ForeNet: Fourier Recurrent Neural Networks for Time Series Prediction

By

Ying-Qian ZHANG

Supervised By

Prof. Lai-Wan CHAN

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

©The Chinese University of Hong Kong

June, 2001

ForeNet: Fourier Recurrent Neural Networks for Time Series Prediction

submitted by

Ying-Qian ZHANG

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

Recurrent neural networks have been established as a general tool for fitting sequential input/output data. Traditionally, recurrent networks have been regarded as non-linear ARMA models and the parameters are chosen via some dedicated learning algorithms, such as RTRL (Real-Time Recurrent Learning) or BPTT (Back-Propagation Through Time). In this thesis, we investigate recurrent networks from another viewpoint, i.e. from the perspective of spectral analysis. We know that Fourier analysis is a useful tool for time series analysis. With some approximations, we rewrite the Fourier analysis of a time series into a recursive form. This recursive form is linked and compared with the recurrent network architecture. As a result, we derive the *ForeNet* (Fourier Recurrent Neural Networks).

ForeNet uses the Fourier recursive equation for the initialization of its weights. Some experiments on the prediction of various time series are performed. The simulation results prove the efficiency of the proposed initialization. It assigns the weights in the region not far away from the minimum. Therefore, the network output is very close to the target values even before training has been taken place.

Unlike traditional recurrent neural networks, complex values are used in *ForeNet*. Thus we apply the CRTRL (Complex Real-Time Recurrent Learning), a training algorithm for complex weights, for the learning of *ForeNet*. We show that CRTRL is able to provide fine tune to the prediction result. We have also compared *ForeNet* with some other models, such as AR model and TDNN (Time-delay Neural Networks). Experimental results show that *ForeNet* speeds up the learning, and the generalization performance is superior to traditional networks.

摘　要

反饋神經網路具有很強的自學習能力。它能通過訓練使得對於給定的輸入產生期望的輸出。對於反饋神經網路在時間序列預測的應用中，通常我們將它類比成一個自回歸移動模型，並且通過一些給定的演算法，如 RTRL（即時反饋學習演算法），來修正它的權值。在這篇論文當中，我們用頻譜分析法來觀察反饋神經網路。首先，我們用近似法將用於時間序列分析的付立葉變換轉換成一種遞迴的形式。然後，建立了遞迴方程與反饋神經網路結構之間的聯繫。根據這種聯繫，我們提出了一個新的預測模型 FORNET（付立葉反饋神經網路）。

我們利用付立葉遞迴方程來設置 FORNET 網路的初始權值。通過一些時間序列的預測實驗，我們證明了這種初始化方法的有效性。它能使網路參數在初始狀態時就接近局部極小點。這使得網路在還未進行訓練之前，它的輸出值已經比較接近期望值。

與一般的神經網路不同的是，FORNET 的參數是複數。所以傳統的訓練演算法並不適用於 FORNET。因此，我們提出了 CRTRL 演算法（複數即時反饋學習演算法）來調整網路的複權值。實驗結果證明 CRTRL 能很好地訓練網路參數，能使預測結果進一步得到改善。同時，我們還將 FORNET 與其他的一些預測模型進行比較，發現 FORNET 有相對比較快的收斂速度，並且能取得較滿意的預測結果。

Acknowledgment

I would like to take this opportunity to convey my sincere thanks and deepest gratitude to my supervisor, Professor Laiwan Chan, who has introduced me to the challenging academic world. She always has door open when I have questions, and gives me the valuable guidance during these two years.

I would also acknowledge Prof. Irwin King and Prof. Fungyu Young. They give me valuable comments on this work.

I would give thanks to Mr. Lawrence Cha and Ms. Xiu Gao, who are members in Prof. Chan's study group. They give many warm discussions and suggestions on my work, and create a pleasant atmosphere of research. I wish to thank other fellow graduate students for their encouragements.

Moreover, I would like to express my sincere gratitude to my family and my friends for their supports.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Modelling of time series is an important task in many fields of research, including medicine, economics, communication, speech processing, control, biology and mechanical engineering. Mathematical models are often used for time series prediction. If there are known underlying deterministic equations, the future observations can be predicted based on the knowledge of the past and current conditions. The relationship between the past observations and the future value of time series can be inferred by an assumed function, which can be implemented by the use of either a linear model or a nonlinear perspective, such as a Neural Network (NN). On one hand, the advantage of the linear model approach is that the calculation speed is very fast. However, the linear model is of limited applicability. On the other hand, the NN methods are powerful, but the selection of the proper architecture and learning algorithm is time consuming.

Feedforward artificial neural network has gained significant popularity during the last decade. Time Delayed Recurrent networks are essentially feedforward networks with the addition of feedback connections. Due to the existence of these recurrent links, the recurrent neural network models preserve information through time and are more powerful than the static feedforward networks, especially in dynamic problems. They have been successfully applied to many areas, such as speech recognition [59, 42], grammar learning [22, 26] and the parsing problem [34]. Time series prediction [72, 4, 64, 16, 12] is also a major applicational area of recurrent networks. The internal memory of past inputs in recurrent networks is adaptive. At present, recurrent networks are mostly formulated as nonlinear autoregression models [15] when applied to time series prediction problem. In our work, we use a novel approach to interpret and construct the recurrent networks.

1.2 Objective

The objective of this thesis is to investigate a proper recurrent neural network model for time series prediction. One approach of time series forecasting is to perform an initial analysis of the data and to choose an appropriate NN architecture, and possibly initial values for the NN parameters according to the most adequate linear model.

Here, first we decompose the time series into different frequency components by Fourier transform, and a prediction equation is achieved using the reconstruction from the frequency components. Then a recurrent network is built to interpret the proposed prediction equation. Our recurrent network is called *Fourier Recurrent Neural Networks (ForeNet).*

The Fourier transform has proven to be a versatile tool that gives a better handle on the series data. A result of using the Fourier Analysis is that the parameters involved are complex numbers. Thus *ForeNet* is a complex-valued recurrent network. At present, most commonly used neural network learning algorithms assumed real parameters. Previous works have been done to extend neural networks to cope with complex weights. The backpropagation algorithms for training a feedforward neural network with complex weights have been proposed [8] [24]. Georgious and Manolakos [38] used Complex Real Time Recurrent Learning algorithm to train a fully recurrent network. Using the complex parameters to construct neural network avoids the problem of the standstill in learning [49], and can also deal with dynamic time-sequential signal more stably and smoothly than the conventional recurrent networks [33].

In our work, *ForeNet* was trained by Complex Real Time Recurrent Learning algorithm (CRTRL), which combines [12] and [38]. We also provide a method of parameters initialization which reduces the initial network error and prevents the network from getting stuck with the initial weights. The experimental results show that *ForeNet* is computational efficient. It increases the rate of convergence and attains better generalization performance.

1.3 Contributions

We list here the main contributions of this thesis:

- Based on Fourier Analysis, we propose a *Fourier Recursive Equation*, which can compute recursively to generate the forecasts.
- Based on *Fourier Recursive Equation*, we build a recurrent model, *Fourier Recurrent Networks (ForeNet)*. The proposed network has a very simple architecture. The recurrent links only exist on the hidden units.
- The proposed method is used to initialize *ForeNet*. Some experiments are demonstrated to show the efficiency of the initialization of *ForeNet*. The optimal initialization makes the initial state of the network much close to a local minimum.

- A Complex Real Time Recurrent Learning algorithm (CRTRL) is proposed for *ForeNet* whose parameters are complex values. Trained by CRTRL, *ForeNet* increases the convergent rate and achieves better generalization performance.
- The batch-mode and the online-mode learning have been combined to do time series forecasting. We have applied this method to the prediction tasks and found it does well in the time series which show nonstationary.
- We also give a way to assess on the effective internal memory in *ForeNet*. And it is usually not easy to analyze the memory in a recurrent model.

We have published a part of our work in [79].

1.4 Thesis Overview

This thesis is organized into 8 chapters. Chapters 3 − 7 constitute the main parts of the thesis and are about the proposed model to handle time series prediction. The contents of the individual chapters are as follows:

Chapter 1 is the present chapter. It is an introduction chapter.

Chapter 2 reviews the time series prediction problems using linear models and nonlinear models, such as Neural Networks. Relevant feedforward and recurrent models and their corresponding learning algorithms are reviewed.

Chapter 3 describes the *Fourier Recursive Prediction Equation* which is derived based on Fourier analysis of time series. Further, a recurrent neural network, *ForeNet*, is built to represent the proposed prediction equation.

Chapter 4 gives more applicable architecture of *ForeNet*. Then the initialization method of *ForeNet* is proposed.

Chapter 5 analyzes the coefficients in *ForeNet* by unfolding the original recursive equations. Some experiments are demonstrated to show the performance of *ForeNet* initialization.

Chapter 6 introduces complex-valued Real Time Recurrent learning algorithm for *ForeNet*. And the *ForeNet* model is further analyzed by some designed experiments. The comparisons between *ForeNet* and some other models also provided in this chapter.

Chapter 7 suggests a on-line learning for the proposed model.

Chapter 8 is a discussion and conclusion part. We include the main limitations and advantages of *ForeNet* and mention some future works.

Chapter 2

Literature Review

A time series is a collection of observations made sequentially in time [14]. Suppose we have an observed time series $x(1), x(2), \cdots, x(N)$. Then the basic problem is to estimate future values such as $x(N+k)$, where the integer k is called the lead time. Prediction (forecasting) the future values of an observed time series is an important problem in many areas, including medicine, economics, communication, speech processing, control, biology and mechanical engineering. Commonly, time series prediction problems are approached either from a linear model or from a nonlinear model, especially, a Neural Network model. In this chapter, we give a brief introduction of these approaches.

2.1 Takens' Theorem

Embedding theorem, introduced first by Takens and extended in [61], shows that, under very general conditions, the state of a dynamic system can be accurately reconstructed by a finite windows of the time series. This window is called a time delay embedding.

$$y_t = [x_t, x_{t-\delta}, \ldots, x_{t-(n_d-1)\delta}] \tag{2.1}$$

Where x_t is the value of time series at time t, n_d is the embedding dimension, and δ is the embedding delay. Takens' theorem implies that if very general assumptions are satisfied, there exists a function $g(\cdot)$ such that

$$\begin{aligned} x_{t+1} &= g(x_t, x_{t-\delta}, \ldots, x_{t-(n-1)\delta}) \\ &= g(y_t) \end{aligned} \tag{2.2}$$

For time series prediction, this is an important theorem because it implies perfect predictions are possible using only a finite segment of the values immediately preceding the point to be predicted. In order to achieve perfect prediction, Takens' theorem requires the time series to be noise-free and the function $g(\cdot)$ to be known. However, in practice, these conditions are not satisfied and the selection of δ and n_d may critically affect the prediction accuracy. Many methods have been proposed to find δ and n_d. The goal of these methods is usually to find these parameters that can minimize the embedding dimension n_d without reducing the accuracy of the reconstructed state y_t. For simplicity but without loss of generality the embedding delay can be set equal to unity 1. One may be able to find a minimum embedding dimension to unfold the state space, but the mapping function may be too complicated to approximate.

2.2 Linear Models for Prediction

Linear time series models have two particularly desirable features[70]:they can be understand in great detail and they are straightforward to implement. However, they may be inappropriate for complicated systems. In the following section, there are discussions on the role of external inputs (moving average models) and internal memory (autoregressive models).

2.2.1 Autoregressive Model

Autoregressive Model (AR) was invented by Yule in the end of the 20's in order to predict sunspots. The general AR model expresses future values of the time series as a linear combination of past values plus a random noise component:

$$y(t) = \sum_{m=1}^{d} a_m y(t-m) + e(t) \tag{2.3}$$

This is called an dth-order autoregressive model (AR(d)). Because of the feedback loop existed in the equation, the output can continue indefinitely. So such model is called an infinite impulse response (IIR) filter.

To generate a specific realization of the series, we must specify the initial conditions, usually by the first d values of series. If there was no input, depending on the amount of feedback, after iterating it for a while, the output produced can only decay to zero, diverge, or oscillate periodically.[1]

The dependence of $y(t)$ on previous values of y also complicates the process of finding coefficients a_m that fit the model to time series data [47].

2.2.2 Moving Average Model

One of the simplest ways to use an algebraic equation to describe a system's behavior is to model its next state as a weighted sum of its previous states. That is, if one has measured a series of values $x_i(t)$, one predicts its output using the equation:

$$y(t) = \sum_{l=0}^{L} b_l x_i(t-l) \tag{2.4}$$

The model is called L^{th} order moving average (MA) model. Fitting such model to a data set involves choosing the window size L and finding appropriate values for the b_l. The impulse response of such a filter is described by

[1]In the case of a first-order AR model, this can easily be seen that if the absolute value of the coefficient is less than unity, the value of y exponentially decays to zero; if it is larger than unity, it exponentially explodes.

the coefficients b_l. As l goes from 0 to L, the impulse first "hits" b_0, then b_1, and so on. Because this response dies out after L time steps, MA model is a member of the class of finite impulse response (FIR) filters. When viewing the linear combination as a discrete filter of the noise signal, the MA model can be viewed as thus: A noise process usually has a frequency spectrum containing all or a large number of frequencies ("white" noise). A filter-like the MA model can thus cut out any desired frequency spectrum, leading to a specific, non-random time series.

Properties of the output series $y(t)$ clearly depend on the input series x. For a linear system, the response of the filter is independent of the input.

2.2.3 Autoregressive-moving Average Model

Combining MA and AR models yields the autoregressive-moving average (ARMA) model:

$$y(t) = \sum_{l=0}^{L}(b_l x_i(t-l)) + \sum_{m=1}^{d}(a_m y(t-m)) \tag{2.5}$$

The ARMA model[46] is both more general and more difficult to work with because one must choose L and d intelligently. Despite these difficulties, ARMA models dominated the time series analysis for more than half a century[70].

2.2.4 Fitting a Linear Model to a Given Time Series

In order to fit AR, MA or ARMA models to the time series prediction, two estimations should be made:

- Fitting the coefficients. The coefficients of an AR(M) model from the observed correlational structure of an observed signal. Another approach is the estimation of the coefficients as a regression problem: expressing the next value as a function of M previous values. This can be done by minimizing squared errors. For finding MA and full ARMA coefficients from observed data, standard techniques exit, often expressed as efficient recursive procedures.

- Fitting the order of the model. There are several heuristics to find the "right" order, such as the Akaike Information Criterion (AIC)[1], the minimum description length (MDL)[57] and the SVD approach[39]. There is not a unique best choice for the values or even for the number of coefficients to model a data set.

Detailed discussion on how to select the order of ARMA model can be found in [29][17][11].

2.2.5 State-space Reconstruction

An ARMA model can be rewritten as a dot product between vectors of the time-lagged variables and coefficients:

$$y_t = a \cdot Y_{t-1} + b \cdot X_{t-1} \tag{2.6}$$

where $Y_t = [y_t y_{t-1} \cdots y_{t-(d-1)}]^T$, and $a = [a_1 a_2 \cdots a_d]^T$. Such lag vectors are called tapped delay lines. They are used in the context of signal processing and time series analysis.

There is a deep connection between time-lagged vectors and underlying dynamics. This connection was proposed in 1980 by Ruelle, Packard et al (1980) and Takens (1981). Delay vectors of sufficient length are not just a representation of the state of a linear system. It turns out that delay vectors can recover the full geometrical structure of a nonlinear system.

2.3 Neural Network Models for Time Series Processing

Neural Networks form the basis of an entirely different nonlinear approach to the analysis of time series. NNs have proven to be a promising alternative to traditional techniques for nonlinear temporal prediction tasks. Recurrent and feedforward Neural Networks have been proposed[23][54] for simulating nonlinear-ARMA and nonlinear-AR models respectively. And several authors have given an overview of different types of neural networks for use in time series processing [19, 48, 40, 9]. For instance, [40] reviewed connectionist network architectures and training algorithms and provided comparisons on the different models structures and predictive power; [48] concerned the different NN models based on the type of memory: delay (akin to time windows and delays), exponential (akin to recurrent connections) and gamma (a memory model for continuous time domains). This section provides a description of various neural model structures.

2.3.1 Feed-forward Neural Networks

Among the most wide-spread neural networks are feedforward networks.

Time Delay Neural Networks

A special case in this study is the approximation of a linear AR model by a feedforward Neural Network. The approximation of an AR model of order n by a feedforward NN with n input units was proposed in [66][41]. Feedforward networks with tapped input delay lines can be used to represent time series with limited context (2.1). Such memory form is a buffer containing the n most recent inputs. It is also called a delay space embedding and forms the basis of traditional statical autoregressive (AR) model. Such network structure incorporating embedded time delays is the Time-Delay Neural Network (*TDNN*)(Figure (2.1)). The input vector X consists of the recent samples, L steps backward in time, $X(t) = [x(t), x(t-1), \ldots, x(t-L+1)]^T$. Via the weighted connections these values enter the first hidden layer of units whose outputs $y(t)$ are calculated as

$$y(t) = f(\sum_{k=0}^{L-1} w_{jk}x(t-k) + w_{jb}) \tag{2.7}$$

where w_{jb} represents a bias weight. And the outputs from the hidden layer are fed into the output layer

$$z(t) = (\sum_{j=1}^{N} w_{oj}y(t) + w_{ob}) \tag{2.8}$$

The *TDNN* can be used for nonlinear prediction of a stationary time series, i.e. a time series with statistics that remains unchanged over time [31]. When using it for prediction, we should be not only faced with the problem of determining how many hidden units are appropriate, it is also necessary to choose the number of the delay used as input. If using too few previous values then it will not be possible to capture the dynamics of the system that generated the data from which are modelling, and prediction accuracy will suffer. If using too many previous values, training the network may possibly suffer due to redundancy in the inputs as well as from the increased number of parameters to be determined from the data. Some literatures contain several suggestions on how to choose an appropriate lag space, such as trial-and-error, information theoretic methods [52] and generalization based on methods [28].

FIR Multilayer Networks

The network structure of *TDNN* is static in nature; there are no internal dynamics. In order to get a dynamic structure, Wan proposed a Finite Impulse

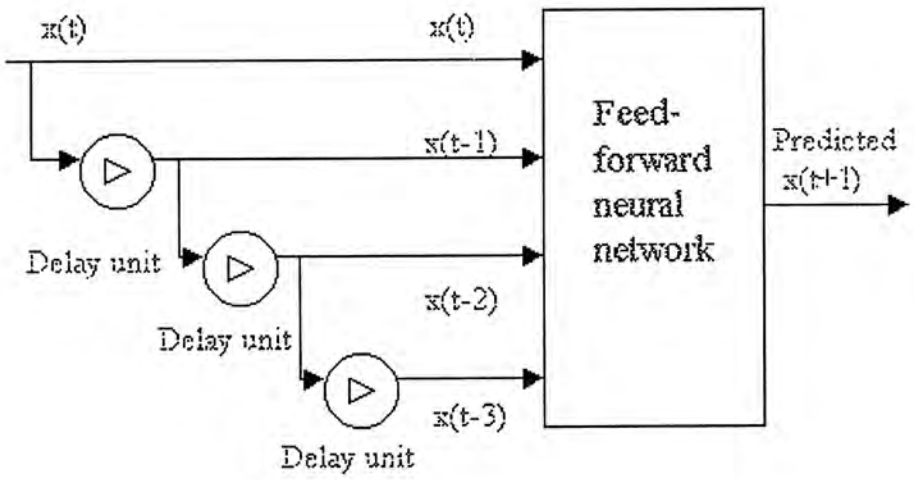


Figure 2.1: Time Delay Neural Networks

Response (FIR) model to simulate a simple Autoregressive Moving Average (ARMA) model[68][67]. A modification of the basic neuron is accomplished by replacing each static synaptic weight by an *FIR* linear filter. Figure 2.2 shows a dynamic model of a neuron using FIR filters as synapses. A synapse i is then represented by a vector

$W_i = [w_i(0), w_i(1), ..., w_i(M)]$,

and the delay line

$X_i(n) = [x_i(n), x_i(n-1), ..., x_i(n-M)]$.

The filter operation is the scalar product $W_i \cdot X_i(n)$. Finally, we have the neurons output

$$y(t) = \varphi(\sum W_i \cdot X_i(n)) \tag{2.9}$$

Wan derived temporal backpropagation algorithm to train *FIR* networks. An FIR network constitutes a powerful tool for use in time series prediction

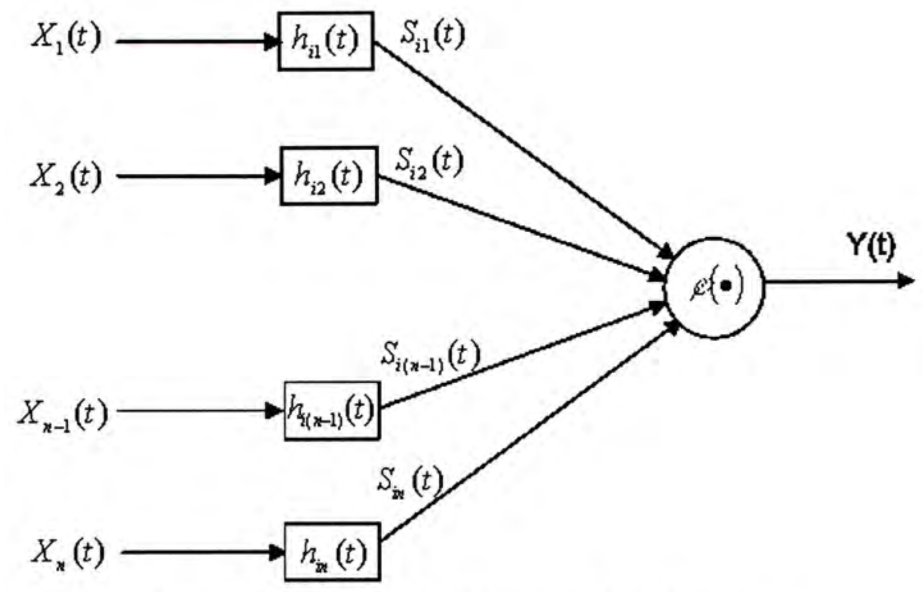


Figure 2.2: FIR Multilayer Networks

[69]. And similar with *TDNN* model, the *FIR* networks also suffer some problems.

2.3.2 Recurrent Neural Networks

Various enhancements of the basic feedforward architecture has been suggested in order to make the network "memory" of previous inputs more flexible, thus relaxing the accuracy with which the dimension of the externally provided lag space vector must be determined. In the following sections, it is described how this flexibility can be obtained by making the networks recurrent.

Jordan Networks

Jordan network[37] consists of a multilayer perceptron with one hidden layer and a feedback loop from the output layer to an additional input (or context) layer. In addition, self-recurrent loops on each unit in the context layer are introduced in this model, i.e. each unit in the context layer is connected with itself, with a weight v_i smaller than 1. Without such self-recurrent loops, the networks form a non-linear function of p past sequence elements and q past

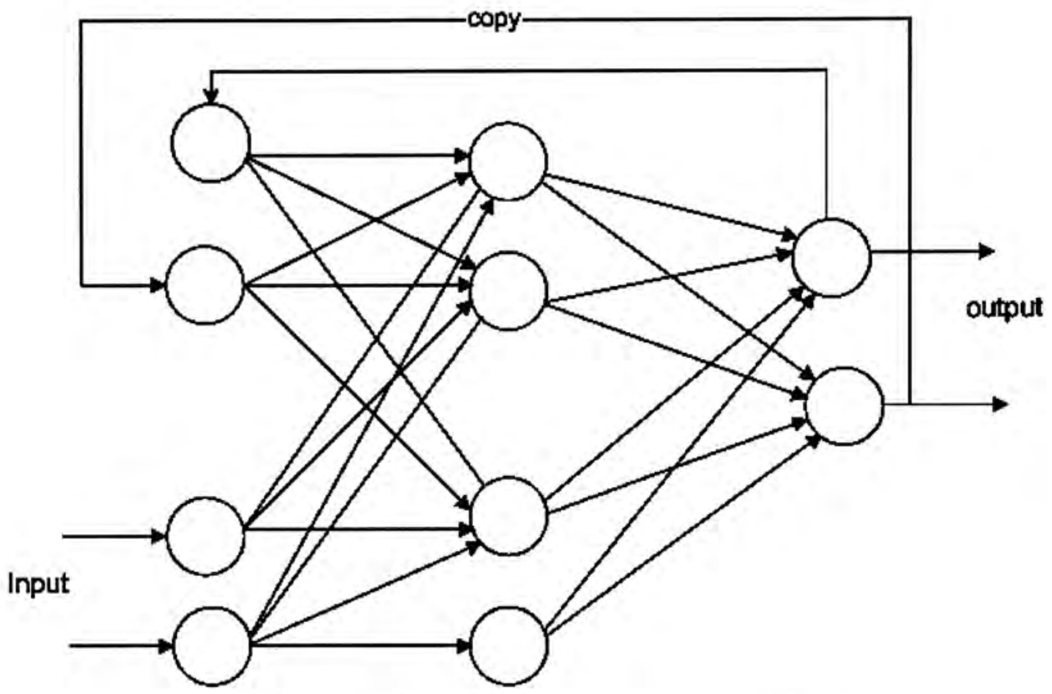


Figure 2.3: The Jordan Networks

estimates:

$$\hat{x}(t) = f(x(t-1), ..., x(t-p), \hat{x}(t-1), ..., \hat{x}(t-q)) \tag{2.10}$$

So the nonlinear ARMA model discussed above can be said to be implicitly contained in this network.

Elman Networks

The Elman network[22] is an MLP with an additional input layer, called the state layer, receiving as feedback a copy of the activations from the hidden layer at the previous time step (see Figure (2.4)) . The Elman network can be trained with any learning algorithm for MLPs, such as backpropagation or conjugate gradient. It belongs to the class of so-called simple recurrent networks(SRN)[32]. Even though it contains feedback connections, it is not viewed as a dynamical system in which activations can spread indefinitely. Instead, activations for each layer are computed only once at each time step (each presentation of one sequence vector).

