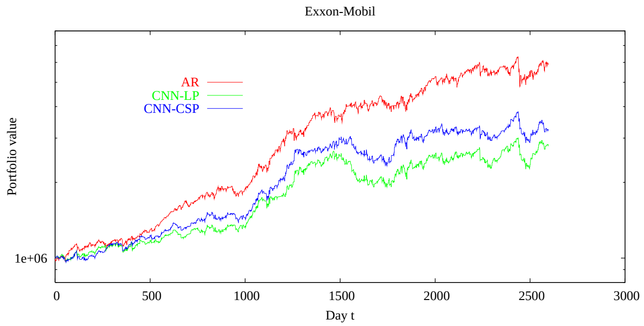


Figure 6.48: Portfolio value grows with time for trading Exxon-Mobil stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$14.705 and \$33.175, respectively.

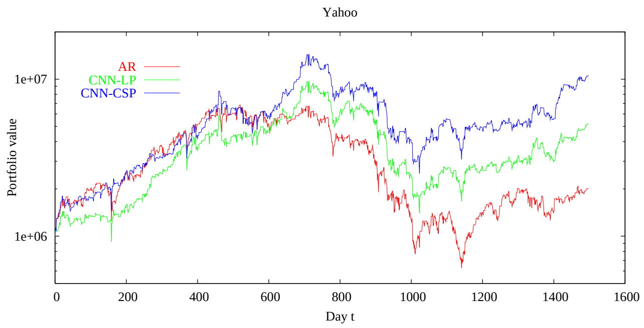


Figure 6.49: Portfolio value grows with time for trading Yahoo stock based on Predictors AR, CNN-LP, and CNN-CSP. The stock prices at the beginning and the ending days of the investment duration are \$2.015 and \$20.465, respectively.

on March 15, 2000. All three predictors predicted that the stock price would continue its uptrend. However, on March 16, 2000, the closing price dropped to \$28.00. All portfolios suffered more than 50% loss in that single day due to an unexpected news release. Such price changes were never seen in training and are totally unpredictable based on the historical information. The September 11 event caused aother huge unpredictable price drops. The YHOO stock experienced the typical technology stock bubble from March 2000 to the middle of 2002. All three long-only (without short position) portfolios based on the three predictors experience continuous loss. For PSS, the stock price dropped continuously from July 1998 to Oct 1998. As a result, all three long-only portfolios can only lose money. Also for PSS, before July 1 2002, the stock price was on an up-trend, but between July 1 2002 and July 23 2002, its price dropped over 25%. The predictors had trouble in predicting such big trend switches and ended losing a lot of money.

Table 6.5: Annual return of portfolios based on three predictors (AR, CNN-LP, and CNN-CSP) and Buy-and-Hold strategy for the ten different stocks. All returns are based on portfolio values without leverage. The highest annual return achieved for each stock is highlighted.

Stock	Buy-and-Hold	AR	CNN-LP	CNN-CSP
AMR	−22.07%	<u>−15.53%</u>	−23.77%	−23.45%
C	<u>26.77%</u>	12.60%	13.71%	18.61%
GE	<u>15.76%</u>	10.27%	14.88%	15.13%
IBM	15.92%	9.11%	15.68%	<u>24.16%</u>
MNTR	14.74%	47.87%	74.69%	<u>99.05%</u>
NYT	13.94%	<u>21.12%</u>	10.92%	15.60%
PFGI	11.02%	29.06%	<u>55.92%</u>	53.16%
PSS	<u>1.23%</u>	−3.62%	−11.83%	−13.45%
XOM	8.19%	<u>18.87%</u>	10.61%	12.10%
YHOO	47.54%	12.54%	31.92%	<u>48.61%</u>
Average	13.30%	14.23%	19.27%	<u>24.95%</u>

During the time the prediction quality is low and the stock is experiencing big fluctuations, the naive trading strategy used here simply disregard the risk. On the other hand, the only input to the ANN model is the historical stock prices. In reality, stock prices are greatly influenced by recent trading volumes, the macro economic factors (such as interest rate, treasure bill rate, crude oil price, unemployment rate, consumer sentiments, *etc*), industry/sector performance, change in analyst's estimation (downgrade and upgrade), scandals and the occurrence of big events (such as the Iraq war and the September 11 terrorist attack), and many other factors [55]. Historical stock prices alone cannot determine its future price in many cases, especially large jumps and drops in prices. Inevitably, multiple factors should be used to feed and train ANN models. The only purpose for adopting the simple naive trading strategy here is to figure out whether CNN-CSP is better than AR and CNN-LP. We believe that much better returns can be achieved by designing a more comprehensive strategy.

We include here in Table 6.6 annual return by year for CNN-CSP so that one may have an idea about year-to-year performance. The annual returns vary from one year to next year significantly. This is the typical when there is no risk control is applied. Also, we can see that Years 2000 and 2001 are the most difficult years for the long-only portfolios based on our simple naive trading strategy and CNN-CSP. The global economic environment in those years are very tough for long-only portfolio managers because of the burst of technology bubble and events such as the September 11 happened.

6.4 Summary and Conclusions

In this chapter, we have presented the setup of experiments, including the predictors compared and the performance measures used. We then examin predictions on each channel when channel-specific preprocessing is used. We test the performance of different parameters in channel-specific preprocessing and conclude that performance varies little with different combinations of mother filters and low-pass filters for individual channels as long as those filters are not far off the target. We then select a single set of parameter to be used for channel-specific preprocessing, along with the target low-pass filter for traditional preprocessing. Last, we conduct experiments on ten stock benchmarks using three predictors: autoregressive model, ANN model with traditional low-pass preprocessing, and ANN model with channel-specific preprocessing.

Table 6.6: Annual return by year for portfolios based on CNN-CSP. The numbers in the table are percentage numbers.

Year	AMR	C	GE	IBM	MNTR	NYT	PFGI	PSS	XOM	YHOO
1993	-9.99	-0.10	19.83	4.32	288.02	11.39			13.43	
1994	-25.76	5.11	12.15	15.66	143.51	27.40	127.45		7.64	
1995	15.62	17.59	7.14	15.53	86.46	83.21	96.49		19.84	
1996	2.38	34.73	9.96	28.80	114.51	10.93	139.47		12.58	
1997	-3.83	44.95	5.11	31.50	41.71	38.20	114.85		57.48	
1998	-10.89	25.86	23.16	69.39	98.96	-6.00	34.68	-49.59	17.07	138.05
1999	-13.39	49.25	-17.84	3.03	36.61	21.80	42.36	-8.00	-2.64	118.44
2000	-41.39	27.25	-7.44	85.67	162.32	-6.57	-22.94	14.71	7.55	-70.24
2001	-56.30	-5.46	-10.79	35.60	201.42	14.06	18.15	-11.53	-4.74	23.03
2002	-43.59	-8.83	41.93	20.56	1.85	-10.84	4.16	-12.66	13.12	70.57

The experiments show that our proposed channel-specific preprocessing for noisy stock-price time series is superior to traditional low-pass filtering. By decomposing the signals into different channels, one can preprocess individual channels differently by taking advantage of their individual properties. We exploit the properties that signals in the low-frequency channel have much larger magnitude than two other channels, and that the low-frequency channel is almost noise-free and does not require further denoising. This avoids introducing any lags in the low-frequency channel. This is the main advantage of our proposed preprocessing approach over the traditional low-pass filtering approach.

A comparison of our ANN model with channel-specific preprocessing and the traditional autoregressive and buy-and-hold models leads us to consider that our approach has better forecasting power and outperforms the AR and buy-and-hold models significantly. The results are consistent across a variety of stocks over a period of ten years.

We also believe that, in the future, multiple factors should be considered in training ANN in order to predict stock prices. These factors may include the macro economic environment, global events, analysts' oppinions, stock volatility, cash flow information, and price momentum.

Chapter 7

Conclusions And Future Work

In this thesis, we have studied the characteristics of time series, including linearity/nonlinearity, seasonality, stationarity/nonstationarity, and noise. We have reviewed existing approaches for handling these characteristics, and have pointed out problems in existing work. We then focus on neural-network models for general nonlinear chaotic time-series predictions. We survey existing neural-network models for chaotic time-series predictions, and identify the common problems in existing neural-network models that use an single unconstrained objective function (such as mean square errors) as the only goal in learning.

We have also studied noisy time series predictions (with high-frequency noise only) using stock market time series as examples. We have studied systematically the trade-offs between denosing and lags incurred due to denoising. In order to remove unpredictable noisy components in a time series while introducing lags only on the portion of the time series with low magnitude, we have developed an innovative preprocessing method by combining wavelet decomposition and channel-specific preprocessing. We perform pattern-wise transformations to handle nonstationarity in the low-frequency channel, and perform low-pass filtering for the high-frequency channels according to the level of noise in individual high-frequency channels.

To solve the problems identified in existing neural-network prediction models, we propose a constrained formulation for ANN learning that allow multiple learning criteria and prior knowledge to be incorporated. Such criteria include testing errors on multiple validation sets that can model regime changes in piecewise chaotic time series, and the validation error of the ANN learned on the objective used in testing. We have further proposed a new RFIR architecture that combines a recurrent structure and a memory-based FIR structure, and a violation-guided back-propagation algorithm based on theory of extended saddle point condition for continuous nonlinear constrained optimization.

Based on a constrained formulation of the ANN model and training by the violation-guided back-propagation algorithm, we have conducted experiments on a broad range of chaotic time-series benchmarks. Our experimental results show that our proposed ANN model along with the training algorithm significantly improves over existing work.

We then extend our work to predict noisy time series using stock-price time series as benchmarks. First, we have developed a channel-specific preprocessing approach that is superior to the traditional low-pass preprocessing and the traditional autoregressive models.

There are two main aspects that one may continue on this work. First, we can further study the extension of prediction horizons (such as a week or even longer) for stock price time series based on weekly data. Second, for stock-price time series, we can develop more sophisticated trading strategies and feed real-time stock prices to a prediction module developed based on our neural-network predictor.

Bibliography

[1] H. Abarbanel. *Analysis of Observed Chaotic Data*. Springer, NY, 1995.

[2] H. Abarbanel, R. Brown, J. Sidorowich, and L. Tsimring. The analysis of observed chaotic data in physical systems. *Review of Modern Physics*, 65(4):1331–1392, 1993.

[3] H. Abarbanel, T. Frison, and L. Tsimring. Obtaining order in a world of chaos. *IEEE Signal Processing Magazine*, 15(3):49–66, May 1996.

[4] B. Abraham and J. Ledolter. *Statistical Methods for Forecasting*. Wiley, New York, US, 1983.

[5] S. N. Amari, K.-R. Muller, M. Finke, and H. Yang. Statistical theory of overtraining – is cross-validation asymptotically effective? *Advances in Neural Information Processing Systems*, 8:176–182, 1996.

[6] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Trans. on Image Processing*, 1(2):205–220, 1992.

[7] M. Aoki. *State Space Modeling of Time Series*. Springer-Verlag, Nerlin, 1987.

[8] A. Aussem. Dynamical recurrent neural networks towards prediction and modeling of dynamical systems. *Neurocomputing*, 28:207–232, 1999.

[9] A. Aussem. Personal communications, March 2001.

[10] A. Aussem and F. Murtagh. Combining neural network forecasts on wavelet-transformed time series. *Connection Science*, 9:113–121, 1997.

[11] A. Aussem, F. Murtagh, and M. Sarazin. Dynamical recurrent neural networks - towards environmental time series prediction. *Int'l J. of Neural Systems*, 6:145–170, June 1995.

[12] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Prentice-Hall, 1976.

[13] T. Bollerslev, Y. Chou, and K. F. Kroner. ARCH models in finance: A review of the theory and evidence. *J. of Econometrics*, 52:5–59, 1992.

[14] Erik M. Bollt. Model selection, confidence and scaling in predicting chaotic time-series. *International Journal of Bifurcation and Chaos*, 10(6):1407–1422, 2000.

[15] G. E. P. Box and G. M. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, San Fransisco, CA, 1970.

[16] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control, 2nd ed.* Holden-Day, San Francisco, 1976.

[17] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting and Control*. Prentice-Hall, Englewood Cliffs, NJ, 3 edition, 1994.

[18] R. G. Brown. *Smoothing, Forecasting and Prediction*. Prentice Hall, Englewood Cliffs, NJ, 1963.

[19] C. Chatfield, A. B. Koehler, J. K. Ord, and R. D. Snyder. Models for exponential smoothing: A review of recent developments. *The Statistician*, 50, 2001.

[20] C. Chatfield and M. Yar. Holt-winters forecasting: Some practical issues. *The Statistician*, 37:129–140, 1988.

[21] Chris Chatfield. *The analysis of time series – an introduction*. Chapman & Hall, London, 5th edition, 1996.

[22] Chris Chatfield. *Time-series forecasting*. Chapman & Hall/CRC, Boca Raton, Florida, 2001.

[23] Y. X. Chen, C.-W. Hsu, and B. W. Wah. SGPlan: Subgoal partitioning and resolution in planning. In *Proc. Fourth Int'l Planning Competition*. Int'l Conf. on Automated Planning and Scheduling, June 5 2004.

[24] M. Clyde, G. Parmigiani, and B. Vidakovic. Multiple shrinkage and subset selection in wavelets. *Biometrika*, 85(2):391–401, 1998.

[25] R. R. Coifman and D. L. Donoho. Translation-invariant de-noising. Technical report, Department of Statistics, Stanford University, may 1995.

[26] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Trans. on Information Theory*, IT-13(1):21–27, January 1967.

[27] B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, New York, 1991.

[28] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, XLI:909–996, 1988.

[29] C. de Groot and D. Wurtz. Analysis of univariate time series with connectionist nets: A case study of two classical examples. *Neurocomputing*, 3:177–192, 1991.

[30] W. L. Ditto and L. M. Peroca. Mastering chaos. *Scientific American*, pages 62–68, 1993.

[31] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard. Wavelet shrinkage: Asymptopia. *J. Roy. Statist. Soc.*, B 57(2):301–369, 1995.

[32] D. Drossu and Z. Obradovic. Regime signaling techniques for non-stationary time series forecasting. In *Proc. 30th Hwaii Int'l Conf. on System Sciences*, volume 5, pages 530 –538, Wailea, HI, USA, 1997.

[33] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[34] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[35] O. Kardaun F. Murtagh, A. Aussem. The wavelet transform in multivariate data analysis. *COMPSTAT,Part A*, pages 397–402, 1996.

[36] J. Faraway and C. Chatfield. Time series forecasting with neural networks: A comparative study using the airline data. *Appl. Statist.*, 47:231–250, 1998.

[37] D. H. Fisher. Conceptual clustering, learning from examples, and inference. *Machine Learning*, pages 38–49, 1987.

[38] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[39] B. E. Flores and S. L. Pearce. The use of an expert system in the M3 competition. *Int'l J. of Forecasting*, 16:485–496, 2000.

[40] P. H. Franses and D. van Dijk. *Nonlinear time series models in empirical finance.* Cambridge University Press, Cambridge, UK, 2000.

[41] P. H. Franses and D. van Dijk. *The Forecasting Performance of Various Models for Seasonality and Nonlinearity for Quarterly Industrial Production.* Technical Report, Econometric Institute Report EI 2001-14, Erasmus University Rotterdam, Netherlands, April 2001.

[42] E. S. Gardner. Exponential smoothing: the state of the art. *J. Forecasting*, 2:1–21, 1985.

[43] E. S. Gardner and E. McKenzie. Forecasting trends in time series. *Man. Sci.*, 31:1237–1246, 1985.

[44] A. B. Geva. ScaleNet – multiscale neural-network architecture for time series prediction. *IEEE Trans. on Neural Networks*, 9(5):1471–1482, Sept. 1998.

[45] Z. Ghahramani and G.E. Hinton. Switching state-space models. Technical report, 6 King's College Road, Toronto M5S 3H5, Canada, 1998.

[46] C. L. Giles, S. Lawrence, and A. C. Tsoi. Rules inference for financial prediction using recurrent neural networks. In *Proc. of IEEE/IAFE conference on computational intelligence for financial engineering*, pages 253–259. IEEE, 1997.

[47] W. L. Goffe, G. D. Ferrier, and J. Rogers. Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, 60:65–99, 1994.

[48] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Pub. Co., 1989.

[49] C. W. J. Granger and A. P. Andersen. *Introduction to Bilinear Time Series Models.* Vandenhoeck & Ruprect, Göittingen, 1978.

[50] M. Grzebyk and H. Wackernagel. Linear models for spatial or temporal multivariate data. In *Proc.of the 6th Internation Meeting on Statistical Climatology*, pages 427–429, Galway, Ireland, June 1995.

[51] S. Gutjahr, M. Riedmiller, and J. Klingemann. Daily prediction of the foreign exchange rate between the us dollar and the german mark using neural networks. In *Proc. of SPICES*, pages 492–498, 1997.

[52] M. Hallas and G. Dorffner. A comparative study on feedforward and recurrent neural networks in time series prediction using gradient descent learning. In *Proc. of 14th European Meeting on Cybernetics and Systems Research*, volume 2, pages 644–647, 1998.

[53] A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter.* Cambridge University Press, Cambridge, 1989.

[54] A. C. Harvey. *Time Series Models.* Philip Allan, Hemel Hempstead, 1993.

[55] Robert Haugen. *New Finance: The Case Against Efficient Markets.* Prentice Hall, Englewood Cliffs, NJ, 1998.

[56] S. Haykin. *Blind Deconvolution.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[57] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, NJ, 2nd edition, 1999.

[58] R. Hegger and T. Schreiber. The TISEAN software package. *http://www.mpipks-dresden.mpg.de/ tisean*, 2002.

[59] T. Hellstrm. Predicting a rank measure for stock returns. *Theory of Stochastic Processes*, 6(20):64–83, 2000.

[60] M. L. Hilton, B. D. Jawerth, and A. Sengupta. Compression still and moving images with wavelets. *Multimedia Systems*, 2(2):218–227, 1994.

[61] B. G. Horne and C. L. Giles. An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Neural Information Processing Systems*, pages 697–704. MIT Press, Cambridge, MA, 1995.

[62] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward nets are universal approximators. *Neural Networks*, 2:359–366, 1989.

[63] J. M. Hutchinson. *A Radial Basis Function Approach to Financial Time Series Analysis*. Ph.D. Thesis, Massachusetts Institute of Technology, Feb 1994.

[64] G. Iyengar and A. Lippman. Entropy measures for controlled coding. In *Proc. of IS&T Symposium, Digital Video Compression*, San Jose, California, 1 1996.

[65] M. Jachan, G. Matz, and F. Hlawatsch. Time-frequency-autoregressive random processes: Modeling and fast parameter estimation. In *Proc. IEEE ICASSP*, volume VI, pages 125–128, Hong Kong, April 2003.

[66] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[67] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31:3:264–323, 1999.

[68] T. Joachims. Making large-scale svm learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press, 1999.

[69] M. I. Jordan, Z. Ghahramani, and L. K. Saul. Hidden Markov decision trees. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Proc. Conf. Advances in Neural Information Processing Systems, NIPS*, volume 9. MIT Press, 1996.

[70] B. H. Juang and L. R. Rabiner. Hidden Markov models for speech recognition. *Technometrics*, 33:251–272, 1991.

[71] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *J. of Artificial Intelligence Research*, 4:237–285, 1996.

[72] M. G. Kendall, A. Stuart, and J. K. Ord. *The Advanced Theory of Statistics*, volume 3. Griffin, London, 4 edition, 1983.

[73] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[74] C. Kluppelberg, R. Maller, M. Van De Vyver, and D. Wee. Testing for reduction to random walk in autoregressive conditional heteroscedasticity models. *The Econometrics Journal*, 5(2):387–416, 2002.

[75] J. Kohlmorgen, K.-R. Muller, and K. Pawelzik. Analysis of drifting dynamics with competing predictors. In *ICANN*, pages 785–790, 1996.

[76] J. Kohlmorgen, K.-R. Müller, and K. Pawelzik. Analysis of drifting dynamics with neural network hidden markov models. *Advances in Neural Information Processing Systems*, 10, 1998.

[77] T. Koskela, M. Lehtokangas, J. Saarinen, and K. Kaski. Time series prediction with multilayer perceptron, FIR and Elman neural networks. In *Proc. of the World Congress on Neural Networks*, pages 491–496, 1996.

[78] D. Kugiumtzis, B. Lillekjendlie, and N. Christophersen. Chaotic time series part I: Estimation of some invariant properties in state space. *Modeling, Identification and Control*, 15:205–224, 1994.

[79] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proc. Second Berkeley Sympos. Math. Stat. Prob.*, pages 481–492. University of California Press, 1951.

[80] K. J. Lang and G. E. Hinton. *The development of the time-delayed neural network architecture for speech recognition*. Technical Report #CMU-CS-88-152, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[81] S. Laur. Time series of deterministic dynamic systems: Celebrated takens theorem, November 2004.

[82] B. Lillekjendlie, D. Kugiumtzis, and N. Christophersen. Chaotic time series part ii: System identification and prediction. *Modeling, Identification and Control*, 15(4):225–243, 1994.

[83] M. L. Littman. Friend-or-foe q-learning in general-sum games. In *Proc.of ICML*. Morgan Kaufmann, 2001.

[84] Z.-Q. Lu and L. M. Berliner. Markov switching time series mdoels with application to a daily runoff series. *Water Resources Research*, 35:523–534, 1999.

[85] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1984.

[86] E. Maasoumi and J. Racine. Entropy and predictability of stock market returns. *Journal of Econometrics*, 107:291–312, 2002.

[87] M. Maggini, C.L. Giles, and B. Horne. Financial time series forecasting using k-nearest neighbors classification. In *Nonlinear Financial Forecasting: Proceedings of the First INFFC*, pages 169–181. Financial and Technology Publishing, Haymarket, VA, 1997.

[88] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler. The accuracy of extrapolation (time series) methods: results of a forecasting competition. *Int'l J. of Forecasting*, 1:111–153, 1982.

[89] S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, and L. F. Simmons. The M2-Competition: a real-time judgementally based forecasting study. *Int'l J. of Forecasting*, 9:5–23, 1993.

[90] S. Makridakis and M. Hibon. The M3-Competition: results, conclusions and implications. *Int'l J. of Forecasting*, 16:451–476, 2000.

[91] S. Makridakis and S. C. Wheelwright. *Forecasting Methods for Management*. Wiley, New York, 1989.

[92] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman. *Forecasting Methods and Applications*. Wiley, New York, 1998.

[93] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. on Pattern Anal. Machine Intell.*, 11:674–693, 1989.

[94] M. Maruyama, F. Girosi, and T. Poggio. A connection between GRBF and MLP. In *Artificial Intelligence Memo 1291*. Massachusetts Institute of Technology, 1991.

[95] T. Masters. *Signal and Image Processingwith Neural Networks*. John Wiley & Sons, Inc., NY, 1994.

[96] T. Masters. *Neural, Novel and Hybrid Algorithms for Time Series Prediction*. John Wiley & Sons, Inc., NY, 1995.

[97] Mathworks. *Financial Time Series Toolbox*. The Mathworks, Inc.

[98] Matlab. *Matlab User Manual*. The Math Works, Inc.

[99] Matlab. *Signal Processing Toolbox User's Guide*. The Math Works, 2001.

[100] N. Meade. A comparison of the accuracy of short term foreign exchange forecasting methods. *Int'l J. of Forecasting*, 18:67–83, 2002.

[101] R. J. Meinhold and N. D. Singpurwalla. Understanding the Kalman filter. *J. Amer. Statist. Assoc.*, 32:123–127, 1983.

[102] R. J. Meinhold and N. D. Singpurwalla. Robustification of Kalman filter models. *J. Amer. Statist. Assoc.*, 84:479–486, 1989.

[103] G. Melard and J.-M. Pasteels. Automatic arima modeling including interventions, using time series expert software. *Int'l J. of Forecasting*, 16:497–508, 2000.

[104] J. Mingers. An empirical comparison of selected measures for decision-tree induction. *Machine Learning*, 3(4):319–342, March 1989.

[105] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[106] A. Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, UK, November 1992.

[107] N. Morgan and H. Bourlard. Continuous speech recognition using multilayer perceptrons with hidden Markov models. In *IEEE ICASP*, volume 1, pages 413–416, Albuquerque, 1990.

[108] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *ICANN*, pages 999–1004, 1997.

[109] K.-R. Muller, J. Kohlmorgen, and K. Pawelzik. Analysis of switching dynamics with competing neural networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E78-A(10):1306–1315, 1995.

[110] F. Murtagh. Clustering massive data sets. In P.M. Pardalos J. Abello and M.G.C. Reisende, editors, *Handbook of Massive Data Sets*. Kluwer, 2000.

[111] F. Murtagh and A. Aussem. Using the wavelet transform for multivariate data analysis and time series forecasting. In C. Hayashi, H. H. Bock, K. Yajima, Y. Tanaka, N. Ohsumi, and Y. Baba, editors, *Data Science, Classification and Related Methods*, pages 617–624. Springer-Verlag, 1998.

[112] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5:595–603, April 1992.

[113] M. T. Musavi, K. B. Faris, K. H. Chan, and W. Ahmed. On the implementation of RBF technique in neural networks. In *Proc. of Conf. on Analysis of Neural Network Applications*, pages 110–115. ACM, 1991.

[114] G. P. Nason and B. W. Silverman. The stationary wavelet transform and some statistical applications. In *Lecture Notes in Statistics: Wavelets and Statistics*, pages 281–299. Springer-Verlag, New York, 1995.

[115] P. Newbold. Some recent developments in time-series analysis, iii. *Int. Statist. Rev.*, 56:17–29, 1988.

[116] D. F. Nicholls and A. R. Pagan. Varying coefficient regression. In E. J. Hannan, P. R. Krishnaiah, and M. M. Rao, editors, *Handbook of Statistics*, pages 413–449. North-Holland, Amsterdam, 1985.

[117] S. Nowlan and G. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4(4):473–493, 1992.

[118] T. Oates, L. Firoiu, and P. Cohen. Clustering time series with hidden markov models and dynamic time warping. In *Proceedings of the IJCAI-99 Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, pages 17–21, 1999.

[119] M. Palus. Identifying and quantifying chaos by using information-theoretic functionals. In A. S. Weigend and N. A. Gershenfeld, editors, *Time Series Predic-*

tion: Forecasting the Future and Understanding the Past, pages 387–413. Addison-Wesley, Reading, Mass., 1993.

[120] M. Palus, L. Pecen, and D. Pivaka. Estimating predictability: Redundancy and surrogate data method. *Neural Network World*, 4:537–552, 1995.

[121] F. J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[122] A. Pizurica. Image denoising using wavelets and spatial context modeling. Technical report, Belgium, 2002.

[123] A. Pizurica, W. Philips, I. Lemahieu, and M. Acheroy. A versatile wavelet domain nois filtration technique for medical imaging. *IEEE Trans. Medical Imaging*, 2003.

[124] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. of IEEE*, 78(9):1481–1497, 1990.

[125] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. In J. c. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Clarendon Press, Oxford, 1987.

[126] M. B. Priestley. *Spectral Analysis and Time Series*. Academic Press, London, 1981.

[127] M. B. Priestley. *Non-linear and non-stationary time series analysis.* Academic Press, London, 1988.

[128] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[129] L. R. Rabiner. A tutorial on hidden Markov models. *Proceedings of the IEEE*, 73:1349–1387, 1989.

[130] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *Accoustics, Speech and Signal Processing Magazine*, 3(1):4–16, 1986.

[131] T. S. Rao and M. M. Gabr. An introduction to bispectral analysis and bilinear models. In *Lecture Notes in Statistics: No. 24*. Springer, New York, 1984.

[132] B. D. Ripley. Neural networks and related methods for classification. *J. R. Statist. Soc. B*, 56:409–456, 1994.

[133] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, Cambridge, UK, 1996.

[134] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[135] E.W. Saad, D.V. Prokhorov, and D.C. Wunsch, II. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Trans. on Neural Networks*, 9:1456–1470, 11 1998.

[136] P. N. Sabes and M. I. Jordan. Reinforcement learning by probability matching. In S. David, M. M. Touretzky, and M. Perrone, editors, *Advances in Neural Information Proc.*, volume 8. MIT Press, Cambridge, 1995.

[137] W. Schiffmann, M. Joost, and R. Werner. Comparison of optimized backpropagation algorithms. In *Proc. of the European Symposium on Artificial Neural Networks, ESANN'93*, pages 97–104, April 1993.

[138] T. Schreiber. Interdisciplinary application of nonlinear time series methods. *Phys. Rep.*, 308(1):2–64, 1999.

[139] D. Servan-Schreiber, A. Cleermans, and J. L. McClelland. Learning sequential structure in simple recurrent networks. In D. Z. Anderson, editor, *Proc. Neural Information Processing Systems*, pages 643–652, New York, 1988. American Inst. of Physics.

[140] Y. Shang and B. W. Wah. Global optimization for neural network training. *IEEE Computer*, 29(3):45–54, March 1996.

[141] T. Shinbrot, C. Grebogi, E. Ott, and J. A. Yorke. Using small perturbations to control chaos. *Nature*, 363:411–417, 1993.

[142] B. Siliverstovs and D. van Dijk. Forecasting industrial production with linear, nonlinear, and structural change models. Technical report, Erasmus University Rotterdam, Netherlands, May 2003.

[143] J. C. Sprott. *Strange Attractors: Creating Patterns in Chaos.* coldwell@earthlink.net, 2000.

[144] M. Stone. Cross-validatory choice and assessment of statistical predictions. *J. of Royal Statistical Society*, B36:111–133, 1974.

[145] M. Stone. Cross-validation: A review. *Mathematische Operations Forschung Statistischen, Serie Statistics*, 9:127–139, 1978.

[146] F. Takens. Detecting strange attractors in fluid turbulence. In D. Rand and L.S. Young, editors, *Dynamical Systems and Turbulence*. Springer-Verlag, Berlin, 1981.

[147] P. F. Taylor. Modeling stochastic volatility: A review and comparative study. *Math. Finance*, 4:183–204, 1994.

[148] T. Terasvirta. Specification, estimation, and evaluation of smooth transition autoregressive models. *J. Amer. Statist. Assoc.*, 89:208–218, 1994.

[149] G. C. Tiao and R. S. Tsay. Some advances in non-linear and adaptive modelling in time series. *J. of Forecasting*, 13:109–131, 1994.

[150] H. Tong. *Nonlinear Time Series: A Dynamical System Approach.* Oxford University Press, Oxford, 1990.

[151] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[152] R. von Sachs. Modelling and estimation of the time-varying structure of nonstationary time series. *Brazilian Journal of Probability and Statistics*, 10(2):181–204, 1996.

[153] B. Wah and Y. X. Chen. Fast temporal planning using the theory of extended saddle points for mixed nonlinear optimization. *Artificial Intelligence*, (accepted for publication) 2005.

[154] B. W. Wah and Y. X. Chen. Partitioning of temporal planning problems in mixed space using the theory of extended saddle points. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, pages 266–273, November 2003.

[155] B. W. Wah and Y. X. Chen. Subgoal partitioning and global search for solving temporal planning problems in mixed space. *Int'l J. of Artificial Intelligence Tools*, 13(4):767–790, December 2004.

[156] B. W. Wah and M.-L. Qian. Constrained formulations for neural network training and their applications to solve the two-spiral problem. In *Proc. Fifth Int'l Conf. on Computer Science and Informatics*, volume 1, pages 598–601, February 2000.

[157] B. W. Wah and M. L. Qian. Time-series predictions using constrained formulations for neural-network training and cross validation. In *Proc. Int'l Conf. on Intelligent Information Processing, 16th IFIP World Computer Congress*, pages 220–226. Kluwer Academic Press, August 2000.

[158] B. W. Wah and M. L. Qian. Violation-guided learning for constrained formulations in neural network time series prediction. In *Proc. Int'l Joint Conference on Artificial Intelligence*, pages 771–776. IJCAI, August 2001.

[159] B. W. Wah and M.-L. Qian. Violation guided neural-network learning for constrained formulations in time-series predictions. *Int'l J. on Computational Intelligence and Applications*, 1(4):383–398, December 2001.

[160] B. W. Wah and M.-L. Qian. Constrained formulations and algorithms for stock price predictions using recurrent FIR neural networks. In *Proc. 2002 National Conf. on Artificial Intelligence*, pages 211–216. AAAI, August 2002.

[161] B. W. Wah and M.-L. Qian. Chapter 17: Constraint-based neural network learning for time series predictions. In N. Zhong and J. Liu, editors, *Intelligent Technologies for Information Analysis: Advances in Agents, Data Mining, and Statistical Learning*, pages 401–420. Springer-Verlag, 2004.

[162] B. W. Wah and Y. Shang. A new global optimization method for neural network training. In *Proc. Int'l Conf. on Neural Networks*, volume Plenary, Panel and Special Sessions, pages 7–11, Washington, DC, June 1996. (Plenary Speech) IEEE.

[163] B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. In *Proc. Principles and Practice of Constraint Programming*, pages 28–42. Springer-Verlag, October 1999.

[164] B. W. Wah and D. Xin. Optimization of bounds in temporal flexible planning with dynamic controllability. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, pages 40–48, November 2004.

[165] A. Waibel. Modularity in neural networks for speech recognition. *Advances in Neural Network Information Processing Systems*, pages 215–223, January 1989.

[166] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *Trans. on Acoustics Speech and Signal Processing*, 37(3):328–339, March 1989.

[167] E. Wan. Combining fossils and sunspots: Committee predictions. In *IEEE Int'l Conf. on Nerual Networks*, volume 4, pages 2176–2180, Houston, USA, June 1997.

[168] E. A. Wan. Temporal backpropagation for FIR neural networks. In *IEEE Int'l Joint Conf. on Neural Networks*, volume I, pages 575–580, San Diego, CA., 1990.

[169] E. A. Wan. *Finite Impulse Response Neural Networks with Applications in Time Series Prediction.* Ph.D. Thesis, Standford University, 1993.

[170] C. J. Watkins. *Models of Delayed Reinforcement Learning.* Ph.D. thesis, Cambridge University, Cambridge, UK, 1989.

[171] C. J. C. H. Watkins. Combining cross-validation and search. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning*, pages 79–87. Sigma Press, Cheshire, UK, 1987.

[172] C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4 (Special Issue on Reinforcement Learning)):279–292, 1992.

[173] A. S. Weigend and N. A. Gershenfeld, editors. *Time Series Prediction: Forecasting the future and understanding the past.* Addison-Wesley, 1994.

[174] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *Int'l. J. of Neural Systems*, 1(3):209, 1990.

[175] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[176] A. Willsky. Multiresolution markov models for signal and image processing. In *Proceedings of the IEEE*, volume 90, pages 1396–1458, 2002.

[177] Yahoo. Yahoo! finance web page. WWW site: http://finance.yahoo.com/.

[178] B.-L. Zhang, R. Coggins, M.A. Jabri, D. Dersch, and B. Flower. Multiresolution forecasting for future trading using wavelet decompositions. *IEEE Trans. on Neural Networks*, 12:766–775, 7 2001.

[179] G. Zheng, J. L. Starck, J. G. Campbell, and F. Murtagh. Multiscale transforms for filtering financial data streams. *J. of Computational Intelligence in Finance*, 7:18–35, 1999.

Vita

Minglun Qian received his B.S. and M.S. degree in Physics from University of Science and Technology of China in 1992 and 1995 respectively, and received his M.S. degree in Computer Science from University of Illinois at Urbana-Champaign in 1998. Since August 2003, Minglun Qian has been working for a Chicago-based hedge fund company.

Minglun Qian came to the University of Illinois in 1995 majoring in physics. In 1997, he switched to the Computer Science Department. In working towards his M.S. degree in Computer Science, he studied psedo-spectrum method to solve resonant frequency problem for 3-D optical dielectric waveguide and developed efficient algorithms to find k eigenvalues at a computational complexity of no worse than $O(kN\log(N))$ on an N-point grid. For his Ph.D. research, Minglun Qian proposed constrained formulations for neural network learning in order to provide better guidance in searching for solutions, and focused on developing efficient learning algorithms to solve the constrained formulation by combining global optimization techniques and neural network learning. Minglun Qian also spent much of his efforts on predicting stock market time series using the neural network learning tools developed and proposed innovative preprocessing approaches to handle noises in those time series.

Minglun Qian's current research interests include machine learning, nonlinear optimization, digital signal processing, numerical analysis, financial market forecasting, portfolio management, statistical arbitrage, and financial factor analysis.