Mirikitani, Derrick Takeshi

Sequential recurrent connectionist algorithms for time series modeling of nonlinear dynamical systems

Sequential Recurrent Connectionist Algorithms for Time Series Modeling of Nonlinear Dynamical Systems

Derrick Takeshi Mirikitani

Goldsmiths College

University of London

A thesis submitted in full requirements for the degree of

Doctor of Philosophy

May 2010

supervised by Dr. Nikolay Nikolaev

Acknowledgements

This Thesis document would not be possible without the valuable discussions I have had with of a number of important and valued people. To these people, I owe my gratitude and thanks.

First of all, and most importantly, I would like to thank my Advisor, Dr. Nikolaev for facilitating my journey and for providing me with valuable advice, guidance and encouragement whilst completing this thesis. Your time and patience are greatly appreciated.

Thanks also to my colleges from Goldsmiths, including Dr. Mohammed Daoudi, Dr. Lahcen Ourabya, and Dr. Saduf Naqvi for their technical expertise and advice. There were two colleagues from outside Goldsmiths that had provided enormous insight into the research process, Dr. Peter Tino from University of Birmingham, and Dr. Richard Twycross-Lewis from Queen Mary, University of London. Our discussions and your comments have been very useful to me.

I am grateful to the Department of Computing of Goldsmiths College, University of London for giving me the opportunity to embark on this project. Goldsmiths has provided an excellent environment for research and I acknowledge with gratitude the help and support provided by its staff.

Last, but far from least, I would like to give my sincere thanks to all of my

family and my friends for their emotional support, patience and understanding. You are all fantastic.

Abstract

This thesis deals with the methodology of building data driven models of nonlinear systems through the framework of dynamic modeling. More specifically this thesis focuses on sequential optimization of nonlinear dynamic models called recurrent neural networks (RNNs). In particular, the thesis considers fully connected recurrent neural networks with one hidden layer of neurons for modeling of nonlinear dynamical systems. The general objective is to improve sequential training of the RNN through sequential second-order methods and to improve generalization of the RNN by regularization. The total contributions of the proposed thesis can be summarized as follows:

1. First, a sequential Bayesian training and regularization strategy for recurrent neural networks based on an extension of the Evidence Framework is developed.
2. Second, an efficient ensemble method for Sequential Monte Carlo filtering is proposed. The methodology allows for efficient $\mathcal{O}(H^2)$ sequential training of the RNN.
3. Last, the Expectation Maximization (EM) framework is proposed for training RNNs sequentially.

Publications

Journal Articles

1. D. T. Mirikitani, and N. Nikolaev, "Recursive Bayesian Recurrent Neural Networks for Time-Series Modeling," *IEEE Transactions on Neural Networks*, vol. 21, no. 2, pp. 262–274, 2010
2. D. T. Mirikitani, and N. Nikolaev, "Efficient online recurrent connectionist learning with the ensemble Kalman filter," *Neurocomputing*, vol. 73, no. 4–6, pp. 1024–1030, 2010
3. D. T. Mirikitani, and N. Nikolaev, "Nonlinear maximum likelihood estimation of electricity spot prices using recurrent neural networks," *Neural Computing and Applications*, DOI: 10.1007/s00521-010-0344-1, 2010

Conference Proceedings

1. D. T. Mirikitani, and N. Nikolaev, "Recursive Bayesian Levenberg-Marquardt Training of Recurrent Neural Networks," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN '07)*, 2007, pp. 1089–1098.
2. D. T. Mirikitani, and N. Nikolaev, "Dynamic Modeling with Ensemble Kalman Filter Trained Recurrent Neural Networks," in *Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications (ICMLA '08)*, 2008, pp. 843–848.
3. D. T. Mirikitani, and N. Nikolaev, "Recurrent Expectation Maximization Neural Modeling," in *Proceedings of the International Conferences on Computational Intelligence for Modelling, Control and Automation (CIMCA'08)*, 2008, pp. 674–679.

4. D. T. Mirikitani, and L. Ouarbya, "Modeling Dst with Recurrent EM Neural Networks," in *Proceedings of the* 19^{th} *International Conference on Artificial Neural Networks (ICANN2009)*, 2009, pp. 975–984.

5. L. Ouarbya, and D. T. Mirikitani, "Modeling Geomagnetospheric disturbances with Sequential Bayesian Recurrent Neural Networks," in *Proceedings of the* 16^{th} *International Conference on Neural Information Processing (ICONIP2009)*, 2009, pp. 975–984.

6. L. Ouarbya, and D. T. Mirikitani, "Sequential Modeling of D_{st} Dynamics with SEEk Trained Recurrent Neural Networks," in *Proceedings of the International Conference on Intelligent Systems, Modelling and Simulation (ISMS '10)*, 2010, pp. 32–37.

7. D. T. Mirikitani, and I. Park, and M. Daoudi, "Dynamic Reconstruction from Noise Contaminated Data with Sparse Bayesian Recurrent Neural Networks," in *Proceedings of the First Asia International Conference on Modelling & Simulation (AMS'07)*, 2007, pp. 409–414.

8. D. T. Mirikitani, "Day Ahead Ocean Swell Forecasting With Recursively Regularized Recurrent Neural Networks," in *Proceedings of the IEEE OCEANS 2007 - Europe*, 2007, pp. 1–4.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Research Questions Addressed in This Thesis 4

1.2.1 Research Objective 4

1.2.2 Scope of the Study 5

1.3 Contributions of the Thesis 5

1.4 Criteria for Success 6

1.5 Sources of Data 7

1.6 Structure of the Study 8

2 Architectural Structures for Time Series Modeling **10**

2.1 Dynamical Systems and Time Series 10

2.2 Linear Filter Models 14

2.3 Artificial Neural Models 16

2.3.1 Background 17

2.3.2 Basic Principles of Neural Networks 18

2.4 Neural Filter Architectures 19

2.4.1 Feed-forward networks 20

2.4.2 Recurrent neural networks 22

2.5 The RNN Structures Featured in this Thesis 26

2.5.1 Elman Recurrent Neural Network 27

2.5.2 Williams and Zipser Recurrent Neural Network 30

3 Optimization Algorithms for Time Series Modeling **33**

3.1 Supervised Learning as Optimization 34

3.2 Derivative Computation 37

3.2.1 Backpropagation Through Time 37
3.2.2 Real Time Recurrent Learning 40
3.2.3 Time Complexity of Derivative Computation 43
3.3 Batch vs Sequential Training 43
3.4 Batch Training Algorithms 45
3.4.1 First-order Batch Procedures 45
3.4.2 Second-Order Batch Methods 46
3.5 Sequential Training Methods 50
3.5.1 First-Order Sequential Methods 51
3.5.2 Second-Order Sequential Methods 52

4 Recursive Recurrent Bayesian Levenberg-Marquardt Learning 54
4.1 Ill-Posed Problems 55
4.2 Regularization in Neural Network Learning 58
4.3 Regularization for Linear Systems 60
4.4 Regularization for Nonlinear Systems 61
4.4.1 Determination of Parameter α 63
4.5 Regularized Bayesian RNN Training 64
4.5.1 Recursive Weight Estimation 66
4.5.2 Recursive Covariance Update 67
4.5.3 Bayesian Hyperparameter Updates 68
4.5.4 Bayesian Network Pruning 69
4.5.5 Bayesian Confidence Interval Estimation 70
4.6 Experimental Results 70
4.6.1 Modeling Chaotic Processes 71
4.6.2 Laser Intensity Series 72
4.6.3 Forecasting Electricity Spot Prices 76
4.7 Conclusion 80

5 Sequential Bayesian RNN Filtering 82
5.1 Sequential Bayesian Inference 84
5.2 The Kalman Filter 87
5.3 EKF Training of Recurrent Neural Networks 90
5.4 Singular Evolutive Extended Kalman filter 92
5.5 EnKF Training of RNNs 94

5.5.1 Computational Complexity of RNN Ensemble Filtering 96
5.6 Experimental Results 99
5.6.1 Modeling the Volume of the Salt Lake 100
5.6.2 Modeling Electricity Spot Prices of the Spanish Market 103
5.7 Conclusion 105