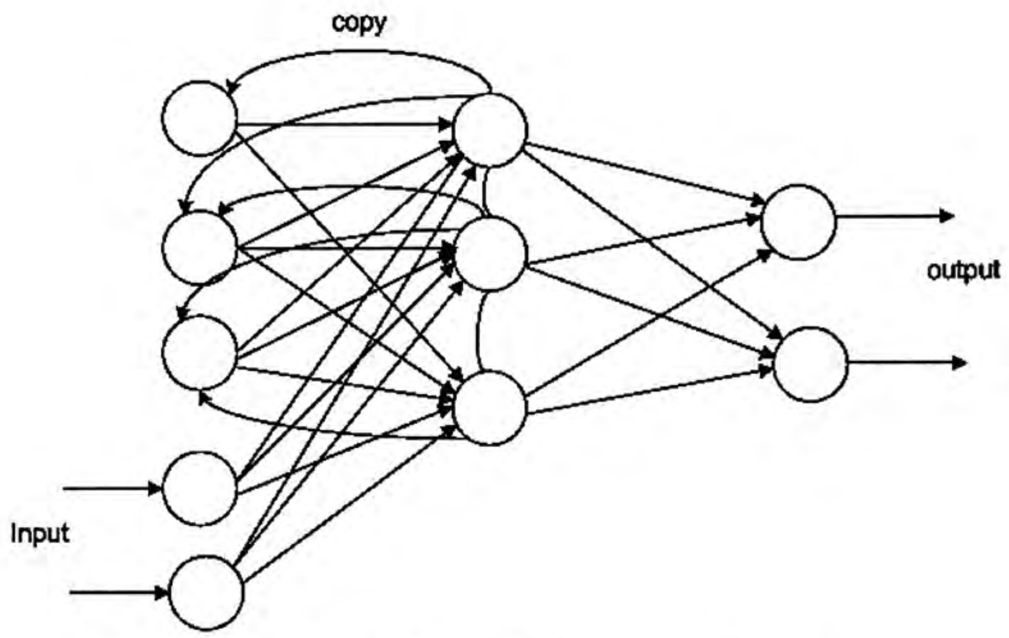


Figure 2.4: The Elman Recurrent Networks

Similar observations can be made about the Elman recurrent network as with respect to the Jordan network. A number of time steps is needed until suitable activations are available in the state layer, before learning can begin. Standard learning algorithms like backpropagation is easy to apply, but can cause problems or lead to non-optimal solutions. And this kind of recurrent net also cannot really deal with an arbitrarily long history[7]. Some examples of applications with Elman networks can be found in [27][18].

NARX Recurrent Neural Networks

NARX are inspired by nonlinear autoregressive models with exogenous inputs. They compute their current output using past inputs and past outputs. The models were proposed in [44][45] (shown in Figure (2.5)):

$$y(t) = f(x(t-D_x), ..., x(t-1), x(t), y(t-D_y), ..., y(t-1)) \tag{2.11}$$

where $x(t)$ and $y(t)$ represent input and output of the network at time t, D_x and D_y are the input-memory and output-memory order, and the function f

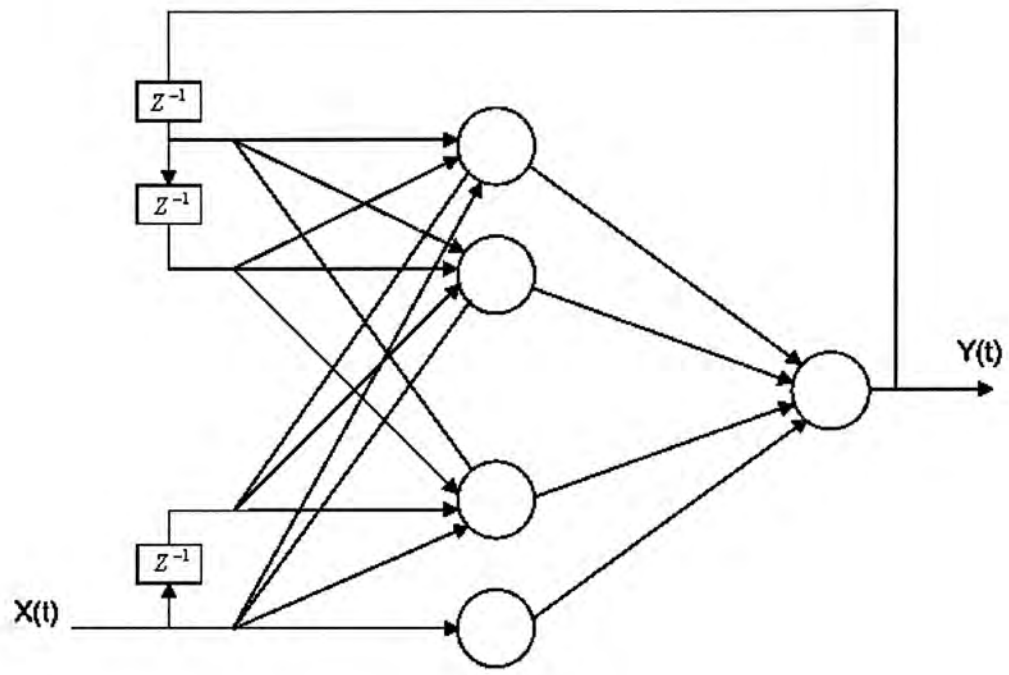


Figure 2.5: NARX Recurrent Neural Networks

is a nonlinear function. Outputs of NARX are used to compute activation values of internal nodes, which in turn are used to recompute output activations. This recurrence gives NARX the ability to encode a history of activations extending back arbitrarily in time. Empirical studies on system identification and grammatical inference problems have demonstrated that NARX networks generally converge much faster and generalize better than other network architectures,including fully recurrent networks. Furthermore, in [44][45][43] the authors proved that NARX networks can reduce the problem of learning long-term dependencies.

Long Short-term Memory Recurrent Neural Networks

A second-order approach has been proposed in [35][36]. This network is called long short-term memory(LSTM), and designed to avoid some of the limitations associated with the training of recurrent networks, such as long-term dependency in fully recurrent networks [6][7],i.e. error signals propagated back in time over the activations of a training sequence and can either blowup or vanish exponentially. The basic idea is to use both node activations and the net inputs to the nodes' transfer functions, provide a self connected node, with a linear activation function that will ensure constant error flow over the self connection. If the weights between these connections are equal to one, the error flow through the unit will be constant for infinite time steps.

The network must learn when to propagate an incoming signal into the unit or to protect the activation value from the incoming signals. If both actions need to be done by the same set of weighted connections into the unit, such connections will receive conflicting weight updates. In LSTM, a variant of RTRL learning algorithm is used which properly takes into account the altered, multiplicative dynamics caused by input and output gates.

Experiments with long input sequences, i.e. with more than 100 data points, have shown that LSTM successfully bridges long time lag, i.e. it is able to simultaneously store information over long periods of time and still learn efficiently[9].

2.3.3 Training Algorithms for Recurrent Networks

In [51], the author reviewed some learning algorithms for dynamic recurrent networks. In the following section, we only discuss two most common algorithms used for recurrent networks.

Back-Propagation Through Time

The back-propagation-through-time (BPTT) algorithm for training a recurrent network is an extension of the standard back-propagation algorithm. It may be derived by unfolding the temporal operation of the network into a layered feedforward network, the topology of which grows by one layer at every time step.

BPTT algorithm proceeds as follows (Williams and Peng, 1990)[74]:

- Let n_0 denote the start time of an epoch and n_1 denote its end time. First, a single forward pass of the data through the network for the interval (n_0, n_1) is performed. The complete record of input data, network state (i.e., synaptic weights of the network), and desired responses over this interval is saved.

- A single backward pass over this past record is performed to compute the values of the local gradients

$$\delta_j(n) = -\frac{\partial \varepsilon_{\text{total}}(n_0, n_1)}{\partial \nu_j(n)} \tag{2.12}$$

where

$$\varepsilon_{\text{total}}(n_0, n_1) = \frac{1}{2} \sum_{n=n_0}^{n_1} \sum_{j \in \Re} e_j^2(n) \tag{2.13}$$

for all $j \in \Re$ and $n_0 < n \leq n_1$. $e_j(n)$ is the error signal at the output of a neuron measured with respect to some desired response. This computation is performed by using the formula:

$$\delta_j(n) = \begin{cases} \varphi' \nu_j(n) e_j(n) & \text{for } n = n_1, \\ \varphi'(\nu_j(n))[e_j(n) + \sum_{k \in \Re} w_{jk} \delta_k(n+1)] & \text{for } n_0 < n < n_1 \end{cases} \tag{2.14}$$

Where $\varphi'()$ is the derivative of an activation function with respect to its argument, and $\nu_j(n)$ is the induced local field of neuron j. The use of this equation is repeated, starting from time n_1 to time n_0.

- Once the computation of back-propagation has been performed back to time $n_0 + 1$, the following adjustment is applied to the synaptic weight w_{ji} of neuron j:

$$\begin{aligned}\Delta w_{ji} &= -\eta \frac{\partial \varepsilon_{\text{total}}(n_0, n_1)}{\partial w_{ji}} \\ &= \eta \sum_{n=n_0+1}^{n_1} \delta_j(n) x_i(n-1)\end{aligned} \tag{2.15}$$

  where η is the learning rate parameter and $x_i(n-1)$ is the input applied to the ith synapse of neuron j at time $n-1$.

Real-Time Recurrent Learning (RTRL)

Another learning algorithm is real-time recurrent learning (RTRL). The synaptic weights of a fully connected recurrent network in real time, that is, while the network continues to perform its signal processing function (Williams and Zipser, 1989).

RTRL algorithm proceeds as follows:

- Initialization:
  - Set the synaptic weights of the algorithm to small values selected from a uniform distribution.
  - Set the initial value of the state vector $\mathbf{x}(0) = \mathbf{0}$.
  - Set $\mathbf{\Lambda}_j(0) = \mathbf{0}$ for $j = 1, 2, ..., q$.
- Computations: Compute for $n = 0, 1, 2, ...,$

$$\mathbf{\Lambda}_j(n+1) = \phi(n)[\mathbf{W}_a(n)\Lambda_j(n) + \mathbf{U}_j(n)] \tag{2.16}$$

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{C}\mathbf{x}(n) \tag{2.17}$$

$$\Delta \mathbf{w}_j(n) = \eta \mathbf{C}\mathbf{\Lambda}_j(n)\mathbf{e}(n) \tag{2.18}$$

Three matrices $\mathbf{\Lambda}_j(n)$, $\mathbf{U}_j(n)$, and $\phi(n)$ are described as follows:

$\mathbf{\Lambda}_j(n)$ is a $q - by - (q + m + 1)$ matrix defined as the partial derivative of the state vector $\mathbf{x}(n)$ with respect to the weight vector $\mathbf{w}_j$:

$$\mathbf{\Lambda}_j(n) = \frac{\partial \mathbf{x}(n)}{\partial \mathbf{w}_j} \qquad j = 1, 2, ..., q \tag{2.19}$$

$\mathbf{U}_j(n)$ is a $q - by - (q + m + 1)$ matrix whose rows are all zero, except for the jth row that is equal to the transpose of vector $\mathbf{W}_j^T \xi(n)$:

$$\mathbf{U}_j(n) = \begin{bmatrix} 0 \\ \xi^T(n) \\ 0 \end{bmatrix} j = 1, 2, ..., q \tag{2.20}$$

$\phi(n)$ is a $q - by - q$ diagonal matrix whose kth diagonal element is the partial derivative of the activation function with respect to its argument, evaluated at $\mathbf{w}_j^T \xi(n)$:

$$\phi(n) = diag(\varphi'(\mathbf{w}_1^T \xi(n)), ..., \varphi'(\mathbf{w}_j^T \xi(n)), ..., \varphi'(\mathbf{w}_q^T \xi(n))) \tag{2.21}$$

- Parameters:

  $m =$ dimensionality of input space;

  $q =$ dimensionality of state space;

  $p =$ dimensionality of output space;

  $\mathbf{w}_j =$ synaptic weight vector of neuron j, $j = 1, 2, ..., q$.

The main problem with BPTT is the need for large resources because of the duplication of units. For long sequences, or for sequences of unknown length, the approach becomes impractical. Compared to BPTT, RTRL can be run on-line, learning while sequences are being presented rather than after they are complete, and it doesn't need to duplicate the units. Several examples investigated by Williams and Zipser [1989][75] demonstrate the power and generality of the RTRL method.

2.4 Combining Neural Networks and other approximation techniques

Recently, the Neural Networks are be complemented with other successful approximation techniques based on wavelets [50][3][78], kernel estimators, nearest neighbors, B-splines, projection pursuit regression and fuzzy models [21].

In [78], the authors proposed a wavelet network as an alternative to feedforward neural networks for approximating arbitrary nonlinear function based on the wavelet transform theory. The basic idea of the wavelet network is to replace the neurons by "wavelons", i.e., computing units obtained by cascading an affine transform and a multidimensional wavelet. Then these transforms and the "synaptic weights" are identified from noise input/output data. The authors mentioned some properties of such wavelet network. First, because wavelet decomposition is a powerful tool for approximation, the network can guarantee the "universal approximation" property. Second, they built an explicit link between the network coefficients and some appropriate transform, which can automatically provided by the wavelet decomposition. And an initial guess for the network parameters can be derived by using the decomposition formula, which also helped drastically to improve backpropagation algorithm behavior. Inspired by [78] and [75], a Recurrent Wavelet Network was proposed in [55], which was trained by the real time recurrent learning algorithm.

The forecasting strategy in [3] is based on the subdivision of the prediction task into elementary tasks. They used a particular wavelet transform aimed at laying bear useful information, which is then treated by the neural network. The wavelet transform decomposes the input signal into detail signals, and a residual. The original signal can be expressed as an additive combination of the wavelet coefficients, at the different resolution levels. Therefore, it suffices to run a DRNN model on each resolution level and then recombine the individual predictions to form the final forecast.

Chapter 3

ForeNet: Model and Representation

In this chapter, a prediction model, *ForeNet*, is proposed. In the first section, we derive a prediction equation based on Fourier analysis of time series. The equation, called *Fourier Recursive Equation*, is in a recursive form. In the second section, we build some links between *Fourier Recursive Equation* and Neural Networks. The proposed equation is then represented by a recurrent network. We call this network *Fourier Recurrent Neural Networks (ForeNet).*

3.1 Fourier Recursive Prediction Equation

The Fourier Analysis of a time series is to decompose the time series into a sum of sinusoidal components [10]. According to the basic result of Fourier analysis, it is always possible to approximate an arbitrary analytic function defined over a finite interval of the real line, to any desired degree of accuracy, by a weighted sum of sine and cosine functions of harmonically increasing frequencies [53]. In our method, we decompose the time series into several frequency components, then reconstruct them to derive a prediction function, which is used for time series forecasting.

3.1.1 Fourier Analysis of Time Series

Consider a sequence of time series data $x(1), \cdots, x(t), \cdots, x(N)$. As the time is discrete, we use the Discrete Fourier Transform (DFT) to form

$$c_n = \sum_{t=1}^{N} x(t) \exp(-j\omega_n t) \qquad n = 1, 2, ..., N \tag{3.1}$$

in this equation[1] , $\omega_n = \frac{2\pi n}{N}$ is a so-called Fourier frequency. Equation (3.1) is called the Fourier decomposition of c_n. The data $x(t)$ may be recovered from the inverse transform

$$x(t) = \frac{1}{N} \sum_{n=1}^{N} c_n \exp(j\omega_n t) \qquad t = 1, 2, ..., N \tag{3.2}$$

Thus the finite sequence can be expressed exactly using all c_n, where n is from 1 to N. Now in Equation (3.2), we define

$$h_n(t) = \frac{1}{N} c_n \exp(j\omega_n t) \tag{3.3}$$

So Equation (3.2) becomes

$$x(t) = \sum_{n=1}^{N} h_n(t) \qquad t = 1, 2, ..., N \tag{3.4}$$

3.1.2 Recursive Form

In this part, we derive the recursive equation to calculate the time series value at the next time step $x(t+1)$ based on its previous data $x(1), x(2), \cdots, x(t)$.

From Equations (3.2) and (3.3), we can get the value of $x(t+1)$:

[1]In the original DFT, the range of summation should be $n = 0, 1, ..., N-1$. Here, in order to be consistent with the expression of time series $x(1), \cdots, x(N)$, we modify the index as $n = 1, 2, ...N$. And there have no effect on the transformation results.

$$\begin{aligned} x(t+1) &= \frac{1}{N}\sum_{n=1}^{N} c_n \exp(j\omega_n(t+1)) \\ &= \sum_{n=1}^{N} h_n(t)\exp(j\omega_n) \end{aligned} \tag{3.5}$$

for $t = 1, 2, ..., N-1$.

It suggests that $x(t+1)$ can be estimated by the value of h at the last time step t. Suppose we append the series $x(1), x(2), \cdots, x(N)$ by an extra value $x(N+1)$ and the series becomes $x(1), \cdots, x(t), \cdots, x(N), x(N+1)$. The number of data is changed to $N+1$. Therefore the new $N+1$ points DFT is updated as follows

$$c'_n = \sum_{t=1}^{N+1} x(t)\exp(-j\omega'_n t) \tag{3.6}$$

for $n = 1, 2, ..., N+1$, where $\omega'_n = \frac{2\pi n}{N+1}$. If the length of time series, N, is sufficiently large, we approximate ω'_n by ω_n. Thus

$$c'_n \cong c_n + x(N+1)\exp(-j\omega_n(N+1)) \tag{3.7}$$

for $n = 1, 2, ...N$. The corresponding $h'_n(N+1)$ becomes

$$\begin{aligned} h'_n(N+1) &= \frac{1}{N+1} c'_n \exp(j\omega'_n(N+1)) \\ &\cong \frac{c_n \exp(j\omega_n(N+1)) + x(N+1)}{N+1} \\ &= \frac{N}{N+1} h_n(N)\exp(j\omega_n) + \frac{x(N+1)}{N+1} \end{aligned} \tag{3.8}$$

It suggests that the value of the function h'_n at time $N+1$ can be derived from its value at previous time step N and the current time series value

$x(N+1)$. Given an initial value $h(0)$, the equations can be used recursively to compute forecasts. And forecasts can easily be achieved using only the latest observation and the previous forecast:

$$\begin{aligned}
\hat{x}(t+1) &= \sum_{i=1}^{N} h_i(t)\exp(j\omega_i') \\
h_i(t+1) &= \frac{N}{N+1}\exp(j\omega_i)h_i(t) + \frac{1}{N+1}x(t+1)
\end{aligned} \tag{3.9}$$

where $x(t)$ is model input. We call this equation *Fourier Recursive Equation*, that can be used to compute the future value. In this equation, only the current series data, $x(t)$, acts as the model input to calculate the next time step value $x(t+1)$. The other past values are stored in the form of the internal memory of the model, and they are memorized by $h(t)$.

3.2 *Fourier Recurrent Neural Network Model (ForeNet)*

As we mentioned before, time series prediction can be approached either from a linear model or from a nonlinear perspective, such as a Neural Network model. In the previous section, the *Fourier Recursive Equation* has been derived to perform time series prediction. On one hand, an obvious advantage of such linear approach is the calculation speed is very fast. However, the linear model is of limited applicability. On the other hand, the NN methods are powerful, but the selection of the proper architecture and learning algorithm is time consuming. In this section, we build a direct relationship between time series and Neural Network model by using a recurrent network to perform the proposed Fourier recursive prediction equation. In this way, the network architecture and its accordingly learning algorithm can be properly determined.

3.2.1 Neural Networks Representation

We have shown that the Fourier recursive equation can be approximated as Equation (3.9), which is in a recursive form. Thus, when we represent the model by a Neural Network, we need a recurrent NN because it can contain internal memory by the feedback weights.

Now we consider our input to the recurrent network is $x(t)$ at each time step t. The network output and hidden units outputs are obtained as follows.

$$\begin{aligned} z(t) &= \sum_{i=1}^{N} v_i y_i(t) \\ y_j(t+1) &= f(\sum_{j=1}^{N} W_{ij} y_j(t) + u_i x(t+1)) \end{aligned} \tag{3.10}$$

where f represents the activation function of the hidden units. W_{ij} is the recurrent weights. u_i and v_i are the weights connecting input to hidden units and hidden to output units. N is the number of hidden units in the network.

To predict the future time series $x(t+1)$ based on historical observations $x(1), x(2), ..., x(t)$, we input $x(t)$ to the network at each time step t. The network is expected to output the next step prediction $x(t+1)$, i.e., prediction of $x(t+1)$ can be obtained after we have inputted the whole sequence, $x(t)$, to the network. As recurrent networks are used, the hidden units can also serve as internal memories to store the intermediate states of the system.

Comparing neural network representation (3.10) to the Fourier Recursive Equation (3.9), we can find a direct correspondence between them. So based on the proposed equation, we build *ForeNet* model as follows:

- First, we remove all recurrent links except those are self-connections on the hidden units.

- Let N represents the number of hidden units in the recurrent model.
- The output of each hidden unit y_i in recurrent model is approximated by h_i in Equation (3.9). Hence, each hidden unit has recurrent weight connected only to itself.
- As for the weights in the recurrent network, let $u_i = \frac{1}{N+1}$, $v_i = \exp(j\omega_i)$, and $W_{ij} = \frac{N}{N+1}\exp(j\omega_i)$ for $j = i$.
- The accordingly recurrent model has the linear activation function both on hidden layer and output layer according to Equation (3.9).

In this way, the output of the network $z(t)$ represents the predicted value $x(t+1)$.

3.2.2 Architecture of *ForeNet*

According to neural network representation, the hidden unit $h(t)$ is derived from its previous value $h(t-1)$ recursively. It suggests that there exist recurrent weights in the hidden layer of neural networks and the recurrent links are diagonally connected only, that is, the recurrent weight matrix $R^{n\times n}$ is in the following form

$$R = \begin{pmatrix} r_{11} & 0 & \cdots & \cdots & 0 \\ 0 & r_{22} & & & \\ 0 & 0 & \ddots & & \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & \cdots & \cdots & r_{nn} \end{pmatrix} \tag{3.11}$$

On one hand, Fully Recurrent Network (FRN) is a very general architecture, which has a great freedom to build its own internal representations for encoding temporal information, and has shown a strong ability to time series learning. On the other hand, locally connected recurrent models have been suggested as an alternative to FRN because of its gain in complexity in both computational time and storage space [12]. In addition, it has been shown that a FRN with linear activation functions is inherently equivalent to some band diagonally connected recurrent networks [13]. Therefore, we adopt the architecture of the self-connected locally connected recurrent network.

The model architecture is shown in Figure 3.1. The model is made up three layers, input layer, hidden layer and output layer. The inter-layer connections are all running forward and recurrent links exist in the hidden layer only. In the model (3.9), we only use one current input to predict the value at the next time step, that is, only one input and one output exist in the neural network. And according to the links built above, there are N number of hidden units in the hidden layer.

Since the value of N is related to the length of time series to perform Fourier Transform, we will show in the next chapter how to estimate the value of N to make the recurrent network applicable while not losing much information during Fourier transform.

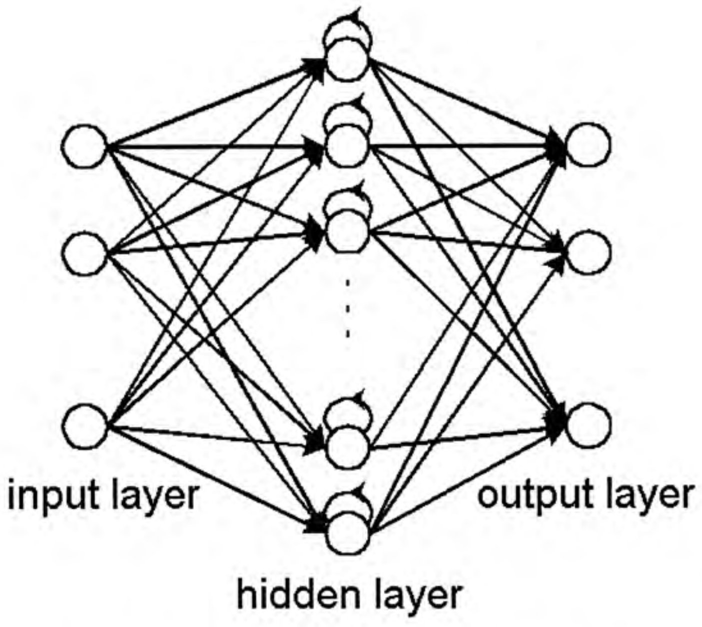


Figure 3.1: Architecture of *ForeNet*

Chapter 4

ForeNet: Implementation

In the previous chapter, we have built *ForeNet* to represent the proposed Fourier prediction equation. The architecture of the model is $1 - N - 1$, i.e. one input unit, N hidden units and one output unit. According to Fourier transform, N should be a large number because it is the length of time series.

Further, since *ForeNet* is built based on Fourier analysis, its parameters are complex values, and because of some approximations made during the model derivation, the predicted values computed from the equation are also complex numbers. However, most of time series are real-valued.

These two problems, a large number of hidden units in the network and the complex-valued output, make *ForeNet* inapplicable when it is applied to time series forecasting.

In this chapter, we will do a little modifications on the proposed prediction equation to make the implementation of *ForeNet* easier.

4.1 Improvement on *ForeNet*

Since *ForeNet* is proposed by NNs analysis and Fourier analysis, we would modify *ForeNet* by the idea from these two methods.

4.1.1 Number of Hidden Neurons

In neural networks, the number of hidden units should be not too large, however, when the Fourier transform is performed on a time series, the period can not be too small or it can not keep original information well after transformation. In this section, we consider the selection of number of hidden units in *ForeNet* as follows.

- In the view of the architecture of NNs, in *ForeNet*, there are one input unit, one output unit and N hidden units. Based upon the proposed method, the output unit is evaluated using the summation of N terms, where N corresponds to the number of hidden units. And it is not practical to have a recurrent network with N hidden units where N is comparable to the length of the sequence, which is supposed to be long. So we would use a smaller value p ($p < N$) to construct the recurrent network. In most time-series, the component frequencies are usually with low orders. High order frequencies usually are of small magnitude and they may correspond to the undesirable noise term. Thus, we can assume that the higher frequency terms are negligible in Equation (3.5).

- As we know, most of time series are nonstationary, that is, the frequency of the signal is changing during the measurement interval. So if we suspect that the frequency content of the series may change while it is being measured, we may need a different analysis procedure in stead of doing DFT on the whole series. In Equation (3.9), we replace N (the length of time series) by p, where p is a fixed small positive value. In this way, we use the shorter section of data to do Discrete Fourier Transform. Moreover, the prediction task is single-step prediction[1] . To approach the short-term prediction, by analyzing only a small number of neighbors, we may get more relevant information of the predicted data.

4.1.2 Real-valued Outputs

Since most of time series are real values, when we apply *ForeNet* to compute the forecasts, we need to transform the complex-valued outputs to the real values. There are two possible ways to do such transformation. The first one is to keep the real part of the complex data and discard the imaginary part; the second one is to use the magnitude of the complex data as the model output directly. However, both approximations would cause the loss of the original information.

In this section, we will introduce a way to make the outputs of *ForeNet* become the real values. The proposed estimation is based on the symmetry properties of the Fourier transform.

Symmetry Properties of the Fourier Transform

The basic properties of the sine and cosine functions which underlie the Fourier transform give rise to certain symmetry conditions which lead to useful simplifications in the computing of the discrete Fourier transform of a finite sequence[53]. To demonstrate the symmetry conditions, we rewrite Equation (3.1) as

[1]when we mention prediction in our work, if no especial statement, we mean one-step ahead prediction.

$$
\begin{aligned}
c_n &= \sum_{t=1}^{N} x_t \exp(-j\omega_n t) \\
&= \sum_{t=1}^{N} (x_t^{re} + jx_t^{im})(\cos(\omega_n t) - j\sin(\omega_n t)) \\
&= \sum_{t=1}^{N} (x_t^{re}\cos(\omega_n t) + x_t^{im}\sin(\omega_n t)) \\
&- j\sum_{t=1}^{N} (x_t^{re}\sin(\omega_n t) - x_t^{im}\cos(\omega_n t)) \\
&= \frac{1}{2}(\alpha(\omega_n) - j\beta(\omega_n)).
\end{aligned}
\tag{4.1}
$$

When there is no presumption that x_t is real, Equation (3.2) can be written as

$$
\begin{aligned}
x_t &= \frac{1}{N}\sum_{n=1}^{N} c_n \exp(j\omega_n t) \\
&= \frac{1}{2N}\sum_{n=1}^{N} (\alpha(\omega_n) - j\beta(\omega_n))(\cos(\omega_n t) + j\sin(\omega_n t)) \\
&= \frac{1}{2N}\sum_{n=1}^{N} (\alpha(\omega_n)\cos(\omega_n t) + \beta(\omega_n)\sin(\omega_n t)) \\
&+ j\frac{1}{2N}\sum_{n=1}^{N} (\alpha(\omega_n)\sin(\omega_n t) + \beta(\omega_n)\cos(\omega_n t) \\
&= x_t^{re} + jx_t^{im}.
\end{aligned}
\tag{4.2}
$$

Consider setting $x_t^{re} = x_t$ and $x_t^{im} = 0$ in (4.1), which is the case when $x(t) = x_t$ is a real-valued series. Then Equation (4.1) becomes

$$
\begin{aligned}
c_n &= \sum_{t=1}^{N} (x_t\cos(\omega_n t) - jx_t\sin(\omega_n t)) \\
&= \frac{1}{2}(\alpha(\omega_n) - j\beta(\omega_n)).
\end{aligned}
\tag{4.3}
$$

In view of the properties of the trigonometrical functions, it can be seen that $x(t)$ has a Fourier transform of which the real part $\alpha(\omega_n) = \alpha(-\omega_n)$ is now an even function and the imaginary part $\beta(\omega_n) = -\beta(-\omega_n)$ is now an odd function. When the latter conditions are applied to Equation (4.2), it can be seen that the imaginary term vanishes, since the trigonometric functions sweep out exactly one period across the time extent.

Improvement

When we derived the prediction equation, we made some approximations, which may cause information loss and original real-valued series can not be guaranteed in Equation (3.5). Moreover, the inverse Fourier transform requires exactly N number of c_n to recover the original series. If only a small number of c_n can be chosen, the phase will not be aymmetrized and the inverse Fourier transform would not be real, thus Equation (3.5) can not be fulfilled and $x(t+1)$ will not be a real number anymore.

In Equation (3.5), only the first small number of c_n is chosen to compute the output. Assume we are using all Fourier coefficients c_n to recover the original series, then the outputs of the equation should be real values because of the contributions of the symmetrical c_n, where the imaginary parts are cancelled. In order to get real data, we should only choose the real part of the output in Equation (3.5). Furthermore, we would use two times of the real part as the predicted output imagining the last a few c_n should be included. Therefore, the Fourier recursive equations (3.5) (3.8) are modified as:

$$\begin{aligned} \hat{x}(t+1) &= 2 \times REAL(\sum_{n=1}^{p} h_n(t)\exp(j\omega_n')) \\ h_n(t) &= \frac{p}{p+1} h_n(t-1)\exp(j\omega_n) + \frac{x(t)}{p+1} \end{aligned} \tag{4.4}$$

And *ForeNet* (Equations (3.10)) is modified as:

$$\begin{aligned} z(t) &= 2 \times y^{re}(t) \\ y(t) &= \sum_{j=1}^{p} h_j(t) W_{jl} \\ h_j(t) &= W_{jj} h_j(t-1) + W_{kj} x(t) \end{aligned} \tag{4.5}$$

The outputs of the network become real. *ForeNet* is now more applicable for NNs learning.

4.2 Parameters Initialization

An initial guess for the network parameters can be derived by the links found in the previous section. Comparing the *Fourier Recursive Equation* (4.4) and *ForeNet* model (4.5), we have an initial estimation of the weights and the value of hidden units at initial time. The proposed initialization is as follows:

1. The recurrent weights to hidden units, W_{ij}, can be initialized based upon the coefficients of $h_n(t-1)$ in Equation (4.4). That is

$$W_{ij}(0) = \frac{p}{p+1} \exp(j\omega_n) \tag{4.6}$$

where p represents the number of hidden units, $w_n = \frac{2\pi n}{p+1}$.

Similarly, the weights between output units and hidden units are initially estimated by

$$W_{jl}(0) = \exp(j\omega'_n) \tag{4.7}$$

where $w'_n = \frac{2\pi n}{t+1}$. n represents the n^{th} hidden units.

The weights between input and hidden units

$$W_{kj}(0) = \frac{1}{p+1} \tag{4.8}$$

When we determine the value of p, accordingly the number of hidden units and the initial weights in *ForeNet* can be determined.

2. The recurrent model needs an initial estimation of the state of hidden units. In the Fourier prediction Equation (4.4), the internal memory is controlled by nature of the time series as well as the model coefficients. We just randomly initialize the values of h_n and let the model build proper internal states.

Thus, the network initialization is finished. *ForeNet* a recurrent model, with the architecture as shown in Figure 3.1 and with the initialization method as described above.

4.3 Application of *ForeNet*: the Process of Time Series Prediction

In the forecasting task, we would use the recurrent network, *ForeNet*. We can perform time series prediction as following three stages.

- **Initialization Stage.** Before *ForeNet* has been trained, it is initialized by the proposed method described in the previous section. We will analyze this initialization method and show its performance in the next chapter, Chapter 5.

- **Training Stage.** Then for NNs learning, Complex Real Time Recurrent Learning Algorithm (CRTRL) is used to update complex-valued parameters in *ForeNet*. We will introduce the algorithm in Chapter 6.

- **Prediction Stage.** After parameter updates, we apply the *ForeNet* to perform prediction on various time series. This stage will be introduced in Chapter 6 and Chapter 7.

4.4 Some Implications

There are some more implications of using the correspondence between the *Fourier Recursive Equation* and its NN representation, *ForeNet.*

- First, *ForeNet* is built based upon the *Fourier Recursive Equation.* The parameters in *ForeNet* are corresponding to the coefficients in the proposed equation. At the beginning of training, we can first determine the coefficients in *ForeNet*, then the architecture of *ForeNet* is fixed and furthermore, initialization of *ForeNet* is achieved.

- The weights of the recurrent networks are pre-defined constants and are evaluated from the input sequence. However, it has to be computed off-line. If we do not allow any off-line computation of the weights, we can apply the recursive equations to estimate the weights. In addition, in the computation, we have made a few approximations. Further, in case if the time series is non-stationary, the pre-defined weights cannot reflect the latest statistics of the time series. Thus, a learning algorithm to adapt the weights is still desirable.

- The internal memory in *ForeNet* is stored in the hidden units. When the training hasn't been started, we can analyze the effective memory in the model.

Chapter 5

ForeNet: Initialization

We have built a recurrent model, *ForeNet*, to represent the proposed prediction equation. In this chapter, in order to analyze the prediction model in a more convenient way, first, we will unfold the recursive equation and suggest some properties of the model. After that, *ForeNet* with initial parameters is applied to time series prediction. The experimental results show the proposed initialization method for *ForeNet* is efficient.

5.1 Unfolded Form of *ForeNet*

The proposed model (4.4) is in an recursive form. In this section, we expand the recursive equation. In the expanded form, the next time step value $x(t+1)$ can be directly represented by its previous value $x(t), x(t-1), \cdots, x(1)$ with different coefficients.

First, we use three variables α_n, β_n and γ_n to replace the coefficients in Equation (4.4). Where $\alpha_n = \frac{p}{p+1}\exp(j\omega_n)$, $\beta_n = \frac{1}{p+1}$, $\gamma_n = \exp(j\omega'_n)$, and $\omega_n = 2\pi n/(p+1)$ $\omega'_n = 2\pi n/(t+1)$. Equation (4.4) is then rewritten by

$$\begin{aligned}\hat{x}(t+1) &= 2 \times (g(t+1))^{re} \\ g(t+1) &= \sum_{n=1}^{p} h_n(t)\gamma_n \\ h_n(t) &= h_n(t-1)\alpha_n + x(t)\beta_n \end{aligned} \tag{5.1}$$

where, $x(t)$ is model input.

Now we unfold the third equation in Equation (5.1) and computes $h_n(t)$ iteratively as follows

$$\begin{aligned}h_n(t) &= h_n(t-1)\alpha_n + x(t)\beta_n \\ &= \alpha_n[h_n(t-2)\alpha_n + x(t-1)\beta_n] + x(t)\beta_n \\ &= \alpha_n^2 h_n(t-2) + \alpha_n\beta_n x(t-1) + \beta_n x(t) \\ &= \alpha_n^3 h_n(t-3) + \alpha_n^2\beta_n x(t-2) + \alpha_n\beta_n x(t-1) + \beta_n x(t) \\ &\quad\vdots \\ &= \beta_n x(t) + \alpha_n\beta_n x(t-1) + \alpha_n^2\beta_n x(t-2) + \ldots + \alpha_n^{t-1}\beta_n x(1) + \alpha_n^t h_n(0) \\ &= \sum_{d=0}^{t-1} \alpha_n^d \beta_n x(t-d) + \alpha_n^t h_n(0)\end{aligned} \tag{5.2}$$

where d represents the time-lag to the current input. In this way, we use infinite past data $x(t)$ and the initial value of h_n to represent the current value of $h_n(t)$. Now using unfolded $h_n(t)$ to rewrite $g(t+1)$ in Equation (5.1)

$$\begin{aligned}g(t+1) &= \sum_{n=1}^{p} h_n(t)\gamma_n \\ &= \sum_{n=1}^{p}(\sum_{d=0}^{t-1}(\alpha_n^d\beta_n\gamma_n x(t-d)) + \alpha_n^t\gamma_n h_n(0)))\end{aligned} \tag{5.3}$$

According to the first equation in Equation (5.1), we only use the real part of $g(t+1)$ to estimate the value of $x(t+1)$. In order to take the real part from the complex number in Equation (5.3), we replace α_n, β_n and γ_n by the original coefficients, that is, $\alpha_n = \frac{p}{p+1}\exp(j\omega_n)$, $\beta_n = \frac{1}{p+1}$, $\gamma_n = \exp(j\omega_n')$, so Equation (5.3) becomes

$$\begin{aligned} g(t+1) = \sum_{n=1}^{p}(\sum_{d=0}^{t-1}(\frac{p^d}{(p+1)^{d+1}}\exp(j(\omega_n d+\omega_n'))x(t-d)) \\ +(\frac{p}{p+1})^t\exp(j(\omega_n t+\omega_n'))h_n(0)) \end{aligned} \tag{5.4}$$

As we know, the complex value $\exp(ja)$ can be expressed by a real part $\cos a$ and an image part $\sin a$, that is, $\exp(ja) = \cos a + j\sin a$. To get the real part of $g(t+1)$, we can use $\cos(\cdot)$ to replace the complex number $\exp(\cdot)$ in Equation (5.4). Thus the estimation of $x(t+1)$ can be achieved by

$$\begin{aligned} \hat{x}(t+1) &= 2\times(g(t+1))^{re} \\ &= 2\sum_{n=1}^{p}(\sum_{d=0}^{t-1}(\frac{p^d}{(p+1)^{d+1}}\cos(\omega_n d+\omega_n')x(t-d)) \\ &\quad +(\frac{p}{p+1})^t\cos(\omega_n t+\omega_n')h_n(0)) \end{aligned} \tag{5.5}$$

In the unfolded Equation (5.5), $x(t+1)$ is estimated by a set of past values $x(t)$. In order to make the expression Equation (5.5) simpler, we introduce two set of variables $\mu_n(d)$ and ν_n to represent the coefficients of different past value. Define

$$\begin{aligned} \mu_n(d) &= \frac{p^d}{(p+1)^{d+1}}\cos(\omega_n d+\omega_n') \\ \nu_n &= (\frac{p}{p+1})^t\cos(\omega_n t+\omega_n') \end{aligned} \tag{5.6}$$

thus, Equation (5.5) becomes

$$\hat{x}(t+1) = 2 \times \sum_{n=1}^{p}(\sum_{d=0}^{t-1}(\mu_n(d)x(t-d)) + \nu_n h_n(0)) \tag{5.7}$$

Equation (5.7) is the unfolded form of Equation (4.4). d represents the time-lag between the current model input and previous input. Time series value at next time step $x(t+1)$ is estimated by the previous t data points. Equation (5.7) is a linear model. Compared to the recursive Equation (4.4), it has one more variable, d. Thus one may assume that d is the "hidden" variable in the recursive form and it is related to other variables. Actually, the value of $x(t+1)$ is only estimated by a finite number of past data because of larger time lag d, smaller weighting the data has. In the following, we will analyze the weighting of the past data at different time lag. Also we will indicate how the internal memory of the model is built.

5.2 Coefficients Analysis

Let's consider the unfolded equation (5.7). This model expresses future values of the time series as a linear combination of past values. Once the coefficients have been estimated, the number of past data which contribute to the data at the next time step will be determined. We call the number of past data needed *embedding dimension.* According to Equation (5.7), $\mu_n(d)$ are the weights of the past data. And according to Equation (5.6), we only give a value to the variable p. The coefficients $\mu_n(d)$ are then estimated.

5.2.1 Analysis of the Coefficients Set, ν_n

The coefficients ν_n, $\nu_n = [\nu_1, \nu_2, \cdots, \nu_p]$. are defined by Equation (5.6). They are determined by p and time length t. Since p is a positive integer number, $|\frac{p}{p+1}| < 1$ and $|\cos(\cdot)| < 1$, thus

$$|\nu_n| = |\frac{p}{p+1} \times \cos(\omega_n t + \omega_n')| < 1$$

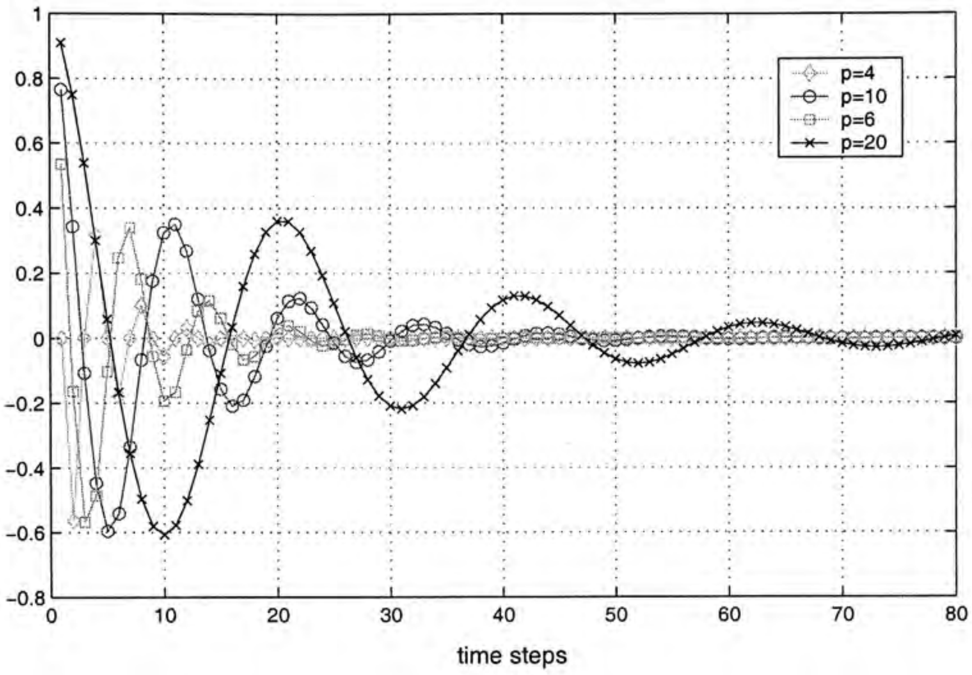


Figure 5.1: The trends of ν with the increasing time steps. Different curves present different trends with various p.

Since the absolute value of $|\nu_n|$ is less than unity, the value of $h_n(0)$ exponentially decays to zero. Figure (5.1) shows how fast the coefficients diminish as the time increases. $h_n(0)$ will have no effect on the prediction value $x(t+1)$ when t is large enough. And the decaying rate depends upon the selected value of p. With the larger p, the effect of $h_n(0)$ will last for a longer time.

In Equation (5.7), the coefficients $\mu_n(d)$ control the weights of inputs at different previous time steps. In the following, we analyze this set of coefficients and suggest a way estimate the parameters.

5.2.2 Analysis of the Coefficients Set, $\mu_n(d)$

The coefficients $\mu_n(d)$, $\mu_n(d) = [\mu_1(d), \mu_2(d), \cdots, \mu_p(d)]$, play an important role in the prediction model because they are the weights of $x(t)$ and control the internal memory of the model. It is defined by

$$\mu_n(d) = \frac{p^d}{(p+1)^{d+1}} \cos(\omega_n d + \omega_n') \tag{5.8}$$

The values of $\mu_n(d)$ are determined by p, and vary with different time-lag d. Both d and p are positive integers, thus we have

$$|\mu_n(d)| = |\frac{p^d}{(p+1)^{d+1}} \cos(\omega_n d + \omega_n')| < 1 \tag{5.9}$$

With time lag d increases, $x(t-d)$ decays exponentially to zero due to its coefficient μ_n becomes extremely small. It implicates after d time steps, the value of $x(t-d)$ will have no contribution to the calculation of the value at next time step, $x(t+1)$, according to Equation (5.7).

Figure (5.2) illustrates the behavior of the coefficient $\mu_1(d)$ as the time lag d increase from 1 to 80. It is clearly shown that the latest model input has the largest contribution to the prediction. And the coefficients of x drop gradually from the first time lag onwards.

This figure also suggests the relationship between the value of p and time-lag d. We define the maximum value of time-lag as M. M is related to the value of p. Different curves in the figure show the decay rates of coefficient with different values of p. With $p = 16$, the model contains more past inputs than that with $p = 2$, which lost the memory of its initial input after roughly $d = 10$ time steps. It suggests larger p achieves larger time delay and the model memorize more past states. We define the maximum time-lag the model can memorize as M. Here M is related to the value of p. If we set

$$|\mu_n(d)| \geq \varepsilon$$

where the value of p is fixed and ε is an arbitrary small value and $d = 1, 2, \cdots$, M is the maximum number of time delays that satisfies the above condition. M represents the "hidden" order or the embedding dimension of the model. And M also implies how many past values can be saved in the model. In our proposed model, once the coefficients of different inputs have

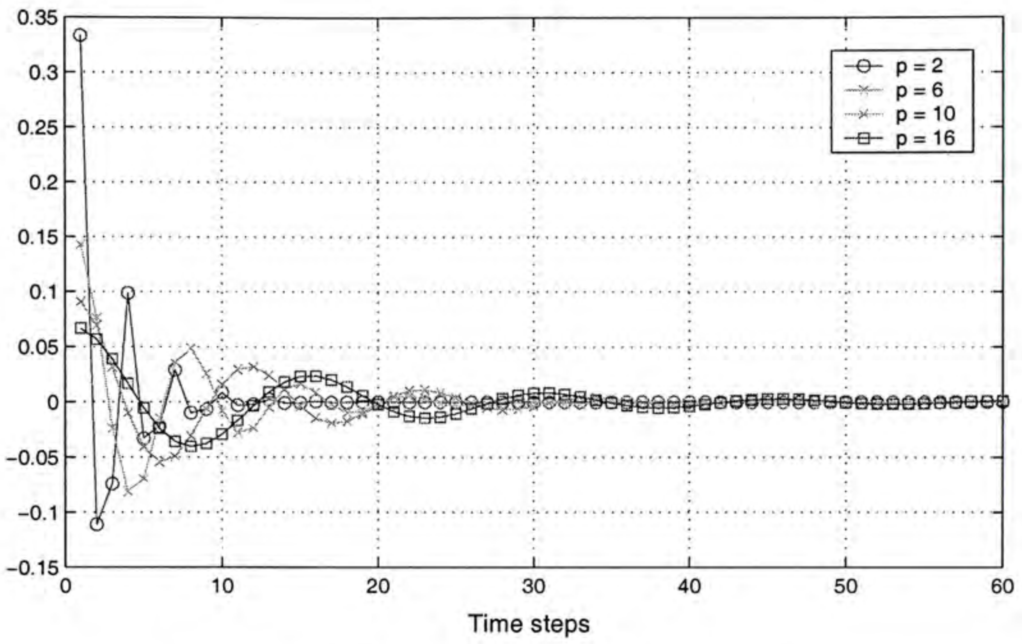


Figure 5.2: Coefficients changing with time

been defined, the hidden order M is then detected. Thus, unlike general AR model, we need not determine the number of model order.

In the above section, by analyzing the unfolded equation (5.7), we implied the relationship between the selection of p and the internal memory of the model.

However, finding a proper p is a tough task. As we know, if a linear model is used to solve the time series prediction problem, the determination of model parameters is heavily dependent on the nature of the time series. As we have analyzed before, the value of p determines how much memory the model has. That is, p decides how many past series data will be used to reconstruct the time series for next time step prediction. And, p is related to the number of hidden units in *ForeNet*.

In the following section, we will demonstrate some experiments with different p.

5.3 Experiments of *ForeNet* Initialization

5.3.1 Objective and Experiment Setting

In this section, some investigations of the proposed prediction model are reported. We tested the performance of the model on some well-known time series, namely the standard sunspots, Mackey-Glass series and several series generated by chaotic processes. The main goals are:

- to assess the predictive ability of proposed prediction equation,
- to infer some "hidden" properties of the proposed model; to observe how much information of the past can be encoded by the prediction equation, and how the embedding dimension can be inferred regarding the underlying process.

For these tasks, we present single-step ahead prediction, i.e. we predict one step into the future given real (measured) values as input. The whole time series are divided into two parts: training set and testing set.[1]

Performance Measure

A measure of fit is given by the *Root Mean-squared Error(RMSE)* of the testing dataset, i.e.

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(x(t) - \hat{x}(t))^2} \tag{5.10}$$

where $x(t)$ is the true value of the sequence, $\hat{x}(t)$ is the prediction. N is the number of data points in the testing set. The prediction accuracy of naive predictor is defined as

[1]Training and testing sets mentioned in this chapter are different to those in NNs training. Here since *ForeNet* hasn't been trained by the learning algorithm, the training set is not used for adjustment of parameters in the model. Using testing set in this chapter is only for purpose of comparison with the performance of *ForeNet* with training in the later chapter.

$$RMSE^{naive} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(x(t) - x(t-1))^2} \tag{5.11}$$

where where $x(t)$ is the true value of the sequence, and $x(t-1)$ represents the prediction.

We used the Root Mean Square Error to compare the results. Also, we used another common method for examining prediction results, that is to plot the predicted curve along with the correct curve. There are usually two main errors which can be found from the plotted curves, amplitude error and phase error. Amplitude error shows that the amplitude of the predicted curve is smaller or larger than the amplitude of the true curve. Most of the phase errors are lagging phase errors, i.e. the predicted curve lags the actual curve. Phase errors may be due to frequency errors. These two types of error cannot be reflected individually in RMSE.

Prediction Steps

As we have mentioned, since only one variable, i.e.p exists in the recursive prediction equation (4.4), it is much convenient to perform predictions with Equation (4.4) than with the unfolding form (5.7). There are several steps involved:

- choose the number of hidden units, p, which is a positive integer value, to get the value of α_n and β_n.
- choose the initial value for h_n. Observed from the unfolding form, the value of h_n decays exponentially as the time increases. So $h_n(0)$ is set randomly between -1 and 1.
- perform the Equation (4.4),

- input the current time series value, calculate the current h:

$$h_n(t) = h_n(t-1)\alpha_n + x(t)\beta_n$$

- compute the predicted next time step value:

$$g(t+1) = \sum_{n=1}^{p} h_n(t)\gamma_n$$
$$\hat{x}(t+1) = 2 \times (g(t+1))^{re}$$

- update γ_n.
- Repeat till whole training series are processed.

• use the fixed parameters to perform prediction on testing series.

5.3.2 Prediction of Sunspot Series

Sunspots are dark blotches on the sun. They were first observed around 1610, and yearly averages have been recorded since 1700. The series is shown in Figure (5.3). The time between maxima ranges from 7 to 15 years and is 11 years on average. The exact underlying mechanism for sunspot appearances has not been known. The sunspot series has served as a benchmark for time series prediction problems in the statistical and connectionist literatures.

In the experiment, we normalize the data to be in the range $[0, 1.0]$. The whole dataset is tabulated from 1700 to 1979 and is partitioned into a training set from 1700 to 1920 and a testing set from 1921 to 1979. Various p are provided to predict sunspot behavior.

Results

For purpose of illustration, Figure (5.4) shows the testing error as the value of p increases. When $p = 3$, the model is the "best" model[2] for prediction.

[2]When we talk about a best model, we will mean the model that has the least root mean square error in its one-step-ahead forecast. This is a convenient but basically arbitrary criterion.

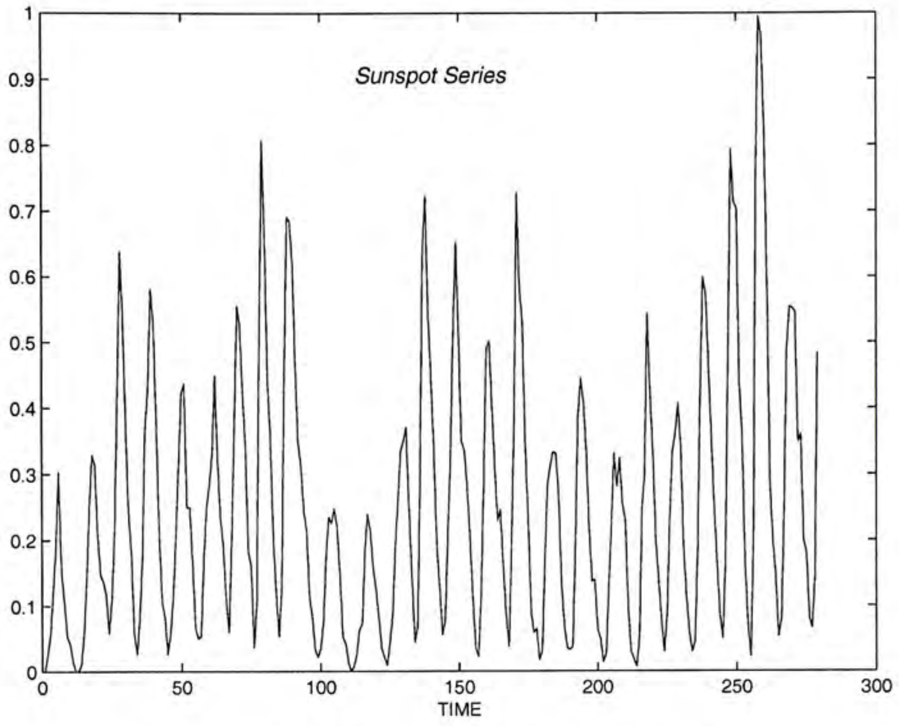


Figure 5.3: Portion of Sunspot series

Method	p=3	p=4	p=6	p=10
proposed method	0.0874	0.1061	0.1391	0.3279
naive predictor	0.1116	0.1116	0.1116	0.1116

Table 5.1: Root mean square error on sunspot series prediction

The training and testing results with $p = 3$ are indicated by the upper and lower panels respectively in Figure (5.5). The one-step ahead prediction root mean square errors (RMSE) are shown in Table (5.1). In the table, we also compare the proposed model with the naive predictor.

When $p = 3$ and $p = 4$ are chosen, the model outperforms the naive predictor according to errors in Table (5.1). In Figure (5.5) we illustrate the prediction curve using the best model with $p = 3$. Compared with the true curve, the predicted curve did well in amplitude prediction but lagged the actual curve.

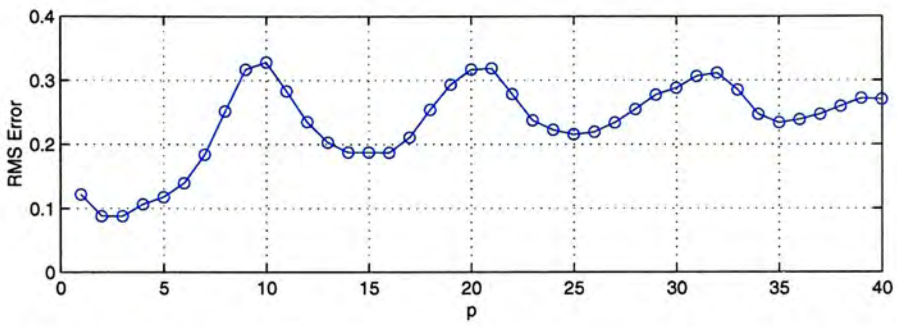


Figure 5.4: Illustration of the testing RMS error on sunspot series using different values of p

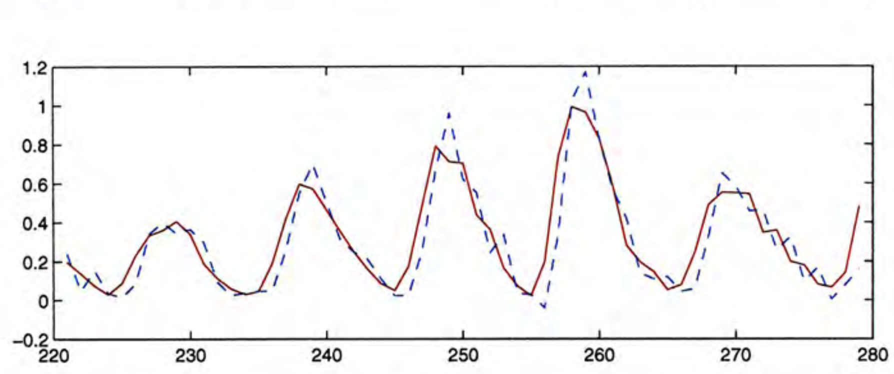


Figure 5.5: Prediction performance of the proposed model. The solid line represents the true series and the dotted line is the model output.

Hidden memory analysis of sunspot data

To characterize some essential properties underlying the data, Weigend et al. varied the input delay vector length of a standard MLP and observed the resulting performance of sunspot prediction [71]. They estimated the dimension of the recursion around 12 and pointed out that for the sunspot data, three degrees of freedom suffice to characterize a point on the solution manifold of the underlying, unknown system. Similar results are also found in [2]. As to our experimental results, we found that using 3 Fourier coefficients to reconstruct original time series recursively can achieve the optimal solution in terms of prediction error on testing data set.

In our model, we build the hidden state vector $h(t)$ during iterations of the equation based on the previous input values $x(t)$. When the iteration commences, the internal state of the model $h(0)$ is randomly set, so the memory of past observations are gradually built up during the iterations. When reaching the enough iterations, i.e. the internal memory contains a full representation of the previous data to do prediction, the model becomes convergence. And at the moment, the number of iterations needed is considered as the embedding dimension (number of past values). For sunspot series, we plot the training curve in Figure (5.6). The upper panel shows the true output and model output. The lower panel is errors when time is from the beginning to the last training data. We randomly initialize the value of h between -1 and 1. Note the transient exists at the beginning of training and its effect will become negligible as is the case in the limit in Equation (5.7), and the prediction errors decay as the network becomes the steady-state mode. It is found from two graphs that after 11 time steps the model approaches stable state. It also infers that 11 past data are enough to build the effective memory in the system to predict the series value at the next time step. And in fact, 11 is roughly

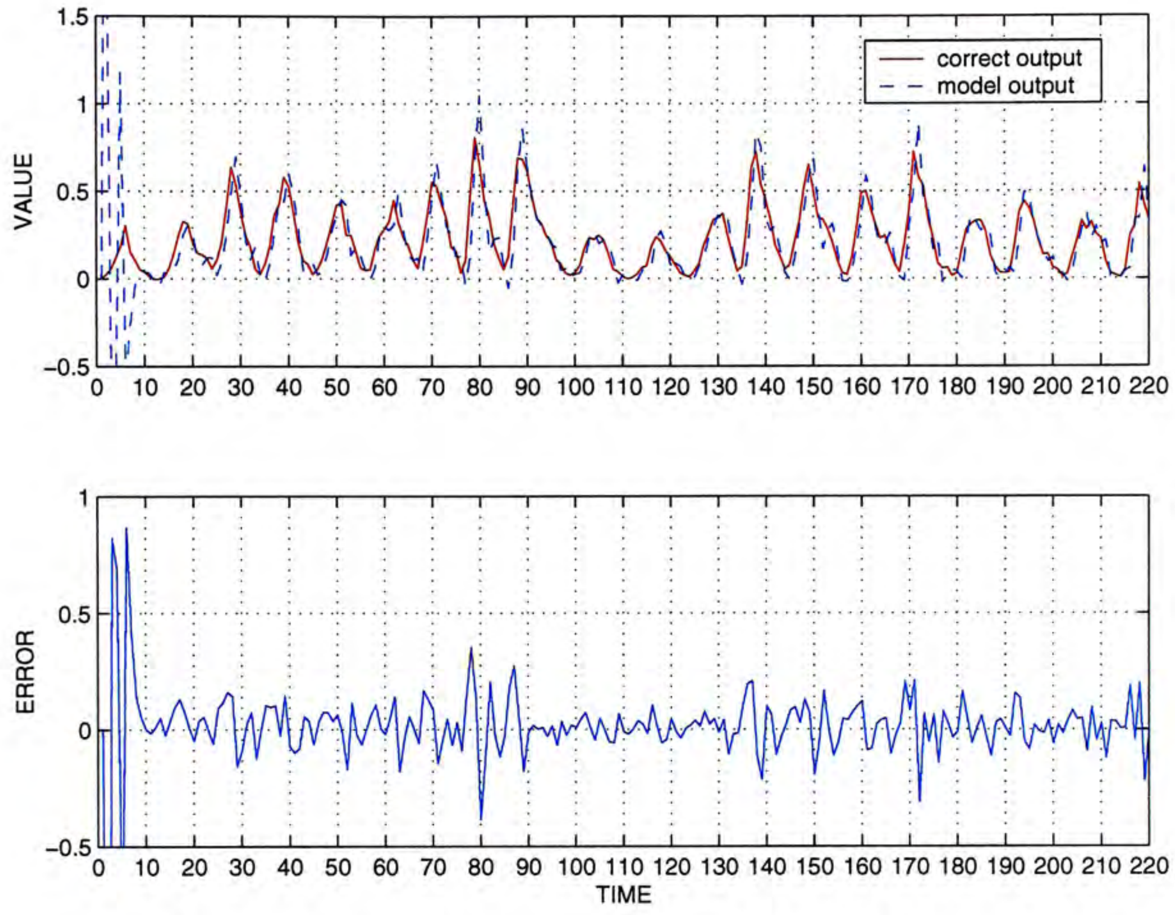


Figure 5.6: Illustration of transient when starting training iterations

the period of sunspot series.

5.3.3 Prediction of Mackey-Glass Series

Mackey-Glass series

Mackey-Glass chaotic time series is generated by a delay differential equation [Mackey and Glass, 1977]

$$\frac{\partial x(t)}{\partial t} = -bx(t) + a\frac{x(t-\tau)}{1+x(t-\tau)^{10}} \tag{5.12}$$

The parameter t is the time variable, x is a function of t, and a, b and τ are constants. We set $a = 0.2$ and $b = 0.1$. Different values of τ produce various degrees of chaos. We use the delay $\tau = 17$, which is just beyond the onset of chaos. Solving Equation (5.12) yield the time series $x(t)$. Mackey-Glass series is chosen to show model performance with high dimensional systems,

Method	p=3	p=4	p=6	p=10
proposed method	0.0474	0.0128	0.0339	0.0879
naive predictor	0.0228	0.0228	0.0228	0.0228

Table 5.2: Root mean square error on Mackey-Glass data prediction

and it possesses many dynamic properties. The prediction equation is used for one-step ahead prediction. For comparison, $\tau = 30$ is also used to generate the series for prediction. The whole dataset consists of 1000 data points. The first 600 points are used for training, and the remaining 400 are testing data.

Results and Analysis

Figure (5.7) demonstrates that how the different values of p effect the model prediction performance. The left pane illustrates the results with delay $\tau = 17$ and the right one is on the series generated by $\tau = 30$. For two series, both using $p = 4$ built the optimal prediction model. And the figures implicate that to predict two series with different degrees of chaos, the model behaviors with various values of p are very similar. When p begins from value 1, the testing error is decreasing till p becomes 4, thereafter, with p increases, the model error also becomes larger. The model uses the same parameter ($p = 4$) achieve the optimal predictions on two Mackey-Glass series. In Figure (5.8), Mackey-Glass time series and the model predicted series are shown. Both are done with $p = 4$. We can see that in this case, not only the magnitude of the series fit well, but the frequency of the extrapolation curve is almost equal to that of the correct curve. In Table (5.2), we also compare the proposed model with the naive predictor. Only using $p = 4$, the model can do better than the naive predictor.

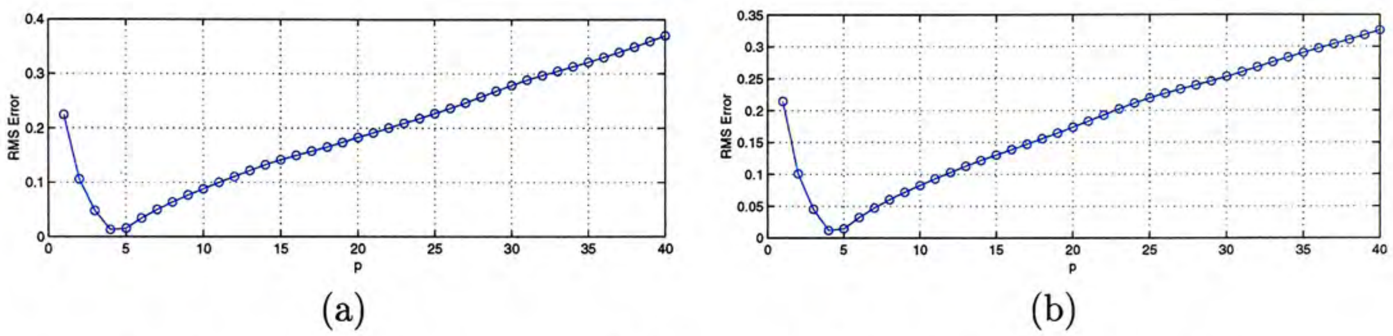


Figure 5.7: Illustration of the testing RMS error on Mackey-Glass series using different value of p. The left pane illustrates the results with delay $\tau = 17$ and the right one is on the series generated by $\tau = 30$.

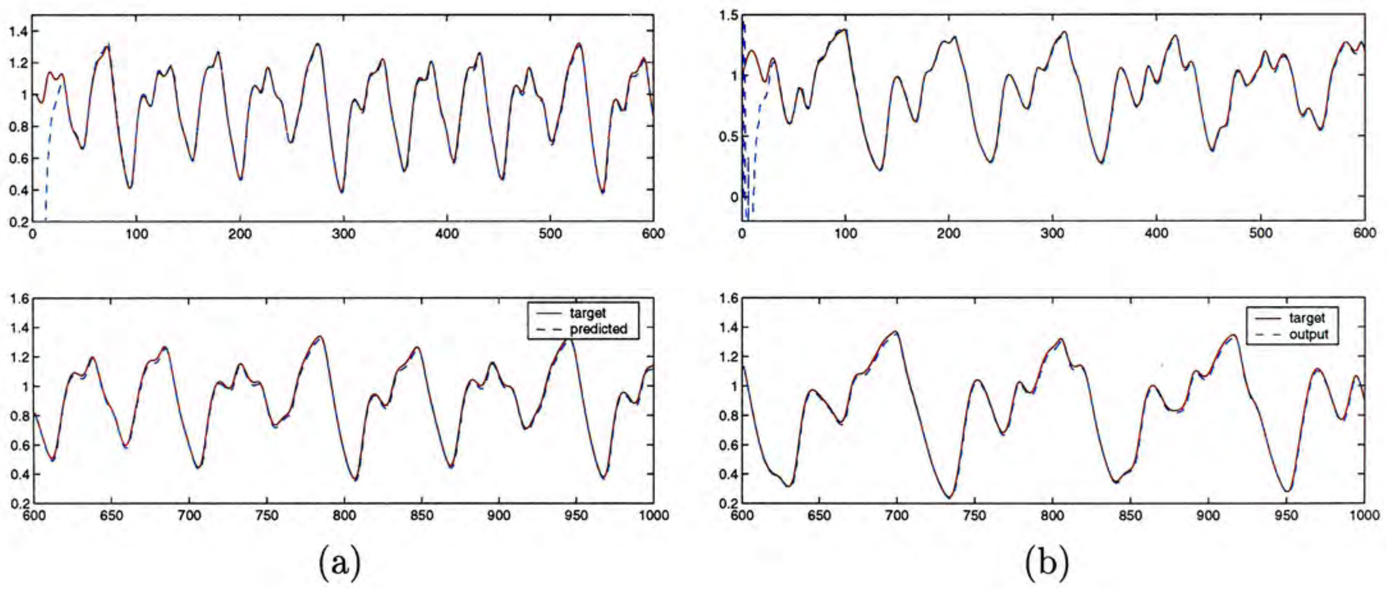


Figure 5.8: Prediction performance of the proposed model. The left pane is Mackey-Glass series with $\tau = 17$ and the right pane shows series with $\tau = 30$. The solid line represents the true series and the dotted line is the model output.

5.3.4 Prediction of Laser Data

Laser data

We also test the Fourier recursive prediction equation on the laser data of the Santa Fe competition[70]. The data set consists of laser intensity collected from a laboratory experiment. Its behavior is chaotic as seen in Figure (5.9). This data set is chosen because

- they are a good example of the complicated behavior that can be seen in a clean, stationary, low-dimensional non-trivial physical system for which the underlying governing equations dynamics are well understood.
- the data set is short; in many fields, such as economics, the data sets may only be a few hundred points long. The data set that was known to have low-dimensional dynamics to use as a test case for analyzing short data sets to help make the task more manageable.
- it is very predictable on the shortest time scales (relatively simple oscillations), but that has global events that are harder to predict.

We adopted total 1500 data points. The first 900 points are for training and the following 600 are for testing.

Results and Analysis

The Figure (5.10) suggests using $p = 1$ can achieve the optimal prediction. The Figure (5.11) illustrates the training and testing results.

The number of time delays (or the order) of the AR model that makes successful predictions provides an upper bound on the minimum embedding dimension [70]. For the laser data, according to the experimental results $p = 1$ is clearly better than other values. Let observe the Figure (5.12), where the

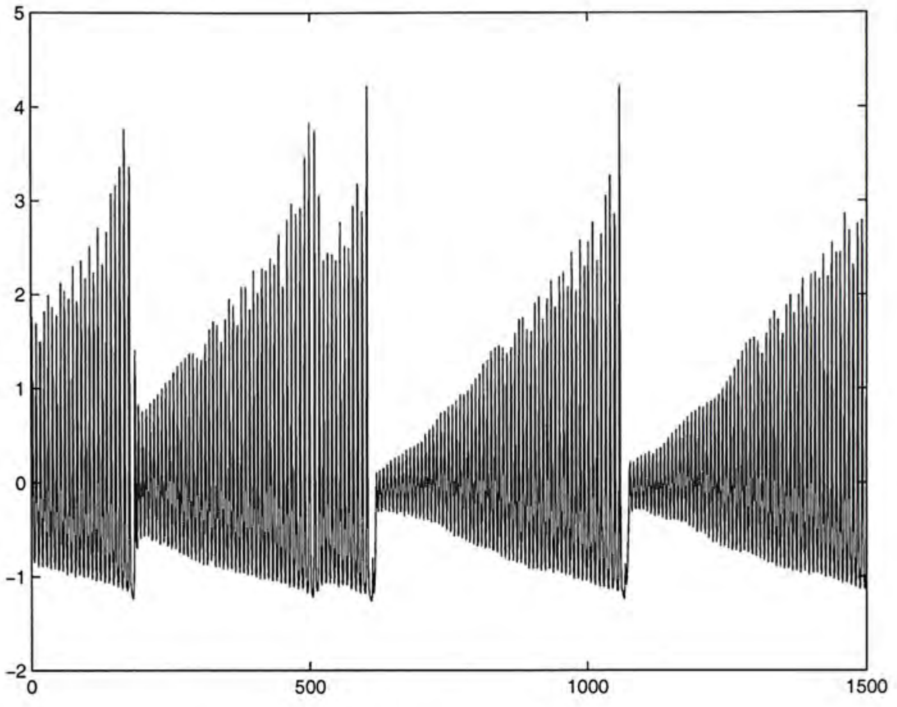


Figure 5.9: Laser Series

	Proposed Model	Naive Predictor
RMSE on testing set	0.4973	0.6664

Table 5.3: Root mean square error on laser series prediction, with $p = 1$

upper graph shows 7 past points can build up the effective memory tȯ do prediction, and the lower graph is the weighting of input data at different time steps when p is set to 1, which also shows after 7 time steps, the effect of input vanishes. Both indicate that when $p = 1$ used in the model, accordingly hidden dimension would be $M = 7$. Let we observe the laser series. It is obvious that there are two "dominant" periods in the series: one is very short but the other is much longer so that it cannot be easily detected using the limited data points. And the *ForeNet* is a local model, so in stead of learning the long trend of time series, the model catches the short period and consider it as the "dominant" period, which is around 7. In [71], weigend showed when the number of delays $d = 6$ the model had the lowest out-of-sample errors.

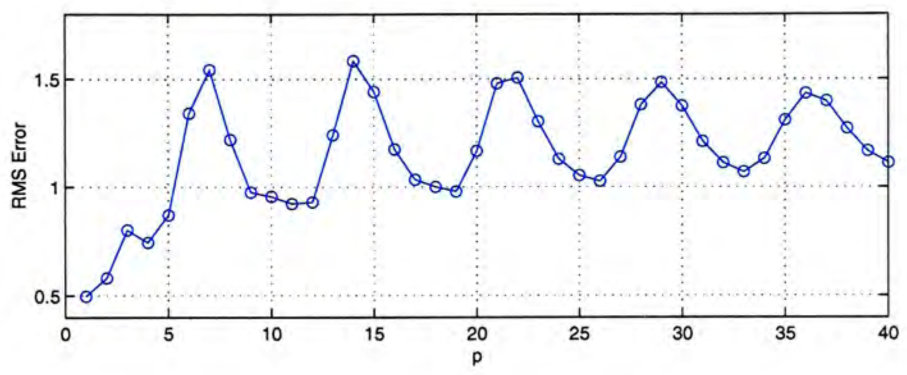


Figure 5.10: Illustration of the testing RMS error on laser series using different values of p

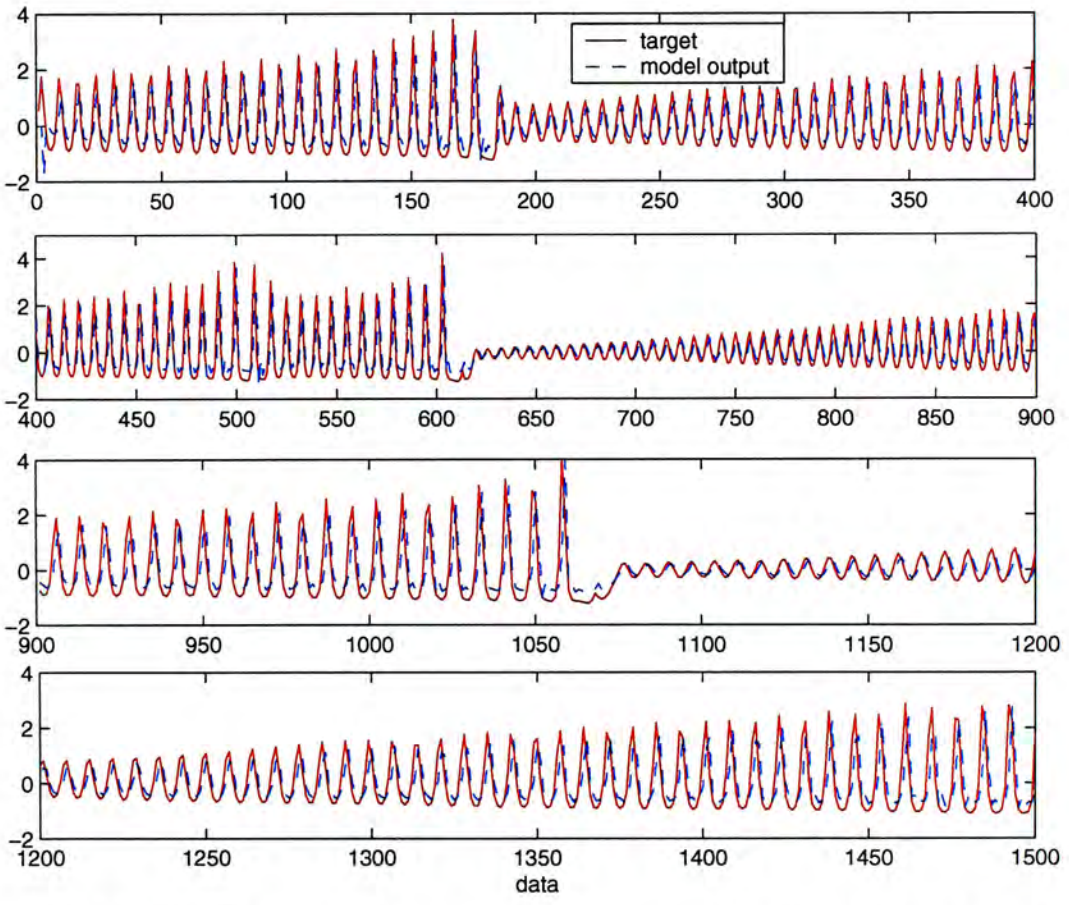


Figure 5.11: Prediction performance of the proposed model using $p = 1$.

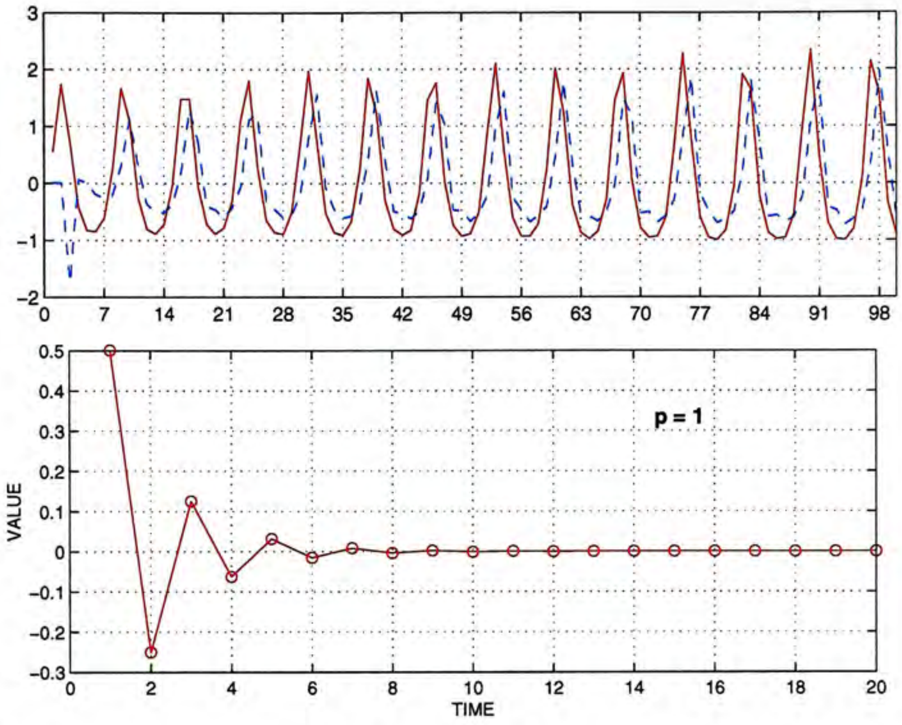


Figure 5.12: The first figure shows a portion of training series, the correct curve (solid line) and the model output (dotted line). The second figure is the weighting of model inputs with time increases.

5.3.5 Three More Series

Series Description

Some more experiments are demonstrated in this part. Consider the following time series:

- Time-varying sinusoidal series (Figure (5.13)), which is generated from the equation:

$$y(t) = sin(w_t t) \qquad t = 1, 2, ...n$$

where $w_t = 0.0081t$ is the frequency varying with time. n is set to 600. The first 300 data is specified as training data, and the following 300 data is used as testing data.

- The sinusoidal series with noise generated from the equation:

$$y(t) = \sin(wt) + \sigma$$

where a uniformly distributed random value σ in the interval $[-0.5, +0.5]$ is added to the sin function at each time step. And $w = 1$, t is from 1 to 200. The first 100 points were used as training data. The next 100 points were testing data. Figure (5.13) shows the series.

- Computer-generated series. It is also from the Santa Fe competition[70]. The series was generated by numerically integrating the equations and has nine degrees of freedom. From the data file, we choose 1000 data, using 600 points for training and 400 for testing.

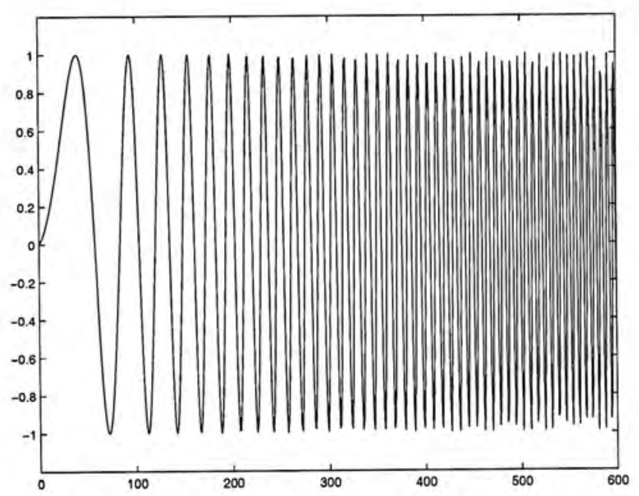
Results and Analysis

The model performance depends on the various values of p, as shown in Figure (5.14). The model did best when using $p = 2$ to predict noisy sinusoidal series; using $p = 3$ to predict the time-varying series and using $p = 4$ to predict computer-generated series. It is noted that the series with shorter hidden period have smaller p to do prediction, which also means that the model need the small number of embedding dimension (the number of past values) to compute value at next time step.

Table (5.4) is the comparisons between the proposed model (with "optimal" p) and the naive predictor. And Figure (5.15) and Figure (5.16) illustrate the training and testing curves respectively. We can find that the model can detect the changes of the frequencies in series well according to the prediction results on time-varying sinusoidal series shown in Figure (5.15). And it can also do well in the noisy series prediction.

	time varying sinusoidal series	noisy sinusoidal series	computer generated series
proposed method	0.1951	0.2931	0.0501
value of p	3	2	4
naive predictor	0.342	0.4795	0.0635

Table 5.4: Root mean square error on time series predictions


(a) time-varying sinusoidal series

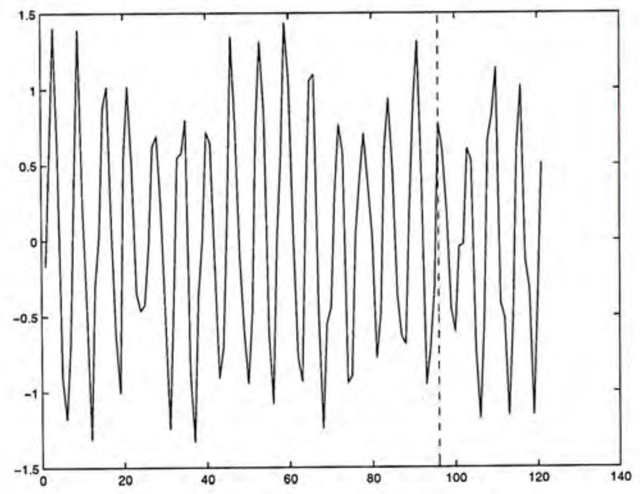
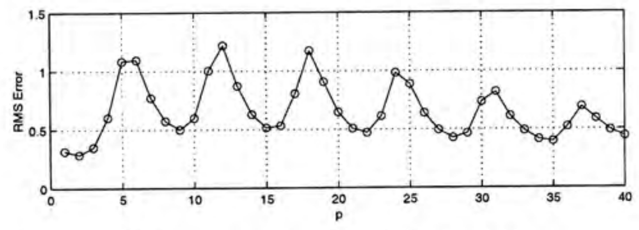
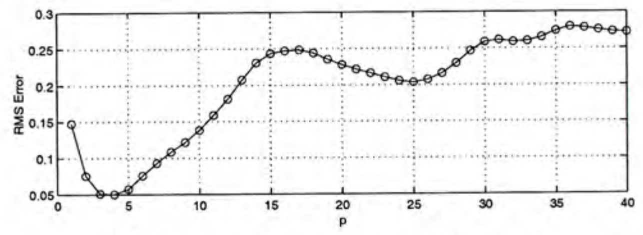

(b) noisy sinusoidal series

Figure 5.13: Two sinusoidal series


(a)noisy sinusoidal series


(b)computer-generated seris

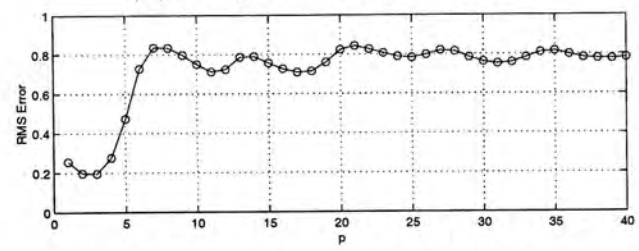

(c)time-varying sinusoidal series

Figure 5.14: Illustration of the testing RMS error using different values of p.

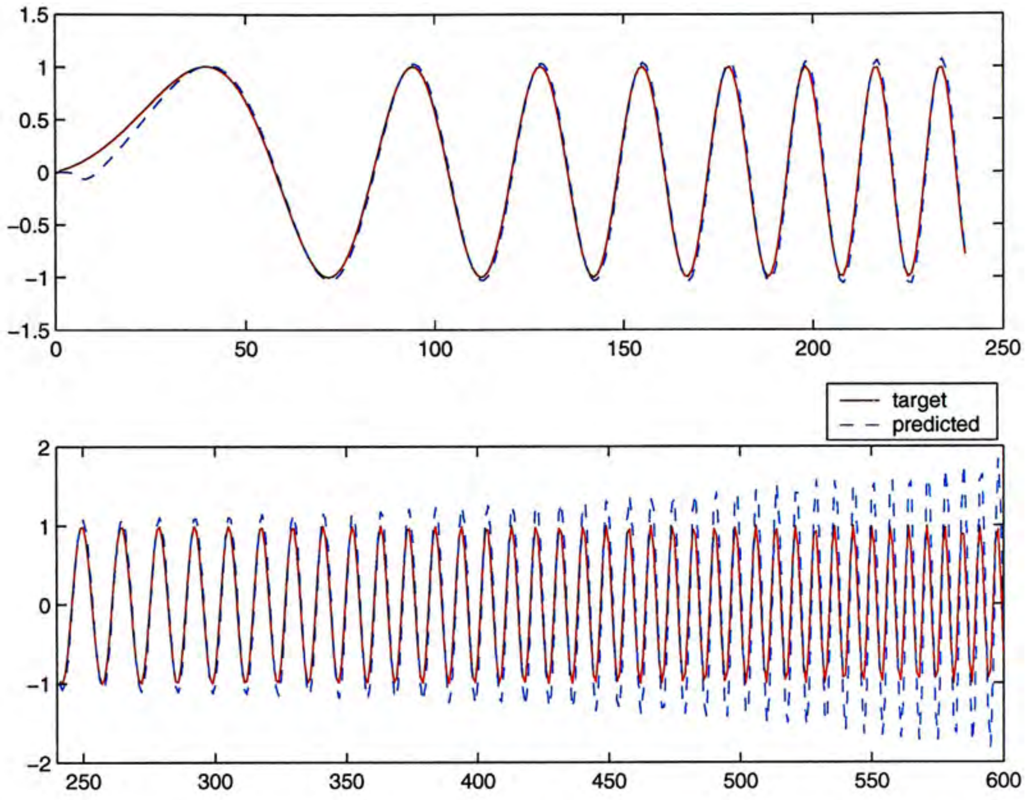


Figure 5.15: Prediction performance of the proposed model on time-varying sinusoidal series. The solid line represents the true series and the dotted line is the model output.

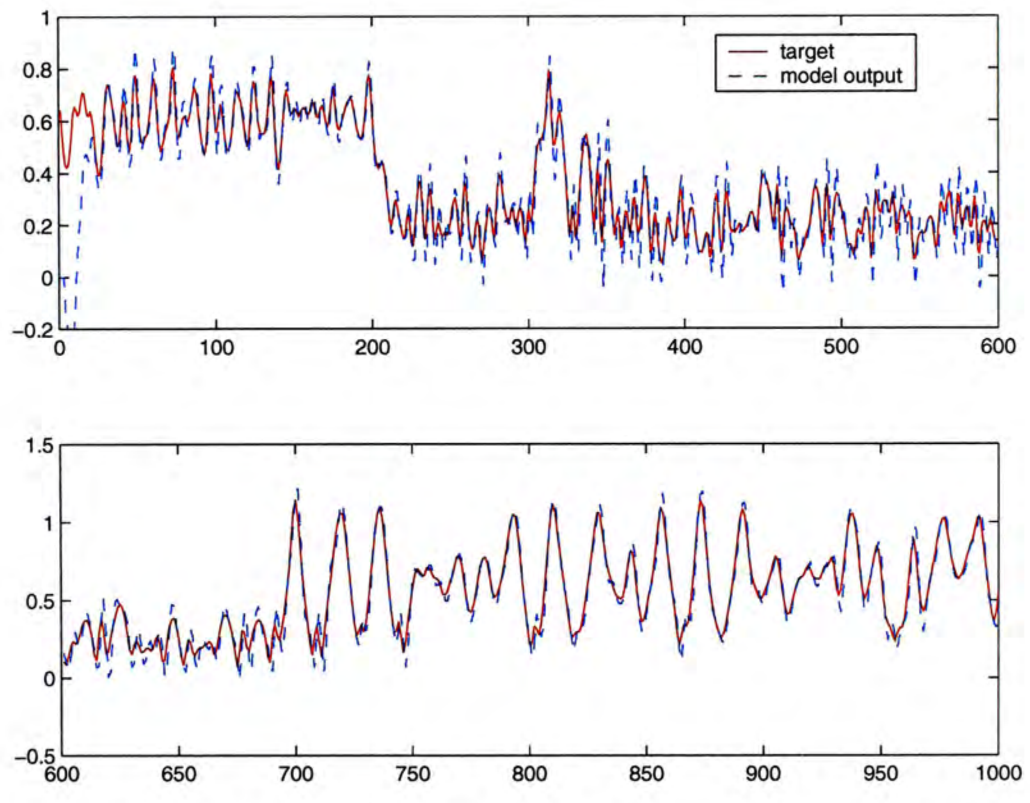


Figure 5.16: Prediction performance of the proposed model on computer-generated series. The solid line represents the true series and the dotted line is the model output.

5.4 Some Implications on the Proposed Initialization Method

We would like to point out some more implications on our proposed model based on the derivation of the prediction method and the experimental results.

- It is mentioned that the series with shorter hidden period have smaller p to do prediction. It is not surprising that with smaller value of p the model can do better on time series with shorter "period". It may be because of the following relationships:
  - The smaller p in the model is related to the relative less embedding dimension of the time series;
  - The embedding dimension of a given time series has somehow correspondence to the "dominant period" of the time series. So shorter period is related to smaller embedding dimension which is needed to reconstruct the given time series.
- Most of the phase errors we found are lagging phase errors; the predicted curve lags the actual curve. Phase errors may be due to frequency errors. In the proposed model (Equation (4.4)), the intermediate states $h(t)$ are complex values, which include both magnitude and phase information. Therefore, one can consider when the prediction is processed, the amplitude error is introduced by estimations of the magnitude of variables, which are related to the selection of p. And the phase error is due to the estimation on the phase of variables. As we known, to reconstruct a time series, it is important that the phases be symmetrized in such a way that the inverse Fourier transform is real and the power at each frequency is unaffected. However, in the most cases, using fourier transform to recover original time series cannot satisfy such vital condition since only

some frequencies can be used. But in our proposed method, we have shown that some approximations were made to remedy it. Thus, one of advantages of the proposed complex-valued model is that the changed phase information in time series may be detected. This can be verified by the above experiments. Especially when observing Figure (5.15), the model can catch the phase changes with time. Thus the predicted curve has small phase error although with obvious amplitude error.

- The proposed model is a local linear model, which is trying to divide the data set to smaller sets and each is modelled by Fourier analysis. The k nearest past data points are decomposed into several frequency coefficients, which are then reconstructed to forecast the value at next time step. Such local linear model is able to capture the geometry well. So its advantage is the ability of adhearing to the local shape of an arbitrary surface; the corresponding disadvantage is that it may be insufficient to understand the global characteristics of the underlying system.

- The prediction performance of the proposed model is dependent on both the coefficients and the nature of given time series, including the length of the data set, the sampling rate, and the embedding dimension. From all performed experiments, we found that the proposed model usually do worse predictions on those time series whose dominant frequencies are high in terms of the phase error and the amplitude error. And usually the small p is selected to predict these time series, including laser series, noisy sinusoidal series and sunspot series. Such phenomena may be caused by the approximations introduced during model derivation, that is, we assume the "important" parts of the series are with low frequencies. In linear systems, such assumption is often safe. However in nonlinear systems, for example laser series, the important part of the signal often cover the entire spectrum (shown in Figure (5.17)). Therefore, such assumption makes some information of signal lost. It is also one shortcoming of our approach.

- If *ForeNet* hasn't been trained, the internal memory is nonadaptive, and it is not appropriate for most time series. In the later chapter, we will train *ForeNet* to make its internal memory adaptive with time.

- The first several data points in time series is used to initial an internal memory of the model. We have realized that the internal memory plays an important role in time series prediction. We can control the model's memory by setting different value of p. However, it is difficult to exactly decide how many training points should be chosen, as the length of the transient will depend on both the nature of time series and the model parameters. When we apply *ForeNet* to a test set in the next chapter, we will not set the internal hidden state to zero before starting the iterations. That is, we always choose the test series to *immediately* follow the training series. When estimating the generalization error, iterations are initiated on the proceeding training series in order that the internal memory of the recurrent networks has been properly built.

- The last implication is the weights involved in the recurrent network are no longer real but complex. Traditional BPTT or RTRL learning algorithms are not applicable. In the later chapter, we will propose a complex-valued recurrent learning algorithm to train *ForeNet*.

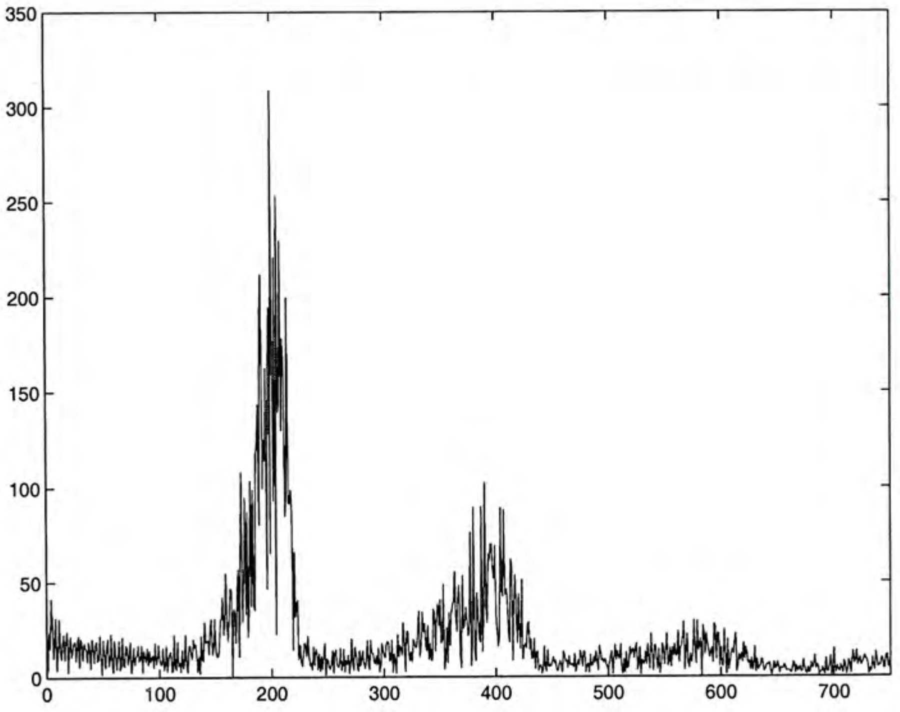


Figure 5.17: Discrete Fourier Transform of the laser series

Chapter 6

ForeNet: Learning Algorithms

We have proposed a recurrent network structure whose parameters are complex values and constructed to perform the predict task. In the previous chapter, we analyse the property of Fourier Recursive model, i.e. *ForeNet* model without training. This chapter shows the algorithms for adjusting the parameters of *ForeNet*. The learning is based on a sample of time series input/output pairs $(x(t), x(t+1))$ for $t = 1, 2, ..., N-1$. The most widely used algorithm for training Recurrent Networks is the Real Time Recurrent Learning (RTRL) algorithm proposed by Williams and Zipser [75] as described in Chapter 2. However, RTRL is not suitable in our case to process complex parameters because it assumes all parameters are real-valued. Thus we are showing the learning algorithm for the complex-valued recurrent network.

There are two steps in our learning process. The first one is the network initialization which has been described in Chapter 4 5. The second is the training of the network using the modified complex RTRL algorithm. We introduce here two basic weight-update methods, bath-mode learning and online learning.

6.1 Complex Real Time Recurrent Learning (CRTRL)

The Complex RTRL algorithm was proposed in [38] to train Fully Recurrent Networks. In this section, CRTRL is modified to suit our model. Real and complex numbers are denoted by lower and upper case letters. The real and imaginary parts of a complex number W is denoted by w_R and w_I respectively. Consider a RRN with p inputs, n hidden units and m output units. We use P, N and M to denote the set of input units, the set of hidden units and the set of output units respectively. All units are interconnected as described above.

The cost function is

$$e(t) = \frac{1}{2} \sum_{u_k \in M} e_k^2(t) \tag{6.1}$$

where

$$e_k(t) = D_k(t) - V_k(t) \tag{6.2}$$

and

$$V_k(t) = 2 \times z_{Rk}(t) \tag{6.3}$$

$e_k(t)$ is the error of the k^{th} output unit at time t. $D_k(t)$ denotes the k^{th} target and $Z_k(t)$ is the k^{th} network output. Based on the prediction method of proposed semi-AR model, when we calculate the network error, we adopt two times of the real part of $Z_k(t)$ as network output.

Forward Phase: For $j = 1, ..., N$

$$Y_j(t+1) = f(\sum_{u_i \in N} W_{ij} Y_j(t) + \sum_{u_k \in P} W_{kj} X_k(t+1)) \tag{6.4}$$

$$Z_j(t+1) = \sum_{u_l \in M} W_{jl} Y_j(t+1) \tag{6.5}$$

where

$Y_j(t+1)$: output of hidden unit j at time $t+1$;

W_{ij} : weights between hidden units;

W_{kj} : weights from input units to hidden units;

W_{jl} : weights from output units to hidden units;

$X_k(t+1)$: k^{th} input at time $t+1$.

f : denotes the linear activation function. That is, $f(X) = X$.

Expressing this equation with the real and imaginary parts of W_{ij}, W_{kj}, Y_j and X_k

$$\begin{aligned} y_{Rj}(t+1) = f(&\sum_{u_i \in N} (w_{Rij} y_{Rj}(t) - w_{Iij} y_{Ij}(t)) \\ &+ \sum_{u_k \in P} (w_{Rkj} x_{Rk}(t+1) - w_{Ikj} x_{Ik}(t+1))) \end{aligned} \tag{6.6}$$

Learning Phase: Since $e(t)$ is a real-valued function, we compute its gradient with respect to W_{ij}

$$\frac{\partial e(t)}{\partial W_{ij}} = \frac{\partial e(t)}{\partial w_{Rij}} + j \frac{\partial e(t)}{\partial w_{Iij}} \tag{6.7}$$

Now, differentiating Equation (6.1), we get:

$$\begin{aligned} \frac{\partial e(t)}{\partial w_{Rij}} &= -2 \sum_{u_k \in N \cup M} e_k(t) \frac{\partial z_{Rk}(t)}{\partial w_{Rij}} \\ \frac{\partial e(t)}{\partial w_{Iij}} &= -2 \sum_{u_k \in N \cup M} e_k(t) \frac{\partial z_{Rk}(t)}{\partial w_{Iij}} \end{aligned} \tag{6.8}$$

and differentiating (6.5),

$$\frac{\partial z_{Rk}(t+1)}{\partial w_{Rij}} = f'(y_{Rk}(t+1)) \frac{\partial y_{Rk}(t+1)}{\partial w_{Rij}} \tag{6.9}$$

The weights between the hidden layer and the output layer are non-recurrent in our model and can be updated easily by error backpropagation.

$$\frac{\partial z_{Rk}(t+1)}{\partial w_{Rij}} = f'(y_{Rk}(t+1))z_{Rl} \tag{6.10}$$

where, $u_k \in M$ and $u_l \in N$. The weights to the hidden units are recurrent and we can propagate the error terms one step back from the output layer to the hidden layer and update the weights according to equation

$$\begin{aligned}\frac{\partial z_{Rk}(t+1)}{\partial w_{Rij}} &= f'(y_{Rk}(t+1))(\sum_{u_l \in N}(w_{Rkl}\frac{\partial z_{Rl}(t)}{\partial w_{Rij}} \\ &+ \delta_{ik}x_{Rj}(t))\end{aligned} \tag{6.11}$$

where

$$\delta_{ik} = \begin{cases} 1 & \text{for } u_i = u_k \\ 0 & \text{for } u_i \neq u_k \end{cases} \tag{6.12}$$

In the similar way, we derive the recursive equation for $\frac{\partial z_{Rk}}{\partial w_{Iij}}$.

Finally, the weight update equation becomes

$$\Delta W_{ij}(t) = -\eta\frac{\partial e(t)}{\partial W_{ij}} + \alpha\Delta W_{ij}(t-1) \tag{6.13}$$

where, η is the learning rate, α is the momentum coefficient and $\Delta W_{ij}(t)$ is the change in W_{ij} at time t. The weight changes are calculated at each epoch along the way and take the full change as the time-sum of $\Delta W_{ij}(t)$ at the end of the training sequence.

6.2 Batch-mode Learning

There are two basic weight update methods, batch-mode and on-line learning. In batch mode learning, every pattern p is evaluated to obtain the derivative

$\partial E_p/\partial w$. After all training pattern are inputted into the network, the total derivative can be obtained and only then the weights are updated[56]. Another method, on-line learning, will be described later in this chapter. And all experiments in the next section will be performed by batch-mode method only.

The whole training steps of *ForeNet* are:

(1) The coefficients of Fourier recursive model are adopted to initialize *ForeNet*.

(2) Learning phase

- For every pattern p in the training set,
  - apply a pattern p in the training set,
  - calculate the pattern error E_p and the single-pattern derivatives $\partial E_p/\partial w$.
- Add up all the single-pattern terms to get the total derivative.
- Update the weights according to CRTRL.
- Apply the validation data set, calculate the validation error $e_V(t)$.
- Compare to the validation error of the last training epoch.
  - If $e_V(t) > e_V(t-1) > \cdots > e_V(t-n) >$, stop training.
  - else, repeat the learning phase.

(3) Apply the testing set, use the trained model to perform the prediction.

6.3 Time Complexity

We have shown in Figure (3.1) that the recurrent links exist only in the hidden layer and all the feedback links are self-connections of hidden units. We define the degree of connectivity, q, which is equal to the number of incoming recurrent connections of each hidden unit. In *ForeNet*, q is 1 since only the self-feedback loop exists.

The computational requirements of *ForeNet* are mainly on the need to store and update the p_{ij}^k values ($p_{ij}^k = \frac{\partial z_k(t)}{\partial w_{ij}}$). For each element p_{ij}^k, we go through Equation (6.11) once in each time step and there are q ($q = 1$ in our case) operations to get the updated value in (6.11). In our model, there are total $n(p+q)$ weights in the hidden layer so updating Equation (6.11) in each time step needs the computational complexity of order $O(nq(q+p))$ operations. Considering the other operations, the total computational complexity is order $O(mn+nq(q+p))$, that is order $O(mn+n)$, where m is the number of output units and n is the number of hidden units. The order of fully recurrent network (FRN) derived by Williams and Zipser[75] is $O(N^4)$, where N is equal to the number of non-input units. Compared to FRN, *ForeNet* has much smaller computational complexity.

6.4 Property Analysis and Experimental Results

In the following sections, we show properties of *ForeNet* by some experiments. The architecture of *ForeNet* is a $1-p-1$ network (shown in Figure (3.1)), i.e., the network had one input unit, one output unit and p hidden units, where p can be determined by the proposed Fourier recursive model as described in Chapter 5. The activation function in the network is a linear function. As mentioned, *ForeNet* initialization is not randomly done. It performs based on Fourier recursive model.

In the algorithm, the learning rate η is set to 0.1 and the momentum coefficient α is 0.5. And the maximum learning iterations is set to 500. We used the root mean square error (RMSE) to monitor prediction performance of the network. The whole data set is divided into three parts in sequence: the first is training data set (used for parameter estimation); the second part is validation set, which serves as the stopping criteria of training process to determine the stopping point before overfitting occurs; the last is testing data set. The network would stop training when the largest learning iterations are reached or the error on validation set becomes increase.

Time Series

We perform the prediction tasks on the following time series.

- Mackey-Glass series generated from Equation (5.12). The series includes 1000 data. The first 400 are used for training and the following 600 points are for validation and testing.
- The laser series from the Santa Fe time series competition. The series is illustrated in Figure (5.9). The network is trained to perform a one step ahead prediction. All samples are scaled to zero mean and unit variance. There are 1500 points in laser series. The first 900 data are the training data. After training, we perform the forecasting on the following 600 data.
- Time-varying sinusoidal series (figure (5.13)). The first 300 data is specified as training data, and the following 300 data is used as testing data.
- The sinusoidal series with noise. The first 100 points were used as training data. The next 60 data were used for validation. The remaining 40 points were testing data. Figure (5.13) shows the series.

- Computer-generated series. We use total 1000 data, the 600 points for training and 400 for testing.

All experiments are performed with MATLAB and running under UNIX OS, Solaris 7. We did the experiments for several times and reported their average performance in the following tables.

6.4.1 Efficient initialization:compared with random initialization

Objective

We have proposed a NN initialization method that is based on the Fourier recursive model. In this section, we show the efficiency of such initialization for NN training. With the optimal NN parameters are determined, the initial error is substantially smaller and the number of iterations required to achieve the convergence is also significantly reduced. We perform the prediction tasks using *ForeNet* and for comparison, we also train *ForeNet* with randomly generated complex initial parameters.

Results and Analysis

Table (6.1) and (6.2) show the network performance using the proposed method and random initialization respectively. In the problem of Mackey-Glass prediction, the proposed method generates the good initial parameters, which make the pretty smaller initial training error (0.0205) than that using randomly setting initial values (2.2589). The network convergent only after 13 iterations with better generalization. The learning curves are shown in Figure (6.1)and Figure (6.2). Similar result can be found on laser series forecasting. The proposed initialization has small initial error thus speed up the convergence speed. Figure (6.3) illustrates the different network performance using different initial method. (a) and (b) present the network outputs computed with the proposed initialization, before training and after training respectively. From the graphs, we can see that the network output is very close to the target values even before training has been taken place. Further training only provides small fine tune to the results. (c) shows the corresponding *ForeNet*'s output with random initialization. Although this data set is a non-stationary series, our method is able to deal with the time-varying signals.

From the prediction of noisy sinusoidal series, we find that the the proposed initial method seems get worse performance in terms of initial root mean square error than random initialization. But compared to random method, which take 52 training iterations to convergent, the proposed method only use 18 iterations to achieve much better generalization according to the final RMS error. One may wonder why the model can get the fast convergence speed even with larger initial error. It has to be noted that RMS error measures network performance, but it show no indication of the closeness to the optimal point in the weight space. Thus small initial error does not necessarily imply better initialization. The efficient initialization methods should be those assigning the weights in the region not far away from the minimum. In addition, the region should not be flat if training is carried out by gradient methods. An efficient initialization can prevent the model from getting struck with the initial weights thus help to achieve good testing results.

series	No.of hidden units	Initial training error	Testing RMSE	No. of iterations	Time taken(sec)
Mackey-Glass series	4	0.0205	0.0102	13	63
laser data	4	0.7449	0.5049	19	142
time-varying sine series	3	0.0746	0.1251	103	257
noisy sinusoidal series	4	0.699	0.0729	18	24

Table 6.1: The prediction performance of *ForeNet* with proposed initialization

series	Initial training error	Testing RMS error	No. of iterations	Time taken (sec)
Mackey-Glass series	2.2589	0.0558	198	866
laser data	1.1961	0.4784	202	1324
time-varying sine series	0.9676	0.3203	80	184
noisy sinusoidal series	0.6889	0.4285	52	60

Table 6.2: The prediction performance of *ForeNet* with random initialization

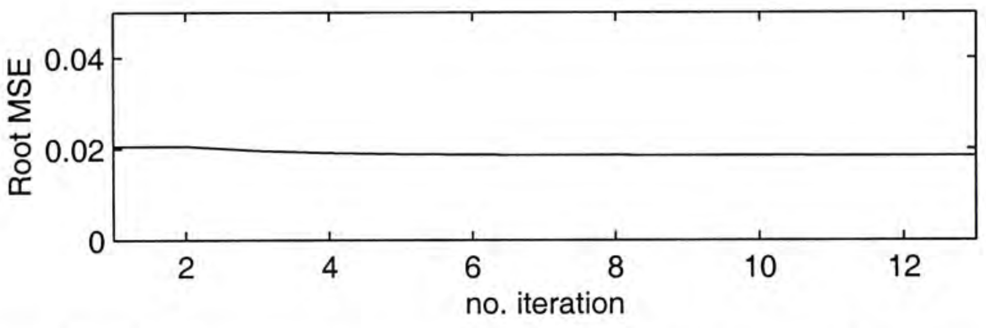


Figure 6.1: The learning curve of *ForeNet* with proposed initialization

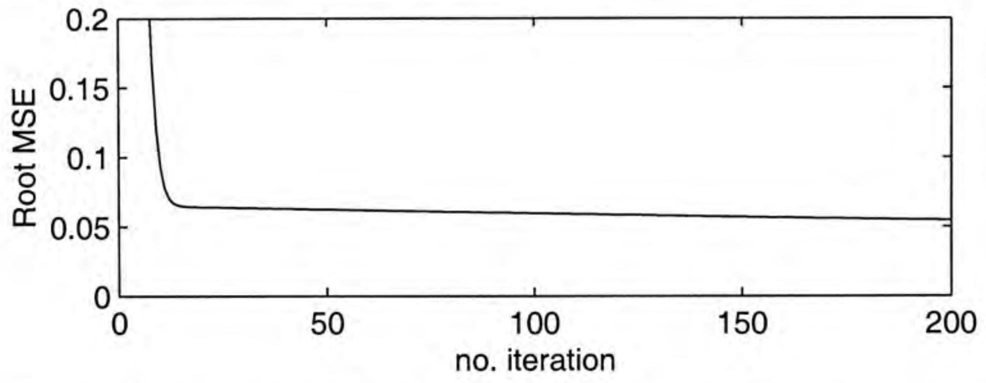
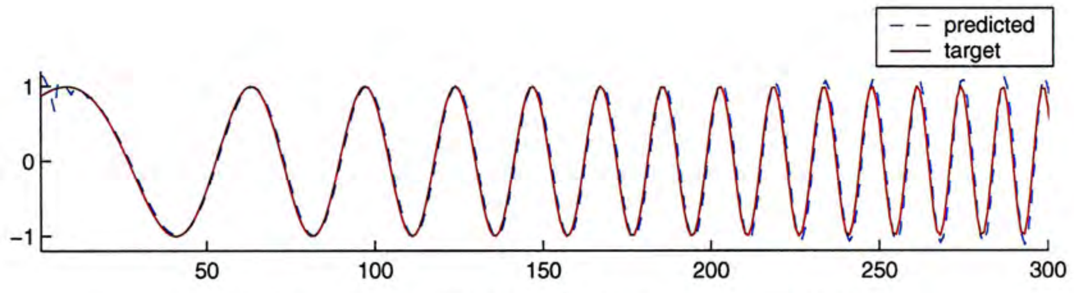
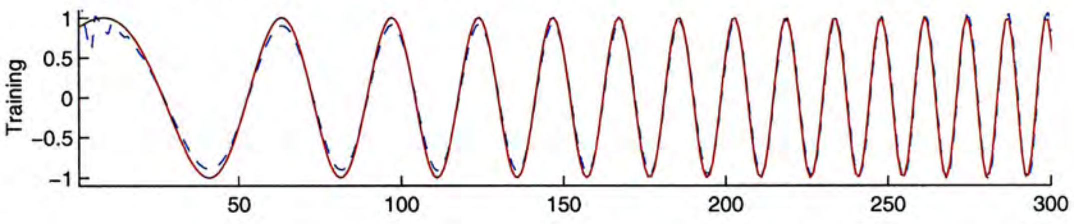


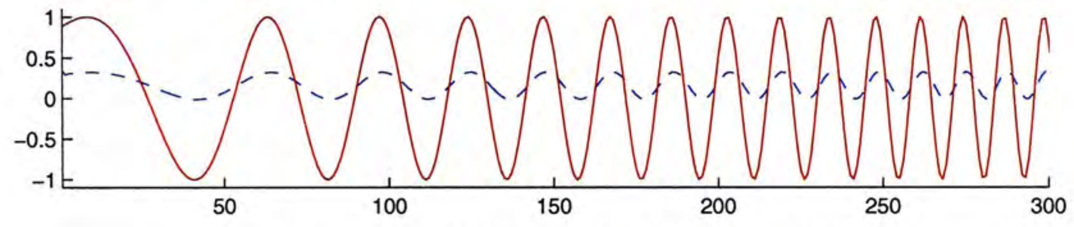
Figure 6.2: learning curve of *ForeNet* with random initialization



(a)with proposed initialization before training



(b)with proposed initialization after training



(c)with random initialization before training

Figure 6.3: Predictions on time-varying sin series.

6.4.2 Complex-valued network:compared with real-valued network

Objective

The parameters in *ForeNet* are complex-valued and *ForeNet* is trained by complex-valued learning algorithm. As we known, a complex number provides not only the magnitude information but the phase state. Therefore a neuron with complex value has more representational power than a real-valued neuron. Moreover, Akira proved the behavior of complex-valued recurrent network is more stable than real-valued recurrent network[33]. For comparisons, we train a recurrent network with randomly generated real-valued weights.

Results and Analysis

Table (6.3) shows the training results using real-valued network. Comparing to Table (6.1), it is obvious that on the predictions of Mackey-Glass series, time-varying sinusoidal series and noisy sinusoidal series, the complex-valued network has more powerful computational ability in terms of testing error and training time. As to laser data forecasting, real-valued model achieve better testing result (with RMS error 0.3906) than complex-valued model, whose testing RMS error is 0.5049. However, to perform the better prediction, real-valued model took 32 minutes to convergent, which is much longer than complex model, whose training time is around 3 minutes.

In [76][77], the authors used the phase of the complex value to represent multiple orientations in a recurrent network thus more stable population activity patterns can be formed in the model.

series	Testing RMSE	Iterations	Time taken(Sec)
Mackey-Glass series	0.0601	500	2117
laser data	0.3906	300	1868
time-varying sine series	0.2836	75	166
noisy sinusoidal series	0.3267	500	575

Table 6.3: The prediction performance of the real-valued RNN

6.4.3 Simple architecture:compared with ring-structure RNN

Objective

Since *ForeNet* is a self-connected recurrent model, each hidden unit is connected to itself only. Thus, compared with fully connected recurrent mode (FRN), the structure of *ForeNet* is very simple, and its simplicity also reduces the computational complexity. One may double the computational ability of such simple recurrent model. In [12], the author suggested a ring-structure locally recurrent model (RRN) and showed that it needs a much shorter time to train the RRN model and its performance is comparable to that of FRN. In this section, we test the computation power of our model.

In order to verify whether other recurrent weights between hidden units are helpful to improve our model's performance, we also use a ring-structure recurrent model (shown in Figure (6.4)) to do the same prediction tasks. We do such verification in the following way

- We assume in the beginning of training, RRN has the same architecture as *ForeNet* model. That is, in RRN, the weights constructed as those in *ForeNet* are initialized by the proposed method. The weights, those from hidden units to their two nearest neighbors, are set to zeros at the beginning of training.

- RRN is also trained by the proposed CRTRL algorithm. All recurrent weights are updated in the same way.

Results and Analysis

We demonstrate here three prediction tasks: the sinusoidal series with noise; Mackey-Glass series; laser series. Table (6.4) and Table (6.5) show the prediction results. The results presented are the root mean square error of testing data, the number of iterations required and the training time taken. The predicted three curves are shown in the Figure (6.5) and Figure (6.6).

The learning speed of self-connected model is faster than ring-structure model according to the number of iterations required and the training time taken. In Mackey-Glass prediction task, RRN needs more time than *ForeNet* in order to achieve the same generalization error. The advantage of the convergence speed is more obvious in laser series prediction problem, which includes a large training data set (1500 data points). The additive recurrent links increase the computational complexity of the model, and RRN requires almost double time to convergence. As to the generalization ability, from the tables, the self-connected recurrent model outperforms the ring-structure recurrent model in terms of the root mean square error. Thus, *ForeNet* with only self-connections in the hidden layer is practically more useful especially in the tasks involved a large number of training data.

6.4.4 Linear model: compared with nonlinear *ForeNet*

Objective

In the proposed *ForeNet*, all units are linear. That makes the recurrent model very simple. In the previous chapters, we have proved that such simple linear

series	No. of recurrent links	RMS Error	No. of iterations	Time taken (sec)
noisy sinusoidal series	4	0.0679	22	23
Mackey-Glass series	4	0.0102	13	63
laser data	4	0.5049	19	142

Table 6.4: Prediction performance of the model with self-connection in hidden layer

series	No. of recurrent links	RMS Error	No. of iterations	Time taken (sec)
noisy sinusoidal series	12	0.1041	25	26
Mackey-Glass series	12	0.0102	13	81
laser data	12	0.5139	23	219

Table 6.5: Prediction performance of the model with ring-structure in hidden layer

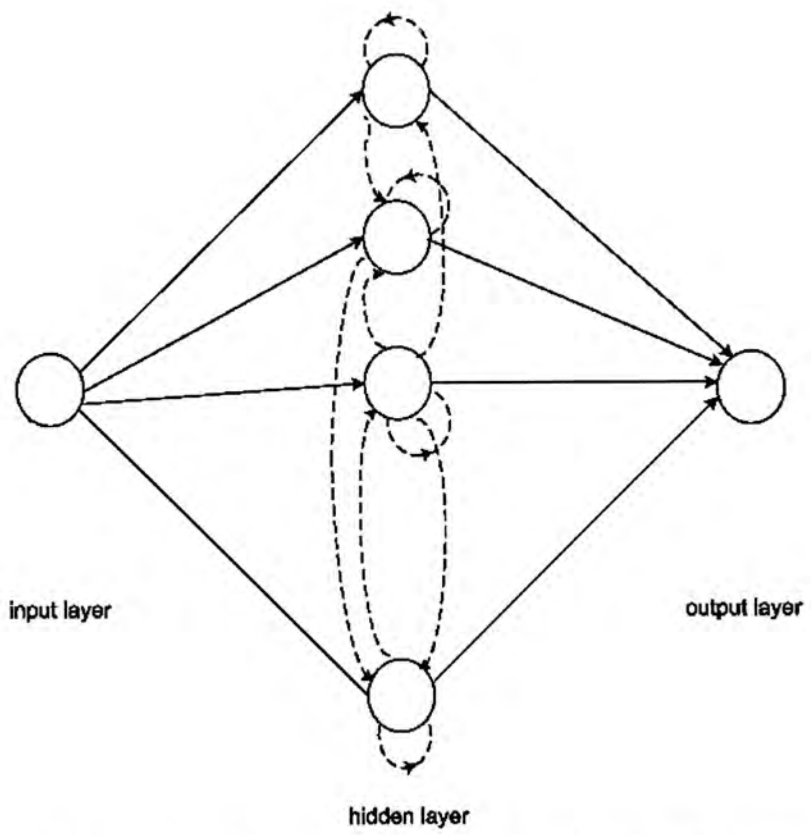
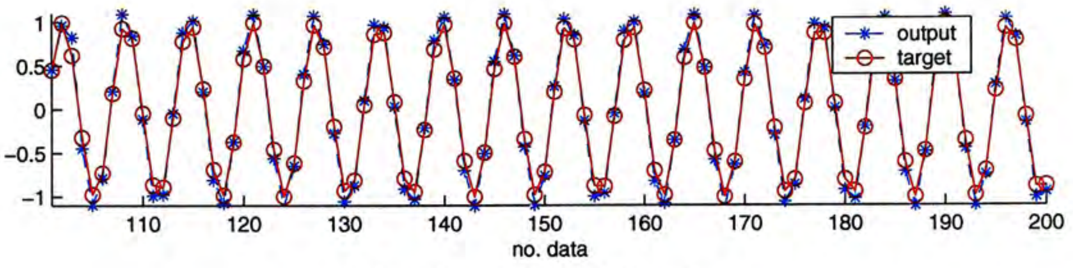
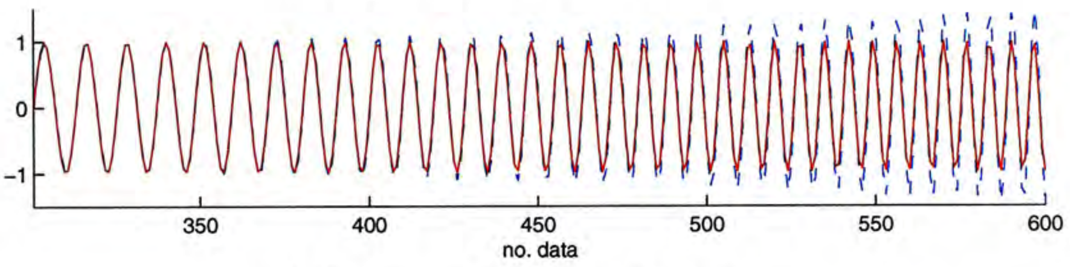


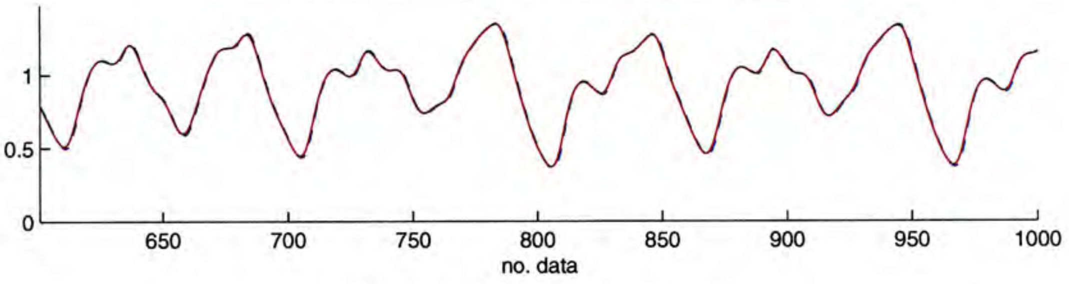
Figure 6.4: Ring-structure Recurrent Model (RRN)



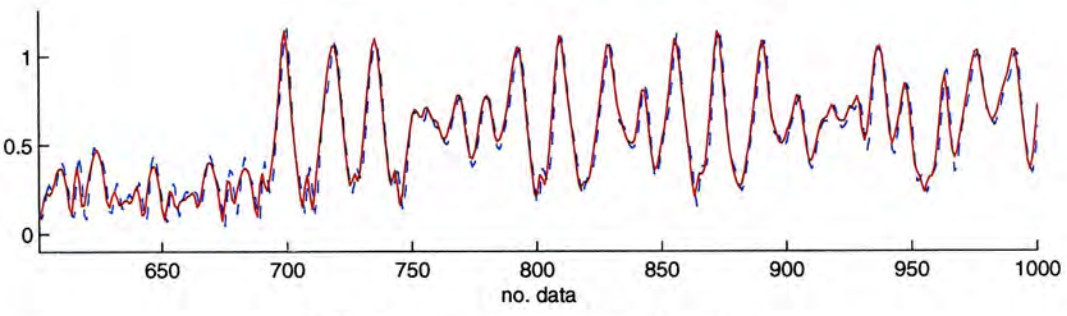
(a) noisy sinusoidal series



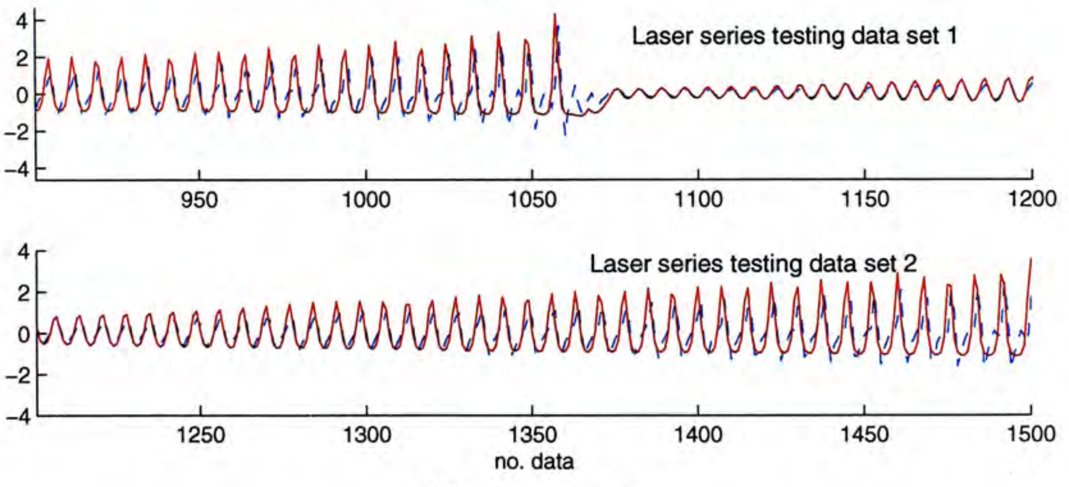
(b) time-varying sinusoidal series



(c) Mackey-Glass series

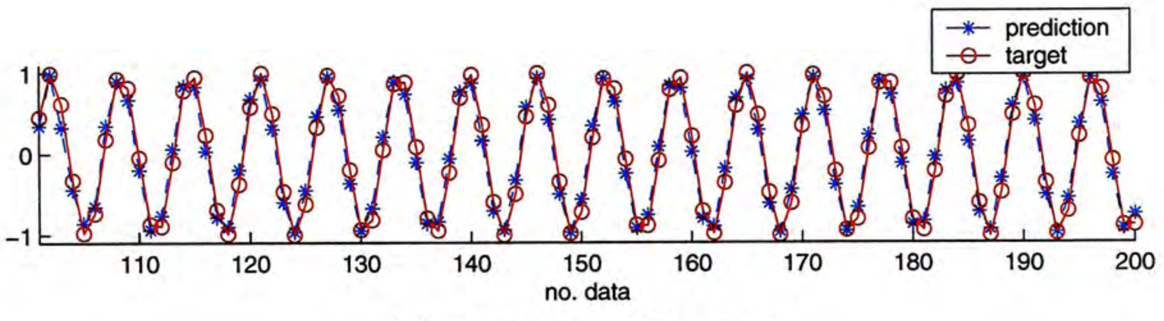


(d) computer-generated series

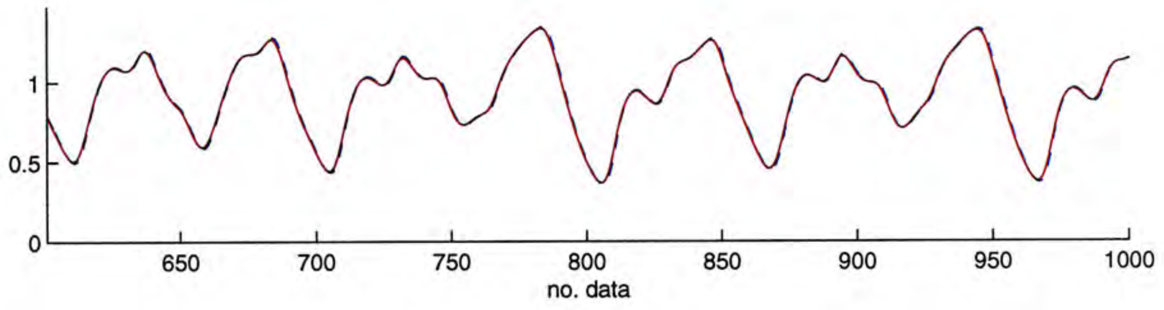


(e) laser series

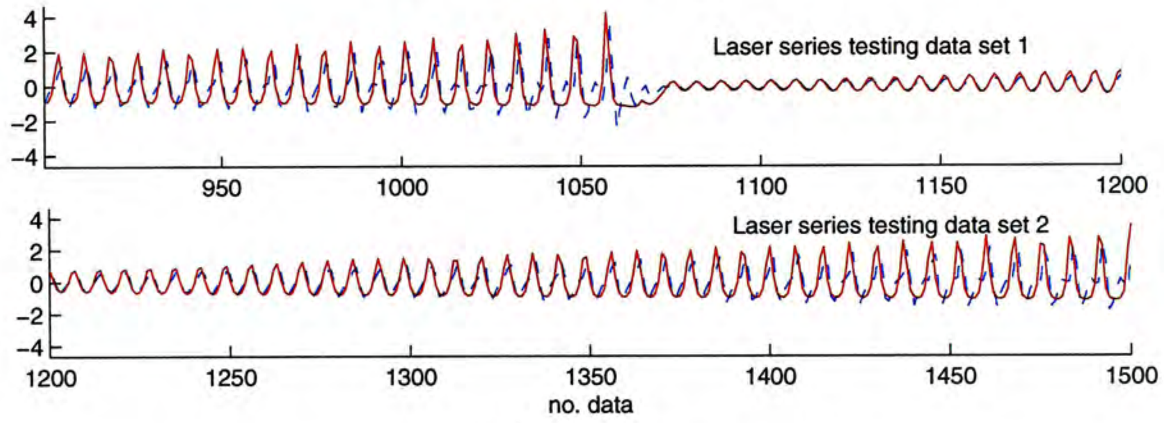
Figure 6.5: The prediction performance of *ForeNet* with self-connection in the hidden layer. The solid line represents the correct outputs and the dotted line shows the network output.



(a) noisy sinusoidal series



(b) Mackey-Glass series



(c) laser series

Figure 6.6: The prediction curves of the model with ring-structure in the hidden layer. The solid line represents the correct outputs and the dotted line shows the network output.

model also has power computational ability. Some interesting aspects of linear NN models have been proposed in [5]. However, since a linear model can only represent linear input-output mapping, one may consider using nonlinear units in *ForeNet* to achieve better generalization, especially to high nonlinear time series problems. In this chapter, we will show the *nonlinear ForeNet* model and its computation ability and discuss whether it is suitable in our case.

Architecture of nonlinear *ForeNet*

As described in chapter 3, in *ForeNet* model, the activation functions in hidden layer and output layer are both linear functions, which are derived based upon the Fourier recursive equation. In order to extend the linear model to the nonlinear model, one common way is to define the activation function as one nonlinear function. In our case, when finding a proper nonlinear transfer function, we consider it as the follows.

Based on the linear Fourier recursive equation (4.4), *ForeNet* model was proposed with linear activation functions, both in initialization stage and training stage. And because of the proposed initialization method, the network can attain faster convergence and better generalization. Therefore, in order to preserve the advantages of proposed linear modelling, the nonlinear function should be properly defined. That is, should we use nonlinear activation function on hidden units directly or on output units?

As we analyzed before, in the proposed equation (4.4), the recursive variable h plays an important role. Moreover, the linear aspect of the recurrent units make the network more stable. Thus it suggests to preserve the linear transfer function on the hidden units. Further, because the time series $x(t)$, $t = 1, 2, ...,$, could be any value, a linear transfer function is needed in the

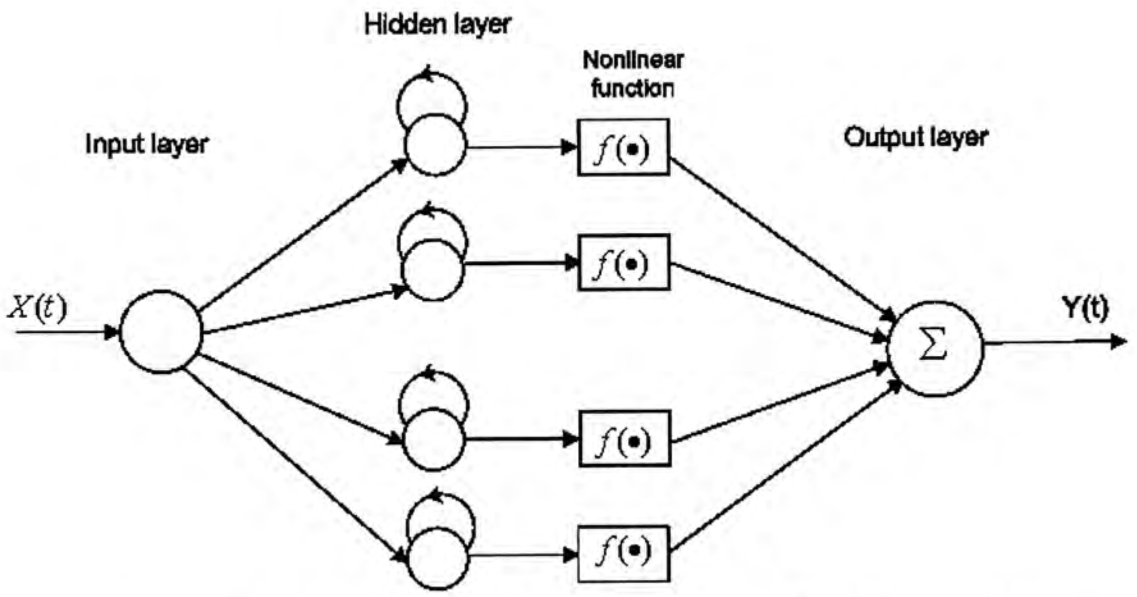


Figure 6.7: Architecture of the nonlinear *ForeNet* model

output layer. From the above discussion, we add a nonlinear function between hidden layer and output layer as shown in figure (6.7). The activation function on hidden units is linear, and the outputs of hidden layer have been feeded back before entering into a nonlinear transfer function $f(\cdot)$. After the nonlinearity, all signals are summed with different weights in the linear output layer. Similar feedback architecture was discussed in [64].

We choose a "symmetrical" nonlinear function, whose outputs are between $[-1, 1]$. Thus *ForeNet* becomes:

$$\begin{aligned} z(t) &= 2 \times y^{re}(t) \\ y(t) &= \sum_{j=1}^{N} f(h_j(t))W_{jl} \\ h_j(t) &= W_{jj}h_j(t-1) + W_{kj}x(t) \end{aligned} \tag{6.14}$$

where $f(\cdot) = \tanh(\cdot)$.

Learning Algorithm and Simulation Results

Similar to learning algorithm for linear *ForeNet* model, to train the model with nonlinear activation function, there are two steps in our learning process. The first one is the network initialization which is a linear method the same as described in the previous chapter. The second is to train the network using the modified complex RTRL algorithm.

Our simulation results are summarized in Table (6.6), we present the out-of-sample Root Mean Square Error (RMSE) for *ForeNet* with linear and nonlinear activation functions respectively.

From the results, we find that for laser series prediction, the additive nonlinear function in the model can help to improve the ability of learning and prediction in terms of training time and testing root mean square error. In the case of noisy series prediction, the nonlinear *ForeNet* can get better generalization ability, but it needs more training time to convergent. Similarly, for sunspot data forecasting, the nonlinearity seems to be helpful to achieve better generalization according to the smaller testing error compared to one using linear *ForeNet*. However, the network used 167 iterations to convergent. That is more than 10 times longer than the training time of the linear model. The same situation can be found in Mackey-Glass prediction. Compared to the linear model, which is convergent after 63 seconds with error $TRMS = 0.0102$, the nonlinear model paid much more time (450 seconds) to achieve a little better generalization ($TRMS = 0.0097$). As to the prediction of time-varying series, the nonlinearity worked even worse than the linear model, no matter in the aspect of convergence speed or generalization ability.

From the experimental results, we show that the nonlinearity in NN can

	Initial RMS error	Testing RMS error	No. of iterations	Time taken (sec)
		laser data		
linear *ForeNet*	0.7449	0.5049	19	142
nonlinear *ForeNet*	0.7599	0.4703	15	108
		sunspot series		
linear *ForeNet*	0.0401	0.0885	14	28
nonlinear *ForeNet*	0.0717	0.0824	167	299
		Mackey-Glass Series		
linear *ForeNet*	0.0205	0.0102	13	63
nonlinear *ForeNet*	0.0216	0.0097	100	438
		time-varying sine series		
linear *ForeNet*	0.0746	0.1251	103	257
nonlinear *ForeNet*	0.0749	0.1633	500	1226
		noisy sinusoidal series		
linear *ForeNet*	0.699	0.0729	18	24
nonlinear *ForeNet*	0.6526	0.0436	46	59

Table 6.6: The prediction performance of the linear and nonlinear model on different time series.

be helpful to forecasting time series, especially to predict highly nonlinear or noisy time series such as laser series and noisy sinusoidal series. But we argue that the nonlinearity in NN does not always provide more powerful computation than linear NN model, especially in our case, the linear model is built with the specific purpose. Recently, many methods with linear or semilinear transfer functions have been studied, such as [73][20][30]. And in [5], the author mentioned that from the standpoint of theoretical biology, certain classes of neurons may be operating most of the time in a linear or quasi-linear regime and linear input-output relations seem to hold for certain specific biological circuits (see [58] for an example). Moreover, usually the nonlinear transfer function has lower and upper saturation limits, then the dynamics may be bounded.

6.4.5 Small number of hidden units

Objective

As we known, in Recurrent Neural networks, with the increasing of number of hidden units, the generalization ability of RNN is also improved ignoring accordingly increased computation time. However, we have shown in Chapter (5) the weights are related to the number of hidden units, thus the different number of hidden units make the different initialization performance of *ForeNet.* That means the relationship between hidden units number and computation ability may not exit in our model.

In order to analyze this relationship and find a trade-off between them, we design some experiments. We use the *ForeNet* model with various number of hidden units and test their prediction. We illustrate here the number of hidden units, the initial training error, the generalization error and training time.

Results and Analysis

Table (6.7) shows the training results. When 10 hidden units are used in the model, the error after network initialization becomes significantly larger and finally the network achieve much larger prediction error. Also, because the initial state may be farther away from the local minimum, using the increased hidden units need more training time to convergent according to training iterations. Such cases are in the prediction of time-varying series, noisy sinusoidal series and Mackey-Glass.

From thees experimental results, we find that increasing the number of hidden units in *ForeNet* may not improve its prediction ability. It is because the number of hidden units implies the embedding dimension of the time series. And It is obviously when the estimated dimension is close to the true dimension, the model is near to local minimum.

The special case is appeared in laser series prediction. Using $1-10-1$ model, the network can achieve better generalization in terms of testing error with the comparative training time. Since *ForeNet* use linear transfer function which makes its limitation on handling high nonlinear time series. Thus, by adding the number of hidden units, the network improves its computation power which helps to solve high nonlinear problem.

6.5 Comparison with Some Other Models

In the above sections, we showed some properties of proposed prediction model *ForeNet*. In this section, we will compare *ForeNet* with AR model, TDNN Network and FIR Network. The brief descriptions of AR model, TDNN Network and FIR Network can be found in chapter 2.

No. of Hidden Units	Initial RMS error	Testing RMS error	No. of iterations	Time taken (sec)
		Laser data		
4	0.7449	0.5049	19	142
10	0.7638	0.3601	19	164
		Mackey-Glass series		
4	0.0205	0.0102	13	63
10	0.0401	0.0127	500	2564
		Time-varying sine series		
3	0.0746	0.1251	103	257
10	0.1951	0.3162	266	765
		Noisy sinusoidal series		
4	0.699	0.0729	18	24
10	0.5620	0.0848	13	29

Table 6.7: The prediction performance using various number of hidden units in *ForeNet*.

6.5.1 Comparison with AR model

First, we do the comparison between AR model and *ForeNet* model. Recall the general AR model:

$$x(t) = \sum_{m=1}^{M} a_m x(t-m) + e(t) \tag{6.15}$$

where M is the order of AR model. $e(t)$ represents either a controlled input to the model or noise.

Compared the *ForeNet* with AR model, since there is only one input in the model, the proposed model is similar with AR model with one order, i.e. AR(1) model. However, in our model, the coefficients are represented by the complex values.

For comparisons, AR(1) model is implemented with MATLAB. The coefficients in AR model are estimated by a set of training data using the least-squares algorithm. After determination of the coefficients, AR model then is applied to do prediction on a set of testing data. In order to show the ability of high order AR model, we provide the prediction results of AR(16) model. Here we use *ForeNet* without training to do the same prediction tasks.

Results and Analysis

We test on five time series. And we use Root Mean Square Error on testing data as measurement. The results are shown in Table (6.8). It is easy to find that *ForeNet* outperforms AR(1) model on all five series. AR model with only one order seems difficult to model the given series, which may be caused by insufficient memory. But since in our model, there are p hidden units, which all act as the storage of the information from past data, thus only using one input variable, it can contain and present more efficient memory for prediction.

methods	*ForeNet* before training	AR(1) model	AR(16) model
Mackey-Glass series	0.0128	0.038	2.409*e*-04
Laser data	0.4973	0.8267	0.3579
Sunspot data	0.0874	0.1546	0.0907
noisy sinusoidal series	0.2931	0.5968	0.0060
time-varying sinusoidal series	0.1951	0.5071	0.0106

Table 6.8: Comparisons of prediction performance among *ForeNet*, AR(1) model and AR(16) model.

When we increase the order of AR model from 1 to 16, as shown in Table (6.8), AR model shows its prediction ability more powerful than our model on Mackey-Glass series, Laser data, noisy series and time-varying series. It is not surprising that systematically increasing the order of AR model, the model will fit the time series better. Nevertheless, it is unfair to compare an AR(16) model with our model which has one order. Moreover, the selection of "right" order of AR model is also a tough task since too large order may induce over-fitting problem [25], that is, the fitting error on the measured data decreases, but the test error of the forecasts beyond the training set will start to increase. Comparing the testing error using *ForeNet* and AR(16), we found that our model can still get better generalization than AR(16). In this case, AR(16) may be fitting extraneous noise in the series.

6.5.2 Comparison with *TDNN* Networks and *FIR* Networks

In this section, we will compare the prediction performance of *ForeNet* with *TDNN* Networks and *FIR* Networks. The comparison is done by forecasting two time series, Mackey-Glass series and Laser series. Here we use the prediction results from [9].

In the case of Mackey-Glass series prediction, the architecture of TDNN was $5-5-1$, that is, 5 input units, 5 hidden units and 1 output unit. The architecture of FIR network was $1-5-1$ with 5 taps in the hidden layer and 5 taps in the output layer, respectively. As to *ForeNet*, we use a $1-4-1$ model. The simulation results are shown in Table (6.9). It is shown our model outperforms other two models in terms of testing root mean square error.

As for Laser data forecasting, in [9], the authors used a $1-12-12-1$ FIR network with 25 taps in the first hidden layer, 5 taps in the second hidden layer and 5 taps in the output layer. TDNN with a delay line of 25 nodes and 12 nodes in the first and second hidden layer were chosen. We still use $1-4-1$ *ForeNet* model to perform the same prediction task. Table (6.9) shows the RMS error on testing set. TDNN and FIR networks performed nearly equal in one step ahead prediction. But *ForeNet* did the best prediction with the smallest error 0.5049.

There were two hidden layers in both TDNN and FIR networks, and they used a large number of neurons in the networks, which can increase their computational ability. And in our model, there is only one hidden layer with 4 hidden units. Though the model size of TDNN and FIR networks is much larger than that of *ForeNet*, *ForeNet* still demonstrates best generalization

methods	*ForeNet*	*TDNN* Networks	*FIR* Networks
Mackey-Glass series	0.0102	0.102	0.0624
Laser data	0.5049	0.738	0.762

Table 6.9: Prediction performance comparison among three networks.

ability among these three networks, and shows its efficient learning ability with a small model size.

6.5.3 Comparison to a few more results

In this section, we survey the best results on the prediction of sunspot data obtained by other methods. Following previous publications[63, 71], comparisons are made on the basis of the average relative variance(ARV), given by

$$ARV = \frac{1}{\sigma^2 N} \sum_{t=1}^{N} (x(t) - \hat{x}(t))^2 \tag{6.16}$$

where $x(t)$ is the true value, $\hat{x}(t)$ is the prediction, and σ^2 is the variance of the true series over the prediction duration N.

Table (6.10) illustrates the one-step-ahead prediction ARV measures for *ForeNet* and two standard benchmark models for the sunspots, namely the TAR model[63] and the multilayer perceptron[71]. In this table, we also show the number of adjustable parameters needed by each model. The *ForeNet* model seems to offer no dramatic advantage over a multilayer perceptron in terms of training error and prediction performance on the first testing set $1921-1955$. Due to the nonstationarity of the time series, for TAR and MLP models, the performance on the second testing set drops largely, thus the *ForeNet* achieves the best prediction on the data from 1956 to 1979. Further, the number of parameters in our model has been significantly reduced from 43 to 13.

Model	Training set (1701-1920)	Testing set 1921-1955	Testing set 1956-1979	number of parameters
TAR model	0.097	0.097	0.280	18
MLP $12 \times 3 \times 1$	0.082	0.086	0.350	43
ForeNet	0.09	0.11	0.17	13

Table 6.10: Prediction performance comparisons

6.6 Summarization

In the previous sections, we predicted several time series with various training methods and models. Table (6.11) shows the comparisons among different learning methods on the forecasting of 4 time series, i.e. laser data, Mackey-Glass series, time-varying sinusoidal series and noisy sinusoidal series. In this section, we summarize the performance of *ForeNet* on the prediction of the series with different properties:

- Series with noise. Such data include noisy sinusoidal series and Mackey-Glass series. From the table, we show that in the case of noisy time series prediction, the performance of *ForeNet* is not as good as that of model with nonlinear activation function in terms of RMS error. However, for the clean time series, such as time-varying sinusoidal series, *ForeNet* can demonstrate the best generalization.

- Series with nonstationary property. Such series include time-varying sinusoidal series. As shown in the table, *ForeNet* model with the proposed learning method can get good prediction results on the nonstationary time series.

- Series with nonlinearity. Laser data is highly nonlinear time series. We found that linear ForeNet model has the limitation on handling such series. Using the nonlinear function in the networks or increasing the number of hidden units can both help to improve the prediction ability of ForeNet.

Methods	Initial RMS error	Testing RMS error	No. of iterations	Time taken (sec)
		Laser data		
ForeNet with proposed initialization	0.7449	0.5049	19	142
ForeNet with random initialization	1.1961	0.4784	202	1324
Real-valued *ForeNet*	0.668	0.3906	300	1868
Ring-structure Recurrent networks	0.7449	0.5139	23	219
nonlinear *ForeNet*	0.7599	0.4703	15	108
AR(1) model	–	0.8267	–	–
		Mackey-Glass		
ForeNet with proposed initialization	0.0205	0.0102	13	63
ForeNet with random initialization	2.2589	0.0558	198	866
Real-valued *ForeNet*	0.6455	0.0601	500	2117
Ring-structure Recurrent networks	0.0205	0.0102	13	81
nonlinear *ForeNet*	0.0216	0.0097	100	438
AR(1) model	–	0.038	–	–
		Time-varying sinusoidal series		
ForeNet with proposed initialization	0.0746	0.1251	103	257
ForeNet with random initialization	0.9676	0.3203	80	184
Real-valued *ForeNet*	0.501	0.2836	75	166
Ring-structure Recurrent networks	–	–	–	–
nonlinear *ForeNet*	0.0749	0.1633	500	1226
AR(1) model	–	0.5071	–	–
		Noisy sinusoidal series		
ForeNet with proposed initialization	0.699	0.0729	18	24
ForeNet with random initialization	0.6889	0.4285	52	60
Real-valued *ForeNet*	0.5450	0.3267	500	575
Ring-structure Recurrent networks	0.6163	0.1041	25	26
nonlinear *ForeNet*	0.6526	0.0436	46	59
AR(1) model	–	0.5968	–	–

Table 6.11: Summarization of the prediction performance using various methods on four time series.

Chapter 7

Learning and Prediction: On-Line Training

On-line learning is an alternative to bach-mode learning. In this chapter, we introduce the advantages and disadvantages of on-line method then propose a prediction method. We show how to perform online prediction, and give the comparison between batch-mode and online-mode learning by some experimental results.

7.1 On-Line Learning Algorithm

In on-line learning, the weights are updated after each pattern presentation, using the gradient of the single-pattern error. Generally, the patterns are presented in a random to void cyclic effect [56]. However, for time series prediction task, it is impossible to change the order of patterns. Thus, the pattern is chosen according to time order.

7.1.1 Advantages and Disadvantages

There are some advantages of on-line approach:

- First, there is no need to store and sum the individual derivatives; each pattern derivative is evaluated, then discarded immediately.

- Second, although the proposed fourier recursive model is derived as a local model, it still hasn't shown its ability of capturing the changes in the different sets of data. It is because we only use the same value of parameters when modelling each set of data. Even in the previous section, *ForeNet* is used to train the parameters, since batch-mode learning method is adopt, and in batch-mode learning, the weights are updated only one time according to the contribution of a whole training set. One may consider after batch-mode training , Neural Networks can capture global data characterize. However, on-line learning adjusts weights according to one recent pattern's contribution. So it can detect the changes in the data distribution. This is important for short-term prediction task, especially when the time series is nonstationary.

With batch-mode learning, since the training patterns can be used repeatedly during training phase, we can check whether we are making progress. We can minimize the objective function to a desirable precision. And we can compute the error function on a validation set and stop training when the generalization error becomes go up. However, with on-line learning, we cannot do the above things. We cannot compute the error function on the training set or validation set for a fixed set of weights because the patterns are discarded after use [60]. Hence, on-line learning is generally more difficult and unreliable than batch-mode learning. Moreover, updating the parameters only based on one time step estimation error, the network cannot capture the temporal information of the whole data set.

7.1.2 Training Process

From the analysis of the advantages and the disadvantages of on-line learning, we propose a learning method which is a balance between bath-mode and on-line mode. The whole data set is divided into two parts, named training set and testing set. First, the network is trained by both-mode learning algorithm with the training data set. Then the on-line learning is applied to perform prediction task. In such way, in the bath-mode learning phase, the network is expected to learn the global distribution of the time series. And then in the on-line learning phase, the network detects the changes in the data distribution and perform better short-term prediction.

The whole training steps are:

(1) *ForeNet* is initialized by the proposed method as described before,

(2) Learning phase:

- For every pattern p in the training set,
  - apply a pattern p in the training set,
  - calculate the pattern error E_p and the single-pattern derivatives $\partial E_p / \partial w$.
- Add up all the single-pattern terms to get the total derivative,
- Update the weights according to CRTRL,
- Apply the validation data set, calculate the validation error $e_V(t)$,
- Compare the validation error with it in the last training epoch,
  - If $e_V(t) > e_V(t-1) > \cdots > e_V(t-n) >$, stop training.
  - else, repeat the learning phase.

(3) Apply the testing data set

- Pick a pattern p from the testing data set,
  1. apply pattern p and forward propagate to obtain network output (prediction), and

   2. calculate the pattern error E_p and back-propagate to obtain the single-pattern derivatives $\partial E_p / \partial w$,

- Update the weights,
- Repeat until finish the prediction of testing data.

7.2 Experiments

In this section, we apply on-line method to some time series prediction tasks. Compared with batch-mode learning, on-line learning can learn the latest statistics of the time series, hence it can help to improve the network performance on one-time step ahead prediction, especially to the time series that are non-stationary. In the following experiments, during batch-mode training process, the learning rate is set to 0.1 and the value of momentum is 0.5. And in the on-line prediction process, we reduce the learning rate to 0.05 and the momentum to 0.25. The network architecture is $1-4-1$ model. Here we test on 4 time series. Each is also divided into training data, validation data and testing data.

Table (7.1) shows the on-line prediction results on four time series. We report the root mean square error on testing data in the table.

The first predicted time series is time-varying sinusoidal series. Shown in Figure (5.13), the frequency of time series is increasing with measure time, and there are nonstationarities on the time scale of the sampling time. In Table (7.1), the root mean square error on testing data is reported. Using on-line mode, the error is 0.0304. Comparing to that with batch-mode learning, (RMSE=0.2346), we can find that to predict the time series which show high nonstationary, on-line method outperforms batch-mode method. For the purpose of comparison, Figure (7.1) shows the network performance using batch-mode method. It is obvious that though the network can do well on the small training data set, its prediction on testing data is much worse. It is because that the frequencies of testing set are much different than those in training set, so it is impossible for the network to discover the changes of frequencies. Compared with Figure (7.1), Figure (7.2) illustrates the prediction result using on-line method. The network can detect the changes of frequencies thus get much better prediction.

In Figure (7.3), the on-line learning curve is shown. With on-line learning, jitter in the $E(t)$ graph is normal. Because weights are updated after each pattern presentation, there is a tendency for the error to be lower on the most recently presented patterns, and it also introduces noise that shows up in the error curves. The graph implies that the learning is working well because the error curves has a downward trend. The curves is overlaid with noise, whose amplitude is related to the learning rate η.

Figure (7.4) demonstrates the changes on the magnitudes and the phases of recurrent weights during on-line prediction. As we have explained before, the predefined values of weights are somehow related to the frequencies of time series. Based upon the proposed idea in the previous chapters, the time series with the large dominant frequency would need the small weights to do good prediction. As we can see in Figure (7.4) that with the frequencies increase, the values of weights decrease correspondingly to match the changes of frequencies. The experimental results match well to our proposed method.

Similar results are found on the cases of Mackey-Glass and sunspot series. The network also improved its performance with on-line mode learning. On both experiments, on-line prediction can improve the prediction accuracy according to testing RMS error.

methods	time-varying sinusoidal series	Mackey-Glass series	sunspot series	laser series
on-line	0.0304	0.0084	0.0839	0.7163
batch-mode	0.2346	0.0102	0.0891	0.5049

Table 7.1: Comparison between on-line learning and bach-mode learning on 4 time series prediction

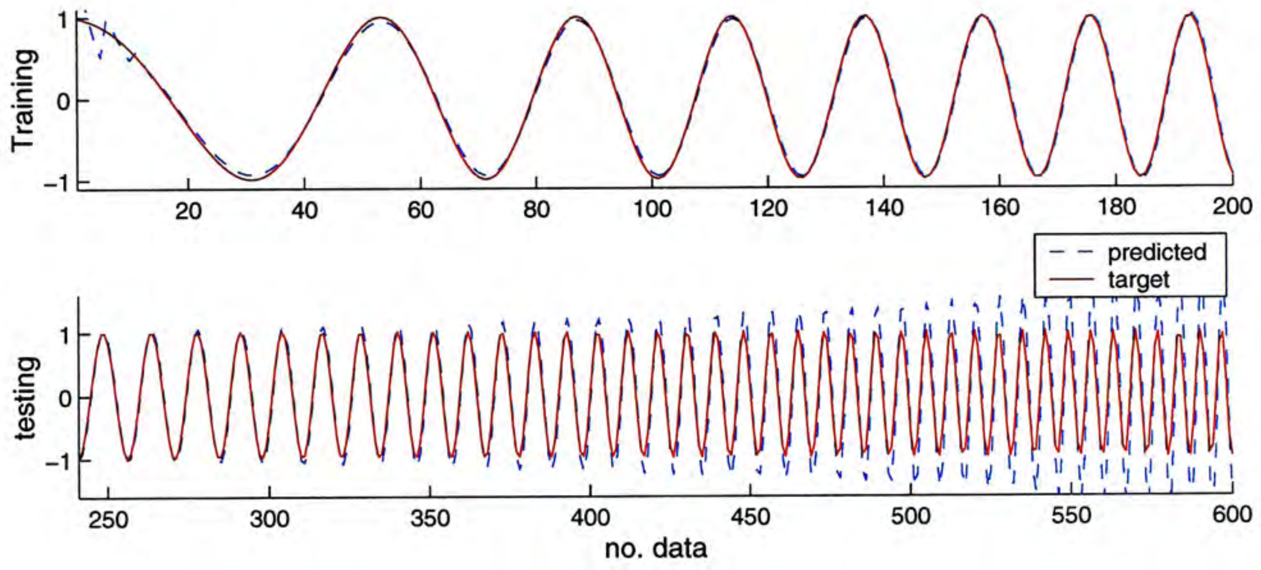


Figure 7.1: Time-varying sinusoidal series prediction using batch-mode learning. The first graph shows the performance on training data set; the second graph shows the prediction on testing data set.

However, as shown in Table (7.1), on the laser series prediction task, on-line learning works not good as batch-mode learning. In the experiments, we tested the series from 901 to 1500. As we can see from Figure (5.9), around 1050^{th} data in the series, there is a collapse, which is difficult to be predicted only paying attention to the previous several data points. This may be one of the reasons that on-line learning does not work in the case of laser series prediction. Another possible reason is laser data is stationary[70]. And as we know, online learning is do well in the series which show nonstationary. So it is of little benefit to stationary time series prediction.

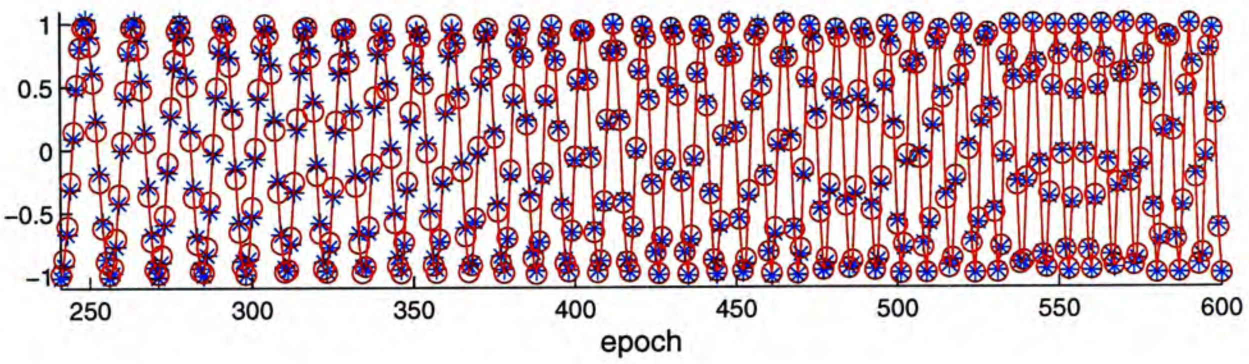


Figure 7.2: Time-varying sinusoidal series prediction using on-line method

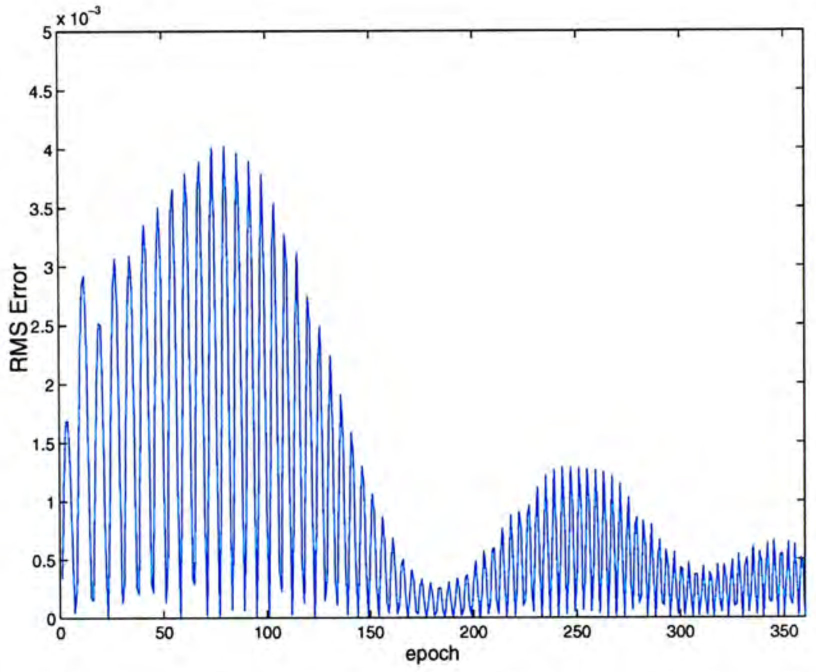


Figure 7.3: Learning curve of on-line method on time-varying sinusoidal series prediction

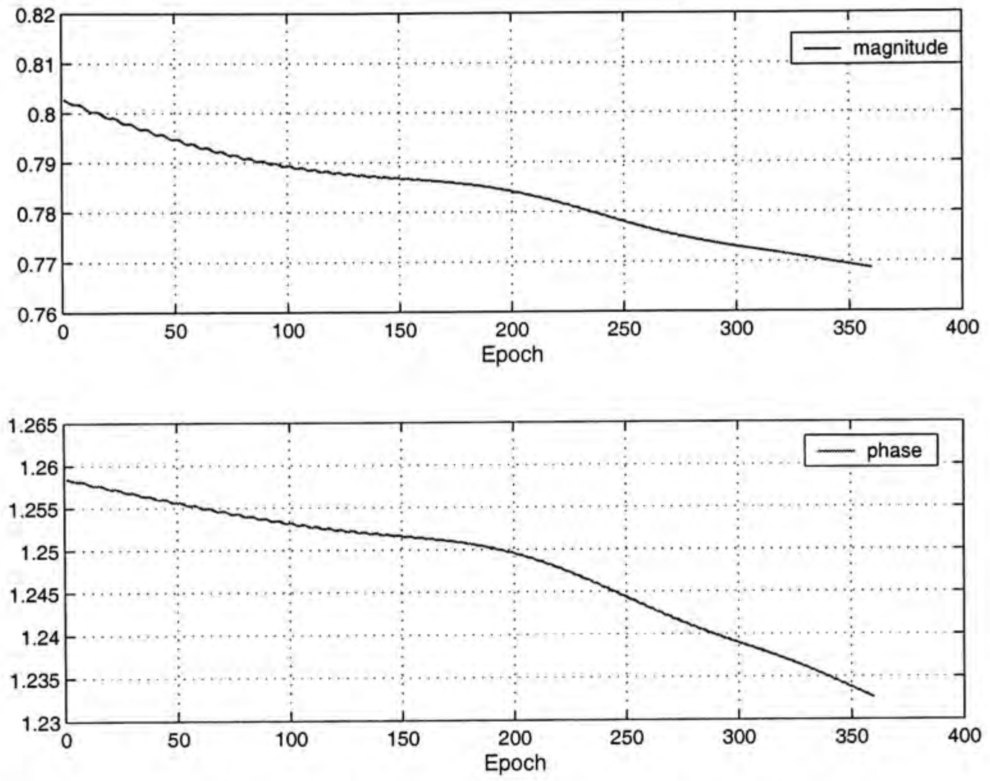


Figure 7.4: Changes on a self-connected weight during on-line prediction; the upper graph shows the changes on magnitudes, and the lower graph shows the changes on its phases.

7.3 Predicting Stock Time Series

In this section, we apply our method to predict the future values of the stock entities. Since the stock market behaves very much like a random-walk process, prediction of stocks is generally believed to be a very difficult task. Thus the different evaluation method should be used.

Returns Prediction

In the case of stock predictions, returns $R(t)$ are often chosen instead of the original stock prices. A common variant is the log-return

$$R(t) = \log \frac{y(t)}{y(t-1)} \tag{7.1}$$

where $y(t)$ denote close values for the stock for each day of trading (Monday-Friday). $R(t)$ has a relatively constant range even if stock data for many years are used as input. Thus the returns $R(t)$ are used as model input series.

Evaluating Performance

The naive prediction of stock returns asserts today's return as the best estimate of tomorrow return. It is a good idea to measure the goodness of a predictor in relation to this trivial predictor[62]. In our work, we compare the model performance to that of the naive return predictor and adopt relative hit rate as measurement.

The hit rate of a predictor indicates how often the sign of the return is correctly predicted. Let the series $\hat{R}(t), t = 1, \ldots, N$ denote the prediction of returns at time t, $R(t), t = 1, \ldots, N$ denote the actual returns. The hit rate is computed as the ratio between the number of correct non-zero predictions $\hat{R}(t)$ and the total number of nonzero moves in the stock time series.

$$H = |\frac{t|R(t)\hat{R}(t) > 0, t = 1, \ldots, N}{t|R(t)\hat{R}(t) \neq 0, t = 1, \ldots, N}| \tag{7.2}$$

The corresponding measure for the naive return predictor is

$$H_N = |\frac{t|R(t)R(t-1) > 0, t = 1, \ldots, N}{t|R(t)R(t-1) \neq 0, t = 1, \ldots, N}| \tag{7.3}$$

The ratio between the two hit rates is

$$H_0 = \frac{H}{H_N} \tag{7.4}$$

where, H_0 is called "Relative hit rate", compares the hit rate of the predictor to that of the naive return predictor. For $H_0 < 1$ the predictor is worse than the naive return predictor, while $H_0 > 1$ implies that the predictor is making better predictions.

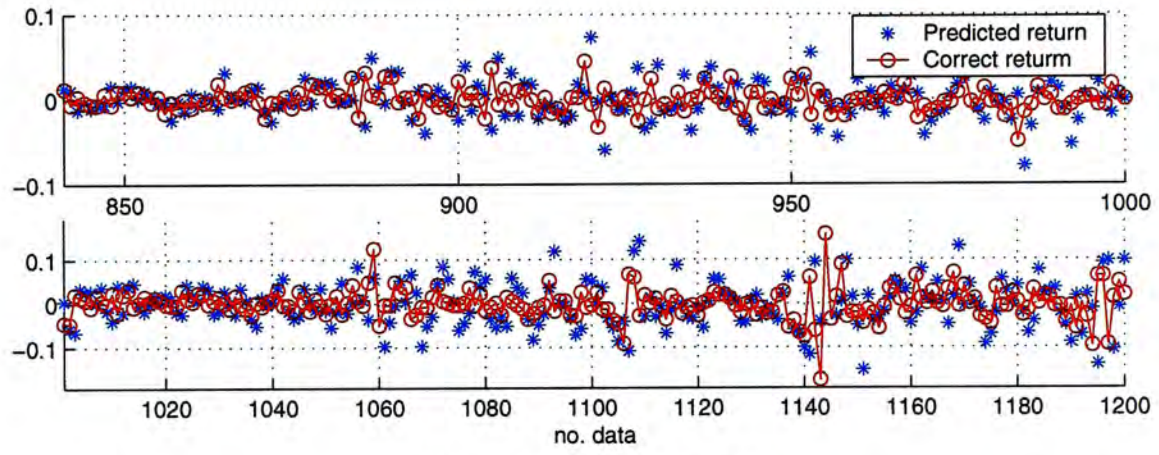
Stock	Hit Rate H	Relative Hit Rate H_0	RMS error
HANG SENG BANK	0.5260	1.0204	0.0457
HENDERSON LAND	0.5477	1.0058	0.0402
SHK PPT	0.4983	0.9605	0.0352

Table 7.2: Prediction results on stock returns.

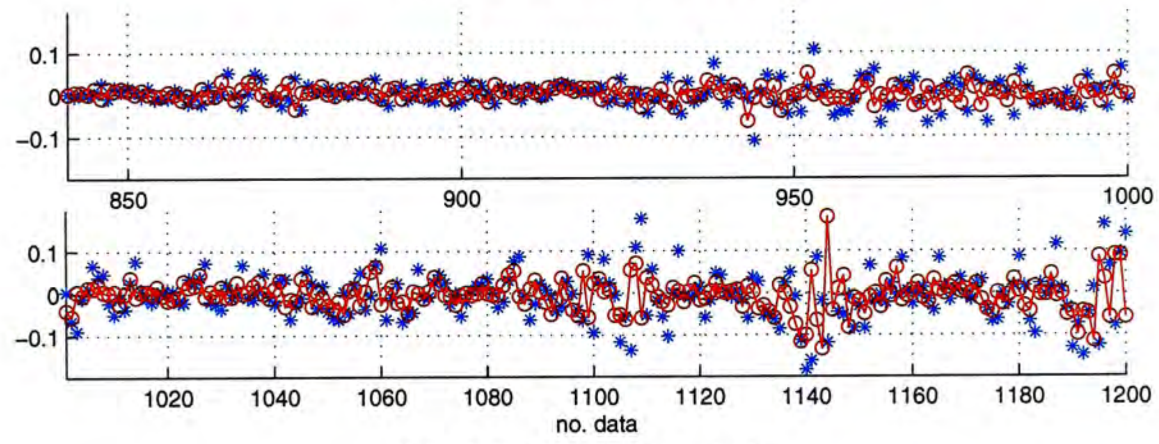
Experimental Results

We predict here the stock returns for 3 stocks, i.e. Hang Seng Bank, Henderson Land and SHK PPT, from $1993-03-15$ to $1998-01-16$, totally 1200 data. The last 360 data points act as testing data. The architecture of *ForeNet* is $1-4-1$ network. The online CRTRL algorithm is used to update the parameters.

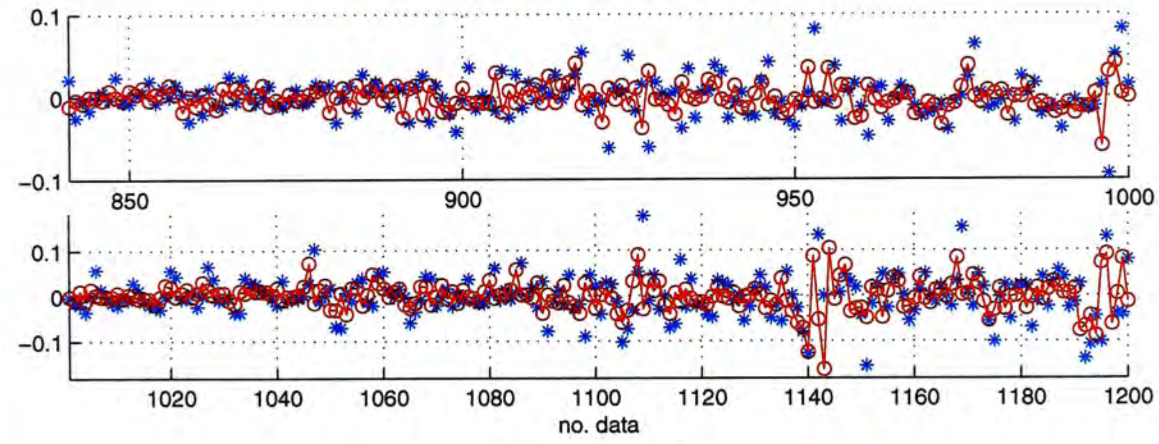
Table (7.2) shows the model's performance on the prediction of stock returns. Because the levels of noise in financial markets are so high, the model can only get the hit rate slightly better than 50% (for Hang Seng Bank: 52.6%, for Henderson Land: 54.77%), and it is even worse than 50% (for SHK PPT: 49.83%). For the stocks Hang Seng Bank and Henderson Land, the model made better predictions than those of the naive return predictor in terms of relative hit rate H_0.



(a) HANG SENG BANK



(b) HENDERSON LAND



(c) SHK PPT

Figure 7.5: The prediction curves of the stock returns.

Chapter 8

Discussions and Conclusions

In our work, we proposed the *Fourier Recursive Equation*, and based on it, *ForeNet* model was built. We analyzed the model initialization, and trained the model with complex-valued RTRL learning algorithm. Some experiments were demonstrated to show the properties of *ForeNet.*

In the following, we analyze the limitations and advantages of the proposed *ForeNet.* Some future works are mentioned as the concluding remarks.

8.1 Limitations of *ForeNet*

There are some limitations in our proposed model.

- The proposed *ForeNet* model is a restricted model. There are only a few free parameters in the model. The selection of number of hidden units and parameters initialization combine into one task because they are both controlled by one variable. Once we decide the architecture of *ForeNet,* the parameters initialization is determined, thus the initial state of the model is set. Such restricted model is very simple on modelling and learning, however, it may also encounter some computational difficulties.
- In the proposed *ForeNet* model, the activation function on hidden layer and output layer are both linear functions. We have demonstrated that

such linearity can also get good predictions on some time series, but may suffer if the time series is high nonlinearity. As shown in Chapter (6), when the linear transfer function is replaced by a nonlinear function, tanh, the network can better predict high nonlinear time series, for example, laser series. The similar improvement can be found if we increase the number of hidden units.

- According to the initial method (from equation (4.4)), after network initialization, the recurrent weights in the neural network are symmetric. That is, we give redundant weights to the network at the beginning of training, and it seems that only using half-number of recurrent weights can also achieve the same performance. However, in the experiments, we found that keeping all recurrent weights (even they are symmetric) are important to prediction because it can keep the phase symmetric, that is vital to recover original time series as described in Chapter 4. We thus introduce the bias to the network, and later training also makes the weights asymmetric.

- In *ForeNet*, we have shown by some experiments that the selection of parameters corresponds to the nature of the given time series. If the "hidden memory" can be built properly, the network achieves convergence quickly. However, how to choose the proper parameters based on different time series is still unknown in our current work.

- *ForeNet* is only used for time series prediction tasks because it is derived based upon a prediction method. And it only uses one variable as the model input, i.e. the network model can only be $1-p-1$ model.

8.2 Advantages of *ForeNet*

Despite some limitations of our proposed model, *ForeNet* still shows its strong ability of handling time series prediction problems.

- The proposed prediction method has been applied to predict the time series. From the experimental results, there is no doubt that the network can achieve the predictions with much faster convergence speed. And the initial method of *ForeNet* is quite efficient, which makes the training faster and better generalization.

- The architecture of *ForeNet* is very simple compared to other recurrent models. The self-recurrent links only exist on the hidden unit. Through simple, as a recurrent model, it can also store and process internal memory and do good prediction.

- *ForeNet* is a complex-valued network. With complex values, the neurons can contain more information and may be more stable in training process compared with real-valued network. And trained by complex-value learning algorithm, the complex-valued model outperforms the real-valued model on time series prediction in terms of convergence speed and generalization ability.

- *ForeNet* is derived based on Fourier recursive equation. Therefore, by analyzing the prediction equation, we can understand the *ForeNet* better. In *ForeNet*, the model stores internal memory in the hidden units. And since there is only one input unit in the network, it is much easier to analyze the memory. We have given some implications on the relationship between the internal memory in the model and the embedding dimension of a given time series. And the proper memory is important to the network performance on prediction of time series. In our model, we can adjust the value of coefficients to control the memory. And as we known, usually it is not easy to analyze a recurrent model because of its complexity.

8.3 Future Works

ForeNet was proposed and some of its properties were analyzed in the thesis. However, there are still many interesting topics related to the proposed model.

Multi-step prediction.

All prediction tasks discussed in the previous chapters are single-step prediction. We can extend single-step to multi-step prediction by feeding back the predicted output as input for the next prediction.

An alternative to the iterated single-step prediction is direct multi-step prediction. That means the network is trained to predict directly several step ahead. However, our recursive prediction equation was derived to perform one-step-ahead prediction problem, accordingly the proposed *ForeNet* is just suitable for forecasting the next time step time series. Because of this limitation, problems of multiple-step-ahead forecasting can not be solved directly.

We can approach multi-step-ahead prediction through iterated one-step-ahead predictions. The estimate value is feeded back as input to the network.

$$\hat{x}(t) = f(\hat{x}(t-1)) \tag{8.1}$$

Equation (8.1) can be iterated forward in time to achieve predictions as far into the future as desirable.

Muti-variables

The architecture of *ForeNet* is $1-p-1$. That is, only one current data can be inputted to the network, so the prediction is calculated based upon one input variable, as well the internal memory. As we shown in Chapter (6), AR model with high order significantly outperforms AR model with one order. We may doubt here whether our model is the same case, and whether it can improve performance if the order is increased.

For future work, the single variable may be extended to multi-variables based on the Fourier analysis of time series. In this way, the model becomes high order model just similar with AR model with high order, which can contain more past information by inputting more past data. Thus, the corresponding *ForeNet* network can be updated to $n-p-1$ model, that is, there will have n input units in the network. In this way, we can combine *Time-delay Neural Networks (TDNN)* and current *ForeNet*. In such combined networks, the model memory will be presented by the "external" memory (by the order of input) and the "internal" memory (by the recurrent links on hidden units). This updated model may achieve more powerful computational ability.

More applications

At present, *ForeNet* is only used for time series prediction. We may consider to do some modifications on the model and apply it to perform other tasks, such as classification problem, in the future.

Different learning algorithms in *ForeNet*

Now we use a complex real-time recurrent learning algorithm (CRTRL) to train the weights in *ForeNet*. Some other training methods can be tried. For example, in [65], the author developed the derivative-free *extended kalman filter* for parameter estimations and neural networks training. They mentioned that such forms have better numerical properties and provide similar performance without the need to analytical calculate Jacobians. Similar methods may be considered in our proposed model to estimate the parameters in stead of CRTRL.

Bibliography

[1] H. Akaike. A new look at the statistical model identification. In *IEEE Trans. Automat. Control*, volume AC-19, pages 716–723, 1974.

[2] A. Aussem. Dynamical recurrent neural networks towards prediction and modeling of dynamical systems. In *IEEE Trans. on Neural Networks*, volume 28, pages 207–232, 1999.

[3] A. Aussem and F. Murtagh. Combining neural network forecasts on wavelet-transformed time series. In *Connection Science-special issue on Combining Neural Nets 9*, volume 9, pages 113–121, 1997.

[4] A. Back and A.C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modelling. In *Neural Computation*, volume 3, pages 375–385, 1991.

[5] P.F. Baldi and K. Hornik. Learning in linear neural networks: A survey. In *IEEE Trans. on Neural Networks*, volume 6, pages 837–858, July 1995.

[6] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent network. In *Proceeding of the 1993 IEEE International Conference on Neural Networks*, volume 3, pages 1183–1188, San Francisco, 1993. IEEE Press.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. In *IEEE Trans. On Neural Networks*, volume 5, pages 157–166, 1994.

[8] N. Benvenuto and F. Piazza. On the complex backpropagation algorithm. In *Signal Processing*, volume 40, pages 967–969, 1992.

[9] T. Bitzer and C.W. Omlin. Neural networks for chaotic time series prediction.

[10] Peter Bloomfield. *Fourier Analysis of Time Series: An Introduction.* John Wiley and Sons, Inc., Canada, 1976.

[11] Elizabeth Bradley. Analysis of time series. In Michael Berthold and David J. Hand, editors, *Intelligent Data Analysis:An Introduction*, pages 167–193. Springer-Verlag, 1999.

[12] Lai Wan Chan and Fung Yu Young. Ring-structured recurrent neural network. In *World Congress on Neural Networks 1993, Portland*, volume 4, pages 328–331, 1993.

[13] Lan Wan Chan. Connection reduction of the recurrent networks. In *Proceeding of ICONIP'95, Beijing*, volume 2, pages 813 – 816, 1995.

[14] Chris Chatfield. *The analysis of time series an introduction.* Chapman-Hall, fifth edition, 1996.

[15] Jerome Connor, Les E. Atlas, and Douglas R. Martin. Recurrent networks and NARMA modeling. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 301–308. Morgan Kaufmann Publishers, Inc., 1992.

[16] J.T. Connor, R.D. Martin, and L.E. Atlas. Recurrent neural network and robust time series prediction. In *IEEE Trans. on Neural Networks*, volume 5, pages 240–254, 1994.

[17] J.R. Dickie and A.K. Nandi. A comparative study of ar order selection methods. In *Signal Processing*, pages 259–255, 1994.

[18] G. Dorffner, E. Leitgeb, and H. Koller. Toward improving exercise ecg for detecting ischemic heart disease with recurrent and feedforward neural nets. In et al. J. Vlontzos, editor, *Neural Networks for Signal Processing*, volume 4, pages 499–508, New York, 1994. IEEE.

[19] Georg Dorffner. Neural networks for time series processing. In *Neural Network World*, volume 6, pages 447–468, 1996.

[20] R. Douglas, c. Koch, M. Mahowald, K. Martin, and H. Suarez. Recurrent excitation in neocortical circuits. In *Science*, volume 269, pages 981–985, 1995.

[21] R. Drossu and Z. Obradovic. Efficient design of neural networks for time series prediction.

[22] J.L. Elman. Finding structure in time. In *Cognitive Science*, volume 14, pages 179–211, 1990.

[23] J.T. Connor et al. Recurrent neural networks and robust time series prediction. In *IEEE Trans. on Neural Networks*, volume 5, pages 240–254, 1994.

[24] George M. Georgiou and Cris Koutsougeras. Complex domain backpropagation. In *IEEE Trans. on Circuits and Systems II : Analog and Digital Signal Processing*, volume 39, pages 330–334, 1992.

[25] Neil A. Gershenfeld and Andreas S. Weigend. The future of time series: Learning and understanding. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Predicting the future and understanding the past*, pages 1–70. CA:Addison-Wesley Publishing, 1993.

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

[27] A. Gordon, J.P.H. Steele, and K. Rossmiller. Predicting trajectories using recurrent neural networks. In et al. C.H. Dagli, editor, *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 365–370, New York, 1991. ASME Press.

[28] C. Goutte. Extracting the relevant delays in time series modelling. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, volume 2, 1997.

[29] G.K. Grunwald, R.J. Hyndman, L. Tedesco, and R.L. Tweedie. A unified view of linear ar(1) models. 1997.

[30] R.L.T. Hahnloser. On the piecewise analysis of linear threshold neurons. In *Neural Networks*, volume 11, pages 691–697, 1998.

[31] Simon Haykin. *Neural Networks A Comprehensive Foundation.* Prentice Hall,Inc., second edition, 1999.

[32] J.A. Hertz, R.G. Palmer, and A.S. Krogh. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.

[33] Akira Hirose and Hirofumi Onishi. Proposal of relative-minimization learning for behavior stabilization of complex-valued recurrent neural networks. In *Neurocomputing*, volume 24, pages 163–171, 1999.

[34] Edward Ho and Lai Wan Chan. How to design a connectionist holistic parser? In *Neural Computation*, volume 11, pages 1995–2016, 1999.

[35] S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural Computation*, volume 9, pages 1735–1780, 1997.

[36] S. Hochreiter and J. Schmidhuber. Lstm can solve hard long time lag problems. In M.I. Jordan M.C. Mozer and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 473–479, Cambridge, MA, 1997. MIT Press.

[37] M.I. Jordan. Attractor dynamics and parallelism in a connecctionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546. Erlbaum, 1987.

[38] George Kechriotis and Elias S. Manolakos. Training fully recurrent neural networks with complex weights. In *IEEE Trans. on Circuits and Systems II : Analog and Digital Signal Processing*, volume 41, pages 235–238, 1994.

[39] K. Konstantinides. Threshold bounds in svd and a new iterative algorithm for order selection in ar models. In *IEEE Trans. Signal Process*, volume 39, pages 1218–1221, May 1991.

[40] Stefan C. Kremer. Spatiotemporal connectionist networks:a taxonomy and review. In *Neural Computation*, volume 13, pages 249–306, 2001.

[41] K.J. Lang, A.H.Waibel, and G.E.Hinton. A time-delay neural network architecture for isolated word recognition. In *Neural Networks*, volume 3, pages 23–43, 1990.

[42] Tan Lee, P.C. Ching, and L.W. Chan. Recurrent neural networks for speech modeling and speech recognition. In *ICASSP-95, Proceedings of IEEE Int'l Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3319–3322, 1995.

[43] T. Lin, B. G. Horne, and C. L. Giles. How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies. In *Neural Networks*, volume 11, pages 861–868, 1998.

[44] T. Lin, B. G. Horne, P. Tino, and C. L. Giles. Learning long-term dependencies in NARX recurrent neural networks. In *IEEE Trans. on Neural Networks*, volume 7, pages 1329–1338, November 1996.

[45] T. Lin, B. G. Horne, P. Tiño, and C. L. Giles. Learning long-term dependencies is not as difficult with NARX recurrent neural networks. In M. Mozer D. Touretzdy and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, page 577, Cambridge, MA, 1996. MIT Press.

[46] L. Ljung, editor. *System Identification:Theory for the User.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[47] M.Berthold and D.Hand, editors. *Intelligent Data Analysis: An Introduction.* Springer-Verlag, 1999.

[48] Michael C. Mozer. Neural net architectures for temporal sequence processing. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Predicting the future and understanding the past.* CA:Addison-Wesley Publishing, 1993.

[49] T. Nitta. An extension of the back-propagation algorithm to complex numbers. In *Neural Networks*, volume 10, pages 1391–1415, 1997.

[50] Y.C. Pati and P.S. Krishnaprasad. Analysis and synthesis of feedforward neural networks using discrete affine wavelet transformations. In *IEEE Trans. on Neural Networks*, volume 4, pages 73–85, 1993.

[51] B.A. Pearlmutter. Gradient calculation for dynamic recurrent neural networks: A survey. In *IEEE Trans. on Neural Networks*, volume 6, pages 1212–1228, Sep. 1995.

[52] F.J. Pineda and J.C. Sommerer. Estimating generalized dimensions and choosing time delays: A fast algorithm. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Predicting the future and understanding the past.* CA:Addison-Wesley Publishing, 1993.

[53] D.S.G. Pollock, editor. *A Handbook of Time-Series Analysis, Signal Processing and Dynamics.* ACADEMIC PRESS, 1999.

[54] P.Werbos. Neural networks, system identification and control in the chemical process industries. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligent Control. Neural, Fuzzy, and Adaptive Approaches*, pages 283–356. Van Nostrand Reinhold, 1992.

[55] S.S. Rao and B. Kumthekar. Recurrent wavelet networks. In *1994 IEEE International Conference on Neural Networks. IEEE World Congress on Computational Intelligence*, volume 5, pages 3143–3147, 1994.

[56] Russell D. Reed and Robert J. Marks II. *Neural Smithing:Supervised Learning in Feedforward Artificial Networks.* The MIT Press, 1998.

[57] J. Rissanen. Modelling by shortest data description. In *Automatica*, volume 14, pages 465–471, 1978.

[58] D.A. Robinson. The use of control systems analysis in the neurophysiology of eye movement. In *Annu. Rev. Neurosci.*, volume 4, pages 463–503, 1981.

[59] T. Robinson and F. Fallside. A recurrent error propagation network speech recognition system. In *Computer Speech and Language*, pages 259–274, 1991.

[60] D. Saad. *On-Line Learning in Neural Networks.* Cambridge University Press, Cambridge, 1998.

[61] T. Sauer, J. Yorke, and M. Casdagli. Embedology. In *J. Statist. Phys.*, volume 65, pages 579–616, 1991.

[62] T.Hellstrom and K.Holmstrom. Predicting the stock market. Technical Report IMa-TOM-1997-07, Malardalen University, P.O.Box 883,S-721 23 Vasteras, Sweden, August 1997.

[63] H. Tong. *Non-linear Time Series: A Dynamical Systems Approach.* Oxford University Press, Oxford, 1990.

[64] A.C. Tsoi and A. Back. Locally recurrent globally feedforward networks, a critical review of architectures. In *IEEE Trans. on Neural Networks*, volume 5, pages 229–239, 1994.

[65] R. van der Merwe and E. A. Wan. Efficient derivative-free kalman filters for online learning. In *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, Apr. 2001. To appear.

[66] A. Waibel. Modular construction of time-delay neural networks for speech recognition. In *Neural Computation*, volume 1, pages 39–46, 1989.

[67] Eric Wan. Temporal backpropagation: An efficient algorithm for finite impulse response neural networks. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 131–140, 1990.

[68] Eric Wan. Temporal backpropagation for fir neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 575–580, 1990.

[69] Eric A. Wan. Time series prediction by using a connectionist network with internal delay lines. In Andreas S. Weigend and Neil A. Gershenfeld,

editors, *Predicting the future and understanding the past*, pages 195–217. CA:Addison-Wesley Publishing, 1993.

[70] Andreas S. Weigend and Neil A. Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley Publishing, Redwood City, 1993.

[71] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Predicting the future: a connectionist approach. In *International Journal of Neural Systems*, volume 1, pages 193–209, 1990.

[72] P.J. Werbos. *The roots of backpropagation : from ordered derivatives to neural networks and political forecasting.* John Wiley and Sons, Inc, 1994.

[73] H. Wersing, W.J. Beyn, and H. Ritter. Dynamical stability conditions for recurrent neural networks with unsaturating piecewise linear transfer functions. In *Neural Computation*, 2000. In Press.

[74] R.J. William and J. Peng. An efficient gradient-based algorithm for online training of recurrent network trajectories. In *Neural Computation*, volume 1, pages 490–501, 1990.

[75] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. In *Neural Computation*, volume 1, pages 270–280, 1989.

[76] Richard S. Zemel and Jonathan Pillow. Encoding multiple orientations in a recurrent network. In *Neurocomputing*, volume 32-33, pages 609–616, 2000.

[77] R.S. Zemel, C.K.I. Williams, and M.C. Mozer. Lending direction to neural networks. In *Neural Networks*, volume 8, pages 503–12, 1995.

[78] Q.H. Zhang and A. Benveniste. Wavelet networks. In *IEEE Trans. on Neural Networks*, volume 3, pages 889–898, 1992.

[79] Y.Q. Zhang and L.W. Chan. Forenet: Fourier recurrent networks for time series prediction. In *ICONIP-2000, Proceedings of 7th Int'l Conference on Neural Information Processing*, 2000.