6 Sequential Maximum Likelihood Learning for RNNs 107
6.1 Maximum Likelihood Learning viaExpectation Maximization 108
6.1.1 Maximization Step 113
6.1.2 Expectation Step 115
6.2 Experimental Results 117
6.2.1 Ontario Electricity Market Price Forecasting 119
6.2.2 Forecasting of the Spanish Electricity Market 122
6.3 Conclusion 126

7 Summary and Conclusion 128
7.1 Summary 128
7.2 Future Directions 131
7.3 Concluding Remarks 132

A Derivation of the BPTT Algorithm with Ordered Derivatives 133

B Derivation of the RTRL Algorithm with Ordered Derivatives 136

C Derivation of Bayesian Levenberg-Marquardt Training Rule 139

Bibliography 142

List of Figures

1.1 Modeling Paradigms 3

2.1 Packard-Takens' Reconstruction Framework 12
2.2 Feed Forward Multilayer Perceptron 21
2.3 Local Recurrent Neural Network 23
2.4 Jorden Recurrent Neural Network 25
2.5 Multi-layer Recurrent Neural Network 26
2.6 Elman Recurrent Neural Network 28
2.7 Fully Recurrent Neural Network 31

4.1 Plots of the Henon and Ikeda map 73
4.2 Plots of the Rössler and Lorenz attractors 73
4.3 Laser time series and return map 75
4.4 Forecasts and Confidence Intervals on the Laser Time Series 75
4.5 Plots of the Errors on Laser Time Seeries 78
4.6 In-sample Fit on Nordpool Time Series 78
4.7 Forecast on the Nordpool Time Series 79
4.8 Error Plots on the Nordpool Time Series 79

5.1 EnKF Neurons vs Ensemble Size vs MSE 99
5.2 Convergence performance of EnKF, EKF, and RTRL 101
5.3 EnKF Out-of-sample predictions on the Salt Lake Data Set 102
5.4 Average convergence times on the Spanish electricity spot price data set . 103
5.5 One hour ahead forecasts of the Spanish electricity market 104

6.1 Out-of-sample predictions of the Ontario spot prices 117
6.2 Out-of-sample forecasts of the second segment of the Ontario data set . . 118
6.3 Out-of-sample forecasts on the Spring week of the Spain data set 119

6.4 Convergence and log-likelihood of the evaluated on Ontario spot prices . 120
6.5 Evolution of the model hyper-parameters on Ontario spot prices 122
6.6 Out-of-sample predictions of the Summer Spain spot prices 123
6.7 Out-of-sample predictions of the Fall Spain spot prices 124
6.8 Out-of-sample predictions of the Winter Spain spot prices 125
6.9 convergence and log-likelihood of the model on Spanish spot prices . . . 127
6.10 Evolution of the model hyper-parameters on the Spanish spot prices . . . 127

Chapter 1

Introduction

1.1 Background and Motivation

The majority of naturally occurring systems are highly nonlinear and not fully observable [98]. These systems tend to be formed through the interaction of many decentralized parts whose nonlinear interaction gives rise to what seems to be organized behavior [84]. Systems with these properties are known as complex systems [59, 143]. Predictive models of complex systems can lead to advances in a wide spectrum of applied science and engineering problems [1, 48, 59, 97, 98, 197, 205].

Arguably the best route to constructing a model of a system is through derivation of the equations of motion from first principles [31], because in theory, given the initial conditions, the equations can be solved to forecast future states of the system[1]. Unfortunately when building models of naturally occurring systems, the forward approach is not always feasible as knowledge from first principles of most complex phenomena are rarely available [130]. Furthermore, even if the equations of motion are known, initial conditions may be difficult to collect [19, 98].

In most situations where the equations of motion are unknown (or where initial conditions are unavailable), observations are used to infer a model that describes the process that generated the data [19]. Usually a finite series of observations corresponding to time-dependent events of a dynamical system are used to reconstruct the equations of motion [20]. This approach is called the inverse approach to modeling of nonlinear dynamical systems [19], which is illustrated in Figure 1.1. The origins of this approach can

[1]This approach is known as the forward approach to modeling systems.

be traced back to an experiment which found that the geometry of a systems attractor can be recovered from a time series [158]. It was later proven by Takens [203] that the geometric information recovered from a time series can lead to a topologically equivalent reconstruction of the underlying systems dynamics [79]. These results show that observations from a single variable over time can provide enough information to specify the state of a dynamical system [98]. Once the state is recovered, to complete the model, all that remains is the estimation of the state transition function, and this is where neural networks fit into the picture [2, 19].

Artificial neural networks (namely the Feed-Forward Multi-layered Perceptron) are universal function approximators [32, 55] which have the capacity to learn and generalize from examples [10, 15, 77]. These models have been used for nonlinear modeling tasks such as the reconstruction of attractors [5, 78, 79, 140] and prediction of future states of a dynamical system [37, 58, 60, 106, 221]. Recurrent neural networks (RNNs) are a class of artificial neural networks which has feedback connections that propagate information back in time allowing for the model structure to retain an internal state. Due to its internal state, the RNN is a powerful computational device [89, 194, 195], that is considered to be a dynamical system [210] which is capable of learning temporally extended dependencies [228]. RNNs have been shown to be universal function approximators [41, 185] with the ability to model nonlinear dynamical systems [191]. The RNN has been found to outperform feed-forward neural models on forecasting time series from synthetic and real world data sets [58, 102].

When building a neural based model of a nonlinear dynamical system from data, there are three essential issues that must be considered to achieve accurate predictions:

1. Complexity Control: Complexity control reflects the assumption that simple models with fewer parameters are more likely to generalize better than models with many parameters. This issue of model complexity becomes absolutely central for short and noisy data sets [15, 151].

2. Error Modeling: Noise in a model can be traced to two sources: model mis-specification, and measurement errors. If the model (i.e. the neural network) is mis-specified, there will be a difference between the model and the underlying process (i.e. the true system). This difference is known as process noise. The second type of error originates form the fact that observations are never perfect. The error from obtaining the observation is known as measurement noise. Accounting for these inaccuracies is essential for accurate forecasts [15, 151].

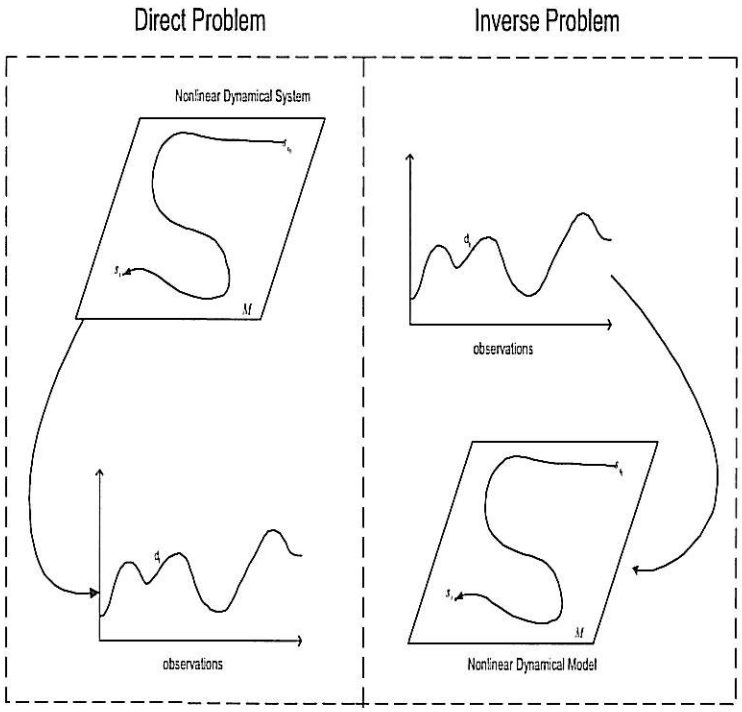

Figure 1.1: Models of dynamic systems can be built in two main ways, the Direct approach and the Inverse approach. The direct approach starts with an understanding of the underlying system. Equations of motion are written down, and the model is verified on the observables of the system. The Inverse approach starts with little or no understanding of the system. Observables of the system are used to reconstruct the equations of motion, and the model is verified on an unseen set of observables.

3. Parameter Estimation: Often times data is not available in batch form, and arrives sequentially. Development of sequential estimation algorithms that can take into account the model, the assumptions on the error, and adjust for the optimal complexity are critical for dynamic modeling [78, 151].

These issues have been addressed in various feed-forward neural time series models [15, 78, 122, 151, 218], but little has been done for their analysis in the case of RNNs.

1.2 Research Questions Addressed in This Thesis

Currently there is no methodology for controlling the functional form, or demanding smoothness from sequentially trained recurrent neural models of nonlinear dynamical systems. It is hypothesized that, *sequential training of recurrent neural networks with non-uniform Bayesian regularization can improve generalization in out-of-sample predictions of nonlinear dynamical systems.*

RNNs are known to be costly to train in a sequential setting. Although there has been research done on reducing the computational complexity of online RNN training algorithms [45, 188, 233], complexity reduction has always come at a tradeoff, i.e. reduction of the predictive ability of the RNN. This thesis attempts to develop *a sequential Monte-Carlo method which reduces the computational complexity of online training of RNNs, and can also lead to improved generalization in out-of-sample predictions of nonlinear dynamical systems.*

During sequential estimation of the RNN weights (parameters), there have been no previous examples of modeling the noise caused by mis-specification of the RNN (process noise), and the noise originating from errors in the measurements (measurement noise). It is hypothesized that, *during sequential training, including information about the characteristics of the process and measurement noise into the training process can improve generalization in out-of-sample predictions of nonlinear dynamical systems.*

1.2.1 Research Objective

The objective of this Thesis is to *improve sequential training of recurrent univariate neural time series models through complexity control, reduction of the computational cost of sequential training, and by inference of the characteristics of data and modeling errors.*

1.2.2 Scope of the Study

This study is limited to the development of sequential recurrent neural time series models which process a single dimensional series of observations which are real numbers, i.e. this thesis is limited to the processing of real valued time series[2]. More specifically, the inputs, outputs, and parameters to the RNN represented by $h(\cdot, \cdot)$ are defined as:

$$y_t = h(\mathbf{u}_t, \mathbf{w}_t) \tag{1.1}$$

where $\mathbf{u}_t \in \mathbb{R}^L$ is the L dimensional input vector to the RNN, $\mathbf{w}_t \in \mathbb{R}^m$ is the m dimensional weight (parameter) vector, and $y_t \in \mathbb{R}^1$ is the one dimensional output of the RNN.

The architectures of the recurrent neural networks are limited to fully connected and Elman RNNs (although the algorithms are general and can be used to train other recurrent connectionist architectures[3]) with a single dimensional output mapped one step ahead into the future.

1.3 Contributions of the Thesis

This thesis extends the literature of RNNs to cover the three areas identified in Section 1.1, and in doing so, the research questions presented in Section 1.2 have been addressed. This has led to the three main contributions of this thesis:

1. Sequential second-order Bayesian regularized RNN training.

   Regularization is known to play an important role in achieving well generalizing recurrent neural models. A recursive algorithm is proposed for sequential RNN training which utilizes Bayesian inference to estimate individualized non-uniform regularization hyperparameters for each weight, and a noise hyperparameter to model uncertainty in the data. These hyperparameters are incorporated into recursive weight and Hessian updates, resulting in a sequential Bayesian version of the Levenberg-Marquardt learning algorithm.

[2]It has not been definitive that symbolic encoding of time series [186] leads to improvement in forecasting, in fact it has been shown otherwise [211].

[3]or even other nonlinear models

2. Computationally efficient $O(H^2)$ online RNN learning.

   Online learning for RNNs is known to be a computationally expensive task. Effort has been focused on devising strategies to reduce the computational burden of online learning for RNNs. However, reduction in computational complexity has come with a tradeoff, degradation in the modeling potential of the network. A Monte Carlo method for online RNN training is proposed that not only reduces the computational complexity of online training, but also provides well generalizing models that outperform (in terms of out-of-sample error measured in terms of rMSE) existing sequential learning algorithms.

3. Sequential Maximum Likelihood RNN hyperparameter estimation.

   Knowledge of the characteristics of the noise contaminating the data and intrinsic to the model can lead to improved RNN parameter estimates. A maximum likelihood framework for RNN training is proposed that allows for computation of these noise characteristics, and provides a mechanism for incorporating this knowledge (of the noise characteristics) back into estimates of the RNN parameters.

1.4 Criteria for Success

In this study as well as in other major studies on time series prediction [217, 218, 220], the validity of the model will be evaluated by the performance of predictions into the future. In this thesis, model performance will be evaluated by the errors the model commits defined by some distance metric between the predicted value and the observed value. The error measures utilized in this thesis have been chosen so that the results are comparable to other studies within the literature.

The experimental tests have been structured to test for any added value of the proposed algorithms against a control group, which consists of the RNN trained without the added improvements (i.e. trained with standard training algorithms), and neural models trained without the recurrent connections, i.e. feed-forward networks[4]. The control group in this study is:

- feed-forward multi layer perceptron trained with the Extended Kalman Filter;

[4]Evaluation of the proposed models against a control group follows standard methodology for experimentalist research in computer science in the sense that the degree of improvement over existing algorithms is demonstrated [190].

- recurrent neural network trained with the RTRL algorithm[5];
- recurrent neural network trained with the Extended Kalman Filter.

Relatively small error values committed by the proposed models, compared to the control group indicate an improvement. The degree of success of this research will be determined by the amount of improvement in prediction accuracy on out-of-sample data over the control group.

1.5 Sources of Data

The main data sets used for evaluation of the performance of the proposed RNNs will be the laser data set from the time series competition data sets provided by Weigend and Gershenfeld [219], along with a variety of chaotic dynamical systems such as the Henon map [81] and the Lorenz differential equations [117]. These data sets are regarded as benchmark data sets in the nonlinear time series literature and will provide a means of comparison to other studies.

Also, more complicated "real world" data sets are used to evaluate the proposed models. Real world data has been chosen as it serves two main purposes; 1) it serves as means to validate the proposed models on a complex task, and 2) it relates the work reported in this thesis to the wider field of modeling real world complex systems. The real world data sets (i.e. the lake volume time series and electricity spot price time series) considered in this thesis are related to the following real world problems:

1. Fluctuations in Large Scale Spatiotemproal Climate Dynamics: Modeling climatic dynamics has become a topic of increasing importance. However, modeling climate dynamics is known to be a difficult problem due to the nonlinearities and chaotic properties of climatological processes. This thesis evaluates RNNs on one specific chaotic climatological process, the change in the volume of the Salt Lake in Utah, USA [1]. The volume of the Salt Lake serves as a proxy of climate change over a wide area (of mountainous regions surrounding the lake) through time [3].

2. Nonlinearities in Deregulated Power Exchanges: Accurate forecast models of electricity spot price dynamics can assist market participants in making informed decisions on the production and supply of power [225]. This can lead to efficient

[5]This study is focused on sequential training algorithms, so global search strategies such as evolutionary algorithms, etc. are not considered

operation of power markets. However, the spot price of electricity is known to be influenced by climatic dynamics [225], which result in highly nonlinear behavior that is difficult to model [13]. Three different power exchanges are used throughout the thesis as each power exchange has its own characteristics; the Nordic Power Exchange, the Spanish Power Exchange and the Ontario Power Exchange.

1.6 Structure of the Study

- Chapter 2 discusses architectural structures proposed in the literature used for approximation of the state transition function. Both linear and nonlinear models are discussed and their strengths and weaknesses are evaluated. Details of neural networks are discussed including their architecture and approximation ability.

- Chapter 3 discusses learning algorithms for training the architectures discussed in Chapter 2. Derivative computation for RNNs are reviewed, and optimization schemes based on first-order and second-order information are discussed.

- Although RNNs have the ability to estimate a dynamical system, in most cases the problem is ill-posed. Chapter 4 discusses recurrent neural networks in the framework of inverse problems. Ill-posed problems are focused on and the solutions around ill-posed problems via regularization are emphasized. A sequential solution to the ill-posed estimation problem, the recursive Bayesian Levenberg-Marquardt training algorithm is presented. Finally the performance of the proposed recursive Bayesian Levenberg-Marquardt RNN training algorithm is evaluated on various data sets.

- Chapter 5 addresses recursive Bayesian state estimation and provides a framework to estimate the weights of a recurrent neural network. This framework implements regularization as it is a suboptimal method (unlikely to overfit) and minimizes a regularized cost function. A novel method to reduce computational complexity of RNN training and increase prediction accuracy of the RNN over well known Kalman based training methods is discussed. The model is validated on various time series forecasting experiments.

- Chapter 6 provides a Maximum Likelihood framework for estimation of model hyperparameters which account for the uncertainty in the model and the belief in the

data. This is achieved through an Expectation Maximization algorithm which uses nonlinear sequential Bayesian filtering and smoothing in the expectation step, and solves for model hyperparameters in the maximization step. The proposed model is compared to previously proposed models on various time series data sets.

- Chapter 7 reviews the work presented in this document. It begins with a summary of the main points of the thesis. Then the contributions are restated, and the limitations to the research are discussed. Future directions for further research are proposed, and finally concluding remarks are provided.

Chapter 2

Architectural Structures for Time Series Modeling

There are two main approaches to building a model of a dynamical system; the direct method and the indirect method. The direct approach to modeling assumes a deep understanding from first principles of the system so that equations of motion can be written down. When less is known about the system, the inverse approach is usually taken. The inverse approach assumes the existence of a "teacher function" [161] which has given rise to the observations. This Chapter provides a review of various time series model structures that can represent various "teacher functions." The final section of the chapter provides a description of the RNN architectures featured in this Thesis.

2.1 Dynamical Systems and Time Series

The effort to understand complex systems has led to an awareness that understanding the constituent parts of systems in nature does not necessarily lead to an understanding of the global dynamics of the system [84]. This has encouraged a shift from the reductionist approach of understanding through compartmentalization and partitioning to a more holistic perspective (macro view) of systems. Modeling plays an important role in the study of such complex systems. In one of his plenary talks, Professor John Holland had classified modeling into three main categories: Exploratory, Existence Proofs, and Data Driven. Exploratory modeling seeks to gain understanding of complex systems through simula-

tion in hopes of understanding relevant stylized facts and extracting the basic principles of what has led to the observed complexity. Building models of complex systems can also be used to provide a proof of existence of a certain concept. For example Von Neumanns' self replicating automata provided a proof of concept of reproduction in non-biological machines (which subsequently changed how life is viewed). The last type of modeling is known as data driven modeling, and is the focus of this theses. Data driven modeling aims to uncover the general dynamical rules that gave rise to the observed phenomena.

Dynamic modeling is a data driven method for modeling the temporal dynamics of systems. The general aim of dynamic modeling is to infer the mapping $\mathbf{F} : \mathbb{R}^{\mathrm{d}} \rightarrow \mathbb{R}^{\mathrm{d}}$ of the unknown dynamical system of dimensionality d. Once the governing dynamics of the system are recovered, they can be used for understanding of the system, forecasting future states, finding fixed points, etc. In the following chapters, this thesis reviews previous work and standard algorithms for "recovering" governing dynamics from data, as well as proposes several new methods and refinements of established techniques.

The concept of dynamical systems can trace its roots far back in the ancient literature, however concrete mathematical descriptions are generally found in Newtonian mechanics. The evolution rule of the dynamical system is usually described by a differential equation or difference equation which determines the state for all future times of the system given an initial condition.

The dynamical system itself is usually a collection of decentralized parts which directly (and/or indirectly) influence the system state through time. The change in the collection of physical mechanisms over time can be described by a system of ordinary differential equations which are referred to as the system *state equations*

$$\frac{d}{dt}\Xi(t) = \mathbf{F}(\Xi(t)) \tag{2.1}$$

where $\Xi(t)$ is the system state (i.e. system state vector) of dimension d at time t and $\mathbf{F}$ is a smooth vector field that is diffeomorphic, i.e. $\mathcal{C}^r$ (continuously differentiable r times). The function ϕ is a solution to the system of differential equations in Equation 2.1, and the inverse function ϕ^{-1} is assumed to exist. Given that the closed form solution to Equation 2.1 exists, a trajectory in state space is defined as $\phi^{(0:t)}(\Xi(0))$. Here the governing rules of the system are assumed to be describable by a set of deterministic mathematical equations. It is assumed that the governing dynamics of all systems under consideration in this thesis are deterministic in nature. Some systems operate without

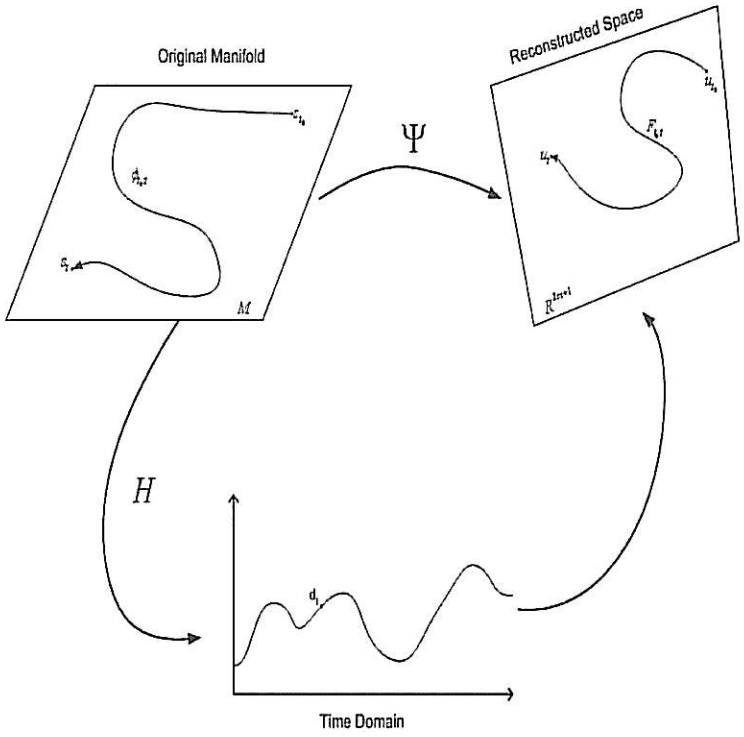

Figure 2.1: A diagram of the reconstruction framework proposed by Packard et al. [158] and proven by Takens' [203].

inputs, i.e. where the observable is generated though the system dynamics. These types of systems will be referred to as output systems. All unknown systems in this thesis are considered to be output systems. The current work is focused on modeling output systems which generate a scalar output signal. Usually the entire system state is not observable and in the worst case only a one dimensional scalar measurement of the system state may be available. It is assumed that the observations are related to the state via the *measurement function*:

$$d(t) = \mathrm{H}(\Xi(t)) \tag{2.2}$$

which is a function $\mathrm{H} : M \rightarrow \mathbb{R}^1$ that maps the manifold M to the observed sequence of numbers $\{d(t)\}$, where $\Xi(t)$ is the state.

As many systems are continuous in nature the output signal $d(t)$ will usually be analog (i.e., continuous). Discretization of the system output via sampling is usually necessary to obtain the time series. Discretization of the equations of motion leads to the approxi-

mation

$$\frac{d}{dt}\Xi(t) \cong \frac{\Xi(t+\kappa) - \Xi(t)}{\kappa} \tag{2.3}$$

where κ is the uniform sampling period [1]. With this approximation, the state at time $t\kappa$ can be written as $\Xi(t\kappa) = \Xi_t$, which implies that the time evolution of the state can be written as a discrete sequence $\Xi_0, \Xi_1, \ldots, \Xi_t$. The equations of motion are now written as a difference equation [1]

$$\Xi_{t+1} = \Xi_t + \kappa \mathbf{F}(\Xi_t) \tag{2.4}$$

which provides discrete time governing dynamics of the system state. To obtain the measurements, if follows that the measurement operator is discritized as [1]:

$$d_t = \mathrm{H}(\Xi_t) \tag{2.5}$$

which results in the observed time series of sampled data

$$\ldots, d_{t-2}, d_{t-1}, d_t, d_{t+1}, d_{t+2}, \ldots \tag{2.6}$$

When building a model of a system from observed data, it is usually the case that prior knowledge about the underlying physics of the system is limited or nonexistent *a priori*. If this is the case, then the model must be built from the data alone. This type of modeling is known as black-box modeling [114]. The first time series model built in this manor can be traced to back to Yule in 1927 [231]. Yule built a model of the dynamics of sun spot activity from a lagged vector of past measurements $\mathbf{u}_t$. Yule treated the system generating the sun spots as an output system in which a delayed vector of system outputs are defined as

$$\mathbf{u}_t = [d_{t-1}, d_{t-2}, \ldots, d_{t-L}]^{\mathrm{T}} \tag{2.7}$$

where L is the dimensionality of the lagged vector $\mathbf{u}_t$. This lagged vector is often also referred to as a tapped delay line in the neural networks literature and a delay vector in the dynamics community.

As Yule was interested in building a predictive model of the sun spot number, he proposed a model of the future number of sun spots y_t as a function of previous measurements

$$y_t = h(\mathbf{w}, \mathbf{u}_t) = h(\mathbf{w}, [d_{t-1}, d_{t-2}, \ldots, d_{t-L}]^{\mathrm{T}}) \tag{2.8}$$

where h is represents the hypothesis of the underlying system dynamics, and $\mathbf{w}$ is the parameters of the model. It was shown that this approach is quite effective and can lead to accurate predictions [220]. However, it was not until the 1980's that the approach was fully justified from a physical perspective [158, 203]. The remainder of this chapter reviews popular time series models. In the next section important linear models are discussed.

2.2 Linear Filter Models

The most simple relationship between previous observations and future values is one of linearity. The finite impulse response (FIR) model [114] is the most basic autoregressive linear time series model. The output d_t is a linear combination of weighted previous observations,

$$d_t = \sum_{k=1}^{P} \varsigma_k u_{t-k} + \varsigma_0 + \epsilon_t \tag{2.9}$$

where $\mathbf{w} = (\varsigma_0, \varsigma_1, \ldots, \varsigma_P)$ are the weights/parameters of the model which are multiplied to their L corresponding previous lagged observations, and ς_0 is the bias term. It is assumed that the error term ϵ_t is Gaussian distributed $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ which represents the uncorrelated zero-mean noise in the measurements. The output of the model y is linearly dependent on the P previous inputs where is P is called the filter order. The optimal forecast for the FIR model in the sense of the mean squared error measure is given by

$$y_t = \sum_{k=1}^{P} \varsigma_k u_{t-k} + \varsigma_0 \tag{2.10}$$

which is a function of the previous $\mathbf{u}_t^e = [u_{t-1}, u_{t-1}, \ldots, u_{t-P}]$ lagged exogenous inputs and the model weights.

The finite impulse response model is a memoryless model (apart from the lagged inputs $\mathbf{u}_t^e$) that regresses the exogenous inputs $\mathbf{u}_t^e$ onto the target d_t. Autoregressive modeling takes a different approach of regressing future target values onto past target values $\mathbf{u}_t = [d_{t-1}, d_{t-2}, \ldots, d_{t-L}]$ [73], described by the equation below:

$$d_t = \sum_{k=1}^{L} \phi_k d_{t-k} + \phi_0 + \epsilon_t \tag{2.11}$$

where $\mathbf{w} = (\phi_0, \ldots, \phi_P)$ are the model parameters. The model error ϵ_t represents Gaussian noise caused by the discrepancy between the underlying system dynamic and the linear autoregression. The forecast d_t for the autoregressive model as discussed in [114] is given by

$$y_t = \sum_{k=1}^{L} \phi_k d_{t-k} + \phi_0 \tag{2.12}$$

The autoregressive model can be traced back to some of the first work in time series analysis proposed by Yule in 1927 [231].

Forecasts can usually be improved by combining both of these structures to form the linear autoregressive model with exogenous inputs (ARX).

$$d_t = \sum_{j=0}^{P} \varsigma_j u_{t-j} + \sum_{k=0}^{L} \phi_k d_{t-k} + \phi_0 + \epsilon_t \tag{2.13}$$

in which the the optimal forecast is given by

$$y_t = \sum_{j=1}^{P} \varsigma_j u_{t-j} + \sum_{k=0}^{L} \phi_k d_{t-k} + \phi_0 \tag{2.14}$$

where the weights are a combined parameter vector $\mathbf{w} = (\varsigma_1, \ldots, \varsigma_P, \phi_0, \ldots, \phi_L)$. This type of model structure has been shown in [116], to be capable of describing any linear system with exogenous inputs, and thus the model is usually considered to be a "complete" model for linear systems [114]. However, the model is not without drawbacks, one of which is that the number of parameters $L+P$ may need to be chosen larger than the dynamics of the underlying system. Numerous extensions of this model structure have been proposed. The most well known extensions of these models is the linear autoregressive moving average model with exogenous inputs (ARMAX).

$$d_t = \sum_{j=1}^{P} \varsigma_j u_{t-j} + \sum_{k=1}^{L} \phi_k d_{t-k} + \sum_{l=1}^{q} \varrho_q \epsilon_{t-q} + \epsilon_t \tag{2.15}$$

where the parameter vector is $\mathbf{w} = (\varsigma_1, \ldots, \varsigma_P, \phi_0, \ldots, \phi_L, \varrho_1, \ldots, \varrho_q)$. The following

recursive algorithm approximately computes the optimal forecast for the ARMAX model,

$$y_t = \sum_{j=1}^{P} \varsigma_j u_{t-j} + \sum_{k=1}^{L} \phi_k d_{t-k} + \sum_{l=1}^{q} \varrho_q \hat{\epsilon}_{t-q} \tag{2.16}$$

where

$$\hat{\epsilon}_{t-k} = d_{t-1} - y_{t-k}, \qquad l = 1, \ldots, q \tag{2.17}$$

The ARMAX model has been popular in the areas of time series modeling and signal processing. Its popularity can be attributed to the relative simplicity of the parameter estimation algorithms and the straightforward statistical analysis of the model.

Although linear models have been popular in the literature due to their well understood properties, and straightforward implementation, the linear model is not always suitable for describing the dynamics of many systems. Most systems that occur in nature contain inherently nonlinearity; application of linear models are not likely to capture the essential dynamics of nonlinear systems [220] and they may be entirely inappropriate for even slightly nonlinear tasks [1, 98].

2.3 Artificial Neural Models

Neural networks are adaptive nonlinear models that may be trained to perform sequence processing tasks from examples. Feed forward neural networks such as multilayer perceptrons, various recurrent neural networks, and some radial basis function networks, possess universal function approximation ability [32, 55, 86, 160, 168], and are considered to be part of a general class of universal function approximators. In practice neural networks are relatively robust to outliers and are capable of fitting highly nonlinear data quite well. When constructing models of nonlinear dynamic systems, neural networks are often chosen because in recent years experimental results have shown that they outperform other nonlinear models in time series forecasting of nonlinear dynamical systems [58, 106, 220].

Recurrent neural networks have a very important advantage over feed forward networks and other models in that they firstly, possess the same adaptability as feed forward networks [200], and secondly, they are able to approximate any smooth dynamical system arbitrarily well [41, 185]. This is a significant improvement over just function approxi-

mation in that the network has the potential to become an implicit model of the dynamic system under consideration [30].

Neural networks are known to be powerful, flexible, and robust nonlinear models amenable to be trained from examples, and able to show good generalization performance and predictive power [15, 78]. This chapter provides a background of neural networks and then goes on to review important neural time series model structures including feed-forward and recurrent neural architectures.

2.3.1 Background

Artificial Neural Networks are biologically inspired models of computation which originated with the work of McCulloch and Pits [129]. Neural physiology was the original motivation behind artificial neural modeling, however the properties of neural models have become highly abstracted from biological reality. Although the field of neural computation has a long interdisciplinary history, there are two general streams of neural network research, one is to understand the physical and/or phenomenological properties of the brain and/or mind, and the other is for machine learning which is the focus of this thesis.

The term connectionism is widely used in the neural network literature. In this thesis connectionism shall be used to define the broadest area of connectionist models in the sense of Farmer [49], who developed a lattice theoretic framework relating various connectionist learning systems such as neural networks, immune systems, autocatalitic networks, and classifier systems. Also, in this thesis, the terms "neural networks" or "artificial neural networks" will refer to models based upon the abstracted functional components and operational principles of biological neural networks without reference to cognitive models.

Early research in artificial neural network models began in the 1940's with McCulloch and Pitts's mathematical representation of a biological neuron as a simple binary switch [129]. Building on McCulloch and Pitts work, Donald Hebb in 1949, [80] suggested that neural pathways are strengthened each time they are used, which led to the first learning algorithm for neural networks. The first computational simulation of a neural network was done at IBM by Nathanial Rochester [10]. Frank Rosenblatt began working on the underlying processing of the fly's visual system in 1950's. This work eventually led to the Perceptron as a model for visual pattern recognition [179]. At this point neural network research began to become very popular leading to models such as

the ADALINE (adaptive linear neuron) and MADALINE (multiple adaptive linear elements) by Widrow and Hoff [226]. The MADALINE was the first neural network to be applied to a real world problem and was used in echo cancelation. However, Minsky and Papert pointed out some of the main limitations of neural network research at that time, including the work of Rosenblatt's simple artificial neural network models [132]. With many yet unsolved issues such as the credit assignment problem, interest and funding in neurocomputing was greatly reduced. A new wave of interest in neural networks began in the mid 1980's mainly as a result of a the popularization of Hopfields work to solve optimization problems using feedback networks [85] and the rediscovery of the Back-Propagation algorithm, used to train multi-layer feedforward networks, by Rumelhart, Hinton and Williams [181, 182] (originally developed by Werbos [223]). Also in the late 1980's computational power became cheap and ubiquitous allowing many researchers to run simulations of artificial neural networks.

2.3.2 Basic Principles of Neural Networks

Following the principles Rumelhart and McCelland in their overview of neural information processing theory, connectionist theory is presented by stating the functional components and operating principles of a connectionist system. Here the outline roughly follows that of Rumelhart and McClelland [183] in which a connectionist model is defined by combining specific functional components and operating principles.

- A set of active processing units denoted by $g(\cdot)$ or $f(\cdot)$ which define how the activities of each unit are updated;
- A state of activation denoted by s;
- An output function for each unit denoted by $\sigma(\cdot)$;
- A pattern of connectivity or topology of the network;
- A propagation rule for propagating the activities of the units through the network which we denote by $h(\cdot)$;
- A learning rule to adapt network connectivity based on interaction with the external environment;

- An external environment that supplies information $\mathbf{u}$ to the network, and a place for the system's response y to take place;

Here the function $g(\cdot)$ refers to the dynamic that governs the hidden layer neurons in the network (this thesis exclusively considers single hidden layer architectures, although multiple hidden layers can be added without complication). The output layer is governed by the function $f(\cdot)$. The activation function $\sigma(\cdot)$ is usually nonlinear, real-valued, monotonically increasing, differentiable, and bounded from above and below. The common choices for the activation function are of the logistical sigmoid family.

The principles presented above, can give rise to a large number of connectionist models including those for temporal pattern recognition. As this thesis focuses on a specific instance of neural networks, namely the recurrent neural network, the basic architecture of the multilayer perceptron and its extension, the recurrent multilayer perceptron, are discussed in the next section.

2.4 Neural Filter Architectures

Temporal pattern processing involves the tracking of variations in a signal that may or may not be stable over time. Due to time dependencies, past and present information may be useful in determining an appropriate future response of the system. Memory plays a key role in temporal sequence processing and thus architectural considerations specifying network topology and the activation dynamics of the processing elements is essential to dealing with temporal patterns. ANNs are classified into two groups based on the directional flow of information within the network. The two groups are feedforward and recurrent models. In a feedforward network, a processing element can send information only to units which it does not receive information from directly or indirectly [179] resulting in a static mapping from its input to its output space. On the other hand, recurrent networks [44] fold the output of a unit back in time creating a circular flow of information which can be modeled as a nonlinear dynamical system [21, 209]. Recurrent networks are particularly well suited for temporal tasks due to their intrinsic memory resulting from the configuration of the network whereas feed forward networks are not specifically designed for temporal processing because of their lack of internal memory. Sliding windows or time delays $\mathbf{u}_t$ must be added to allow for the tracking of time dependencies. RNNs do not need the sliding window which reduces the number of inputs and parameters into the

network speeding up training and reducing the number of free parameters. Without the use of a window, the size and lag of the window do not need to be chosen.

2.4.1 Feed-forward networks

The Feed-forward Multi-layered Precetpron (MLP) shown in Figure 2.2 is the most widespread neural network model in both research and industry [131]. Feed-forward neural networks are universal function approximators [15, 32, 55] which make them powerful structures for nonlinear modeling. The theorem of universal function approximation for the MLP justifies the use of the MLP in time series modeling tasks. The main points behind the MLP universal approximation theorem are that a single hidden layer of neurons with sigmoidal activations and a linear output neuron has the ability to approximate any continuous function to arbitrary accuracy, assuming that enough hidden neurons are available.

In feedforward networks information flows in one direction from the input layer to the output layer. In each layer, there is a row of neurons, and between the layers is a connection represented by an adaptable weight $w^{(i,j)}$. Each neuron combines all of the input signals coming into the unit along with a bias value into a weighted sum and computes some function of the weighted sum. The neuron activation is a function of the weighted sum passed through an activation function. The activation is then broadcast to other units in higher layers, or as the output of the network. The neurons in Feed-forward networks are organized into layers, namely the input, hidden and output. Between layers the processing elements are usually fully connected to the units in the next layer. The jth hidden neuron $g^{(j)}(\mathbf{u}_t, \mathbf{w}^H)$ can be represented mathematically as follows:

$$\acute{s}^{(j)} = \sum_{k=1}^{L} w^{(j,k)} d_{t-k} + w^{(j,b)} \tag{2.18}$$

where $w^{(j,b)}$ denotes the bias of the jth neuron in the hidden layer, and $\mathbf{u}_t = (d_{t-1}, \ldots, d_{t-L})$ denotes the inputs at time t. The jth hidden unit output activation $s^{(j)}$ is computed by

$$s^{(j)} = g^{(j)}(\mathbf{u}_t, \mathbf{w}^H) = \sigma(\acute{s}^{(j)}) \tag{2.19}$$

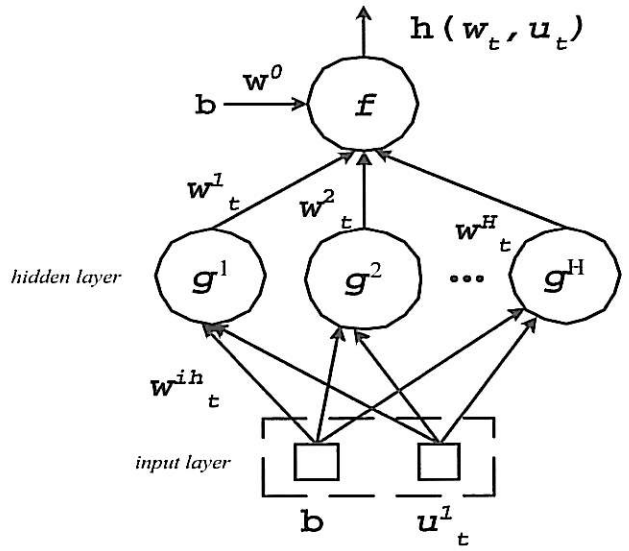

Figure 2.2: Feed Forward Artificial Neural Network: circles represent neurons, and the squares represent the input and the bias

The output of the network y_t computed as

$$y_t = \sum_{j=1}^{H} w^{(o,j)} s^{(j)} + w^{(o,b)} \tag{2.20}$$

For auto-regressive problems a single output unit produces the network output y_t.

One of the main limitations of the feed-forward architecture for time series processing is that the information flow is one directional, from the input to the output (i.e. the network is static). To process temporal data, the feed-forward architecture must be augmented in order to make the "memory" structure of the model more suitable for temporal processing. The literature points to three ways in which this can be achieved

- explicit unfolding of the time domain
- short term memory
- recurrent connections

The simplest way to extend a static neural model is by feeding time directly into the network along with the inputs [100]. A more conventional method of accommodating time into feed forward architecture is through a delay space embedding, also known as a sliding window or tapped delay line. The sliding window widens the input space and orders information from the present to a pre-specified distance L back in time $\mathbf{u}_t = [d_{t-1}, d_{t-2}, \ldots, d_{t-L}]$. At each time step the sliding window discretely shifts over the data series, including the data point of the next time step and removing the last data point from the window [106].

However, feed-forward networks configured in time-delay form have two drawbacks:

- they impose strong limits on the duration of the temporal events;
- they face difficulties in capturing long-term time-varying relationships in the data.

These limitations have been addressed through the recurrent neural network architecture.

2.4.2 Recurrent neural networks

Since feed-forward networks have no implicit memory structure, a delay window is usually added to allow temporal patterns to be processed. This augmentation of the network then makes it necessary to determine the parameters for the lag space before modeling begins. This process is a pre-modeling process as the number of previous observations along with the delay time between the observations need to be estimated. It would be so much more of an advantage if the process of searching for the optimal embedding could somehow be avoided or incorporated into the process of training the network [199].

Recurrent Neural Networks are biased towards learning patterns which occur in temporal order and thus directly address the temporal relationship of their inputs by maintaining an internal state. Recurrent connections in the network allow feedback of information in time. In principle, the memory of a recurrent network is unlimited, but in practice vanishing gradients, and finite precision hardware can severely limit the amount of memory the RNN holds [14]. However, the depth of the memory of RNNs is generally greater than a properly estimated external delay vector, so it is common practice to include only a single exogenous input to the RNN [199].

While the set of topologies of feedforward networks are fairly limited, an RNN can take on any arbitrary topology as any node in the network may be linked with any other

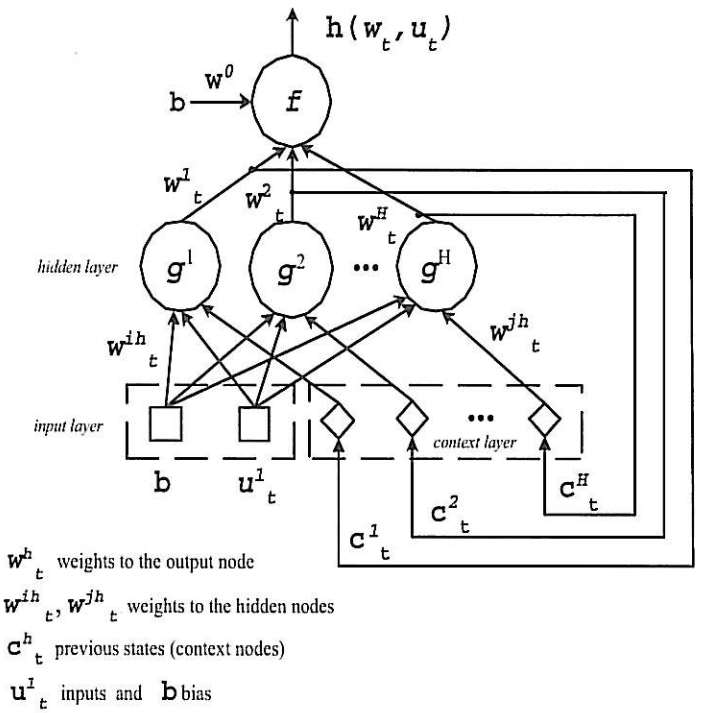

Figure 2.3: Locally Connected Recurrent Artificial Neural Network: circles represent neurons, the two squares represent the input and the bias, and the diamonds represent the context nodes

node (including itself). A number of different discrete-time recurrent neural network architectures have appeared in the literature [147, 172, 214, 230]. The feedback connections considered in this work originate from the output of some neuron, and feed back to the input of the same or some other neuron. The feedback is always passed through the non-linear activation function, which guarantees that the output of some neuron stays bounded.

One of the most elementary recurrent architectures has feedback connections which only feed back locally to the same neuron that originated the signal. This type of feedback connection is known as local feedback or local recurrence. In the *local recurrent network* (Figure 2.3) the hidden layer of a multi-layered preceptron is augmented with local feed back connections [54]. These local feedback connections change the feed-forward MLP to a dynamic system. With the feedback connections the network out-

put is a function of the exogenous inputs $\mathbf{u}_t$, along with the values of the context layer $\mathbf{c}_t = [c^{(1)}, c^{(2)}, \ldots, c^{(k)}, \ldots, c^{(H)}] = \mathbf{s}_{t-1}$ which are the previous values of the hidden layer activations. The context vector of the RNN is the structure by which temporal memory is maintained. The local feedback structure limits the connectivity of each neuron to it self, and no "cross talk" between hidden units takes place. The advantage of a locally connected network generally lies in the simplicity of implementation. Simple Backpropagation have been used to compute the gradients of this type of network [50]. The use of locally recurrent networks, in terms of modeling potential, is limited as there is less connectivity in the hidden layer. The lack of connections between neurons in the hidden layer limits what the network can compute, i.e., the local recurrent network can not learn to mimic a shift register. This has major implications for time series modeling, as the model will need a lag vector for the inputs. A more complex model is obtained by adding global feedback connections to the locally recurrent network. Global feedback connections are feedback links connecting every hidden unit to every other hidden unit. With a global feedback architecture, the hidden layer of the network is fully connected to the outputs of every hidden unit at the last time step. This type of network is known as *globally recurrent networks*.

As globally recurrent networks have much more connectivity in the hidden layer, their modeling potential is much greater than the locally recurrent networks. Much work has been invested into the computational capabilities of the globally recurrent network architecture. It was found that globally recurrent networks with a single hidden layer have the computational capability to represent an arbitrary nonlinear dynamical system. The proof of this has first appeared in [191], which showed that globally recurrent networks can model a smooth dynamical system arbitrarily well over any fixed length of finite time. The proof was based on an extension of work from the feed-forward literature [32, 55, 86] which showed that feed-forward networks can have universal function approximation capability. By removing all feedback connections in the hidden layer and adding a feedback connection flowing from the output unit back to the hidden units, an output recurrent network is created. These output recurrent networks are also called *Jordan networks* [94] which are illustrated in Figure 2.4. It was proven in [71] that an output recurrent network is capable of emulating a global recurrent network. An output recurrent network with $H + 1$ neurons in the hidden layer and an output with a history of $2H$ delayed output values, can be computationally equivalent to a globally recurrent network with H hidden units. However, these capabilities of emulating globally recurrent networks do not come for free. The output feedback network is subject to what is termed a linear slowdown [71].

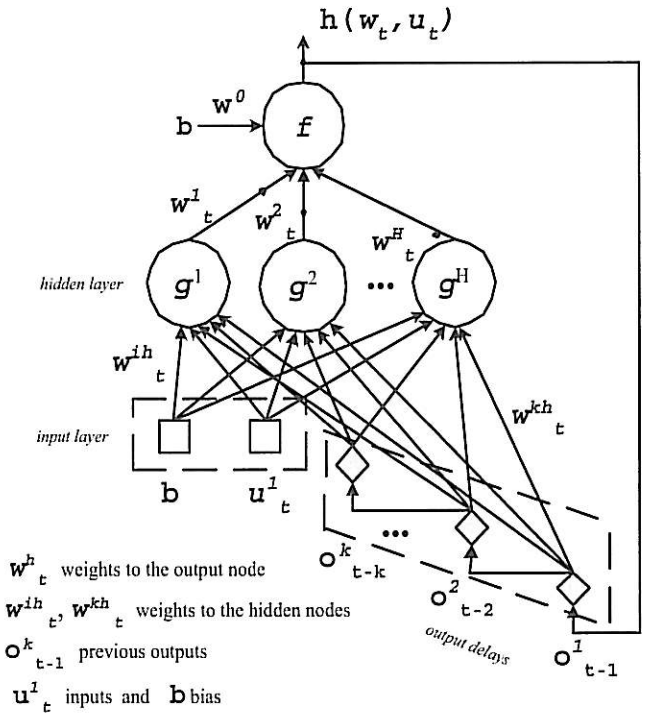

Figure 2.4: Jorden Connected Recurrent Artificial Neural Network: circles represent neurons, the two squares represent the input and bias, and the diamonds represent the context nodes

For emulation of one time step of the globally recurrent network with hidden layer of size H, the output feedback network must be iterated $H+1$ times holding the inputs constant. As output recurrent networks are able to emulate a globally recurrent network, it follows that output recurrent networks share the same modeling capabilities as global recurrent networks. This implies that the output recurrent network can model smooth dynamical systems arbitrarily well.

The previously discussed recurrent network architectures all had a single layer of hidden recurrent connections. However it is not necessary to restrict network architectures to one recurrent hidden layer. Adding hidden layers may allow the RNN to model more complex mappings compared to a model with the same amount of weights with only a single hidden layer. These *multiple hidden layer RNNs* (Figure 2.5) have been reported to work well on complex tasks such as reconstruction of nonlinear dynamics from noisy

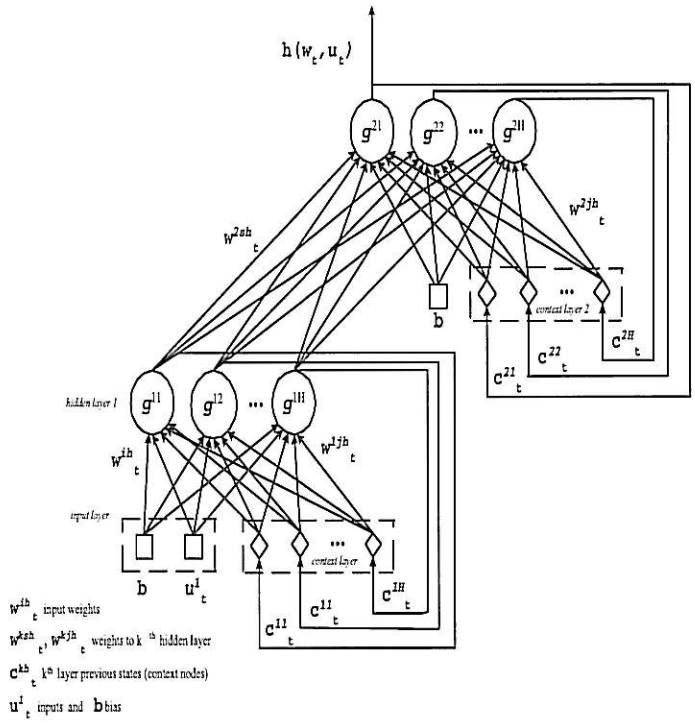

Figure 2.5: Multiple Layer Fully Connected Recurrent Artificial Neural Network: circles represent neurons, square represents the input or the bias, and the diamonds represent the context nodes.

data [78] and control of automotive systems [172]. By adding additional recurrent hidden layers, the possibilities of design choices increases. Various connectivity patterns, i.e., locally recurrent, globally recurrent, are then possible within each layer as well as variations on connectivity between layers.

2.5 The RNN Structures Featured in this Thesis

This section describes the most popular RNN architectures; the Elman [44], and the Williams and Zipser [228] recurrent neural architectures. The common thread between these networks is that both models have globally recurrent connections where each hidden unit input is fully connected to each unit in the context layer. These two RNN model structures are focused on as they are well known in the literature [50, 77, 78] and have

been shown to have universal function approximation capabilities [191].

2.5.1 Elman Recurrent Neural Network

The *Elman network* was originally used for speech processing by Robinson and Fallside [177]. The structure of this network is identical to the feed forward MLP, but with one exception, a context layer. The context layer takes the state information from the hidden activations and feeds it back in to the hidden layer one time step later. The context layer is treated in the same way as the input layer is treated in the feed forward multi layer perceptron; that is, the context layer has weights connecting it to the hidden layer.

Elman networks have been particularly popular for time series prediction [6, 26, 118, 119, 123, 159] due to their approximation capabilities [41]. Theoretically, the RNN has the capacity to represent an arbitrary dynamic system with only a single exogenous input, thus allowing for processing of temporal patterns without an externally provided lag space, i.e., time is implicit in the model. Empirical studies have found that the Elman network provides increased predictive capabilities over feed-forward neural models on synthetic and real world temporal data modeling tasks [58, 102].

The operation of the k-th network node at time t from a dynamic perspective is described by a system of difference equations given by:

$$s^{(k)} = g^{(k)}(\mathbf{u}_t, \mathbf{s}_{t-1}) \tag{2.21}$$

The neuron activation $g(\cdot, \cdot)$ is a nonlinear function of the input $\mathbf{u}_t$ and $\mathbf{s}_{t-1}$, the vector of all neuron activations of the last time step (i.e. the context layer). The output of the network is given by

$$y_t = \sigma(f(\mathbf{s}_t)) \tag{2.22}$$

The state vector $\mathbf{s}_t$ contains the state of each hidden neuron

$$\mathbf{s}_t = [s^{(1)}, s^{(2)}, \ldots, s^{(k)}, \ldots, s^{(H)}] \tag{2.23}$$

and the context vector

$$\mathbf{c}_t = [c^{(1)}, c^{(2)}, \ldots, c^{(k)}, \ldots, c^{(H)}] = \mathbf{s}_{t-1} \tag{2.24}$$

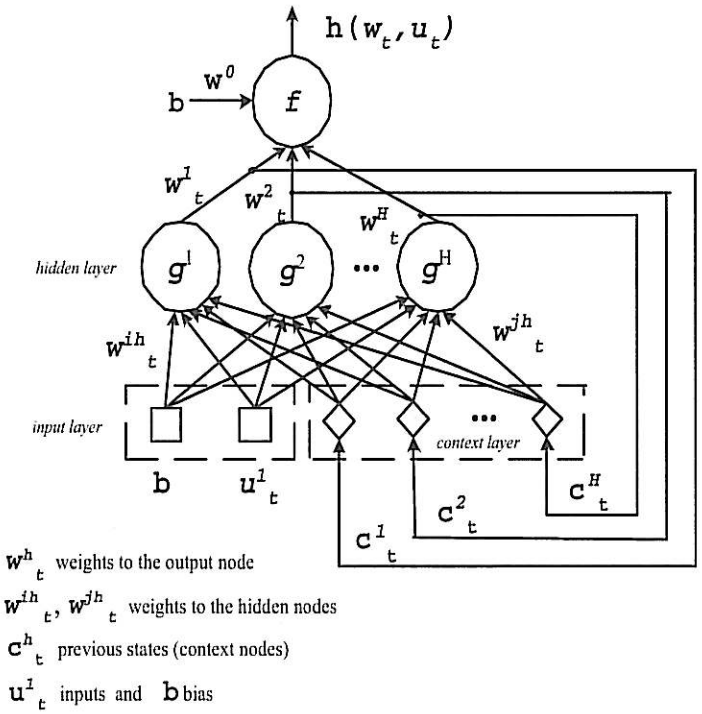

Figure 2.6: Elman Recurrent Artificial Neural Network: circles represent neurons, square represents the input and bias, and the diamonds represent the context nodes

is a vector of previous states, where H is the number of hidden neurons.

Figure 2.6 presents a schematic diagram of such a network using a bias term $b^{(1)}$ and a single network input $\mathbf{u}_t$. For notational convenience in the next chapter, the function $z_t(i)$ is introduced to represent the bias, input, or context later in a single variable.

$$z_t(i) = \begin{cases} b, & i = 0 \\ \mathbf{u}_t^{(j)}, & 1 \leq i \leq L \\ \mathbf{c}_t^{(i-L)}, & L+1 \leq i \leq L+H \end{cases} \tag{2.25}$$

The input vector to the network is defined as before in Equation 2.7. Each hidden node

computes one component $s_t^{(k)}$ of the state vector via the following equation:

$$\begin{aligned} \acute{s}_t^{(k)} &= \sum_{j=0}^{L+H} z_t(j) w^{(k,j)} \\ &= b^{(1)} w^{(k,0)} + \sum_{j=1}^{L} w^{(k,j)} \mathbf{u}_t^{(j)} + \sum_{i=L+1}^{L+H} w^{(k,i)} \mathbf{c}_t^{(i-L)} \end{aligned} \tag{2.26}$$

$$s_t^{(k)} = g^{(k)}(\mathbf{u}_t, \mathbf{s}_{t-1}) = \sigma(\acute{s}_t^{(k)}) \tag{2.27}$$

where $w^{(k,0)}$ is the bias weight, $w^{(k,j)}$ for $1 \leq j \leq L$ is the input weight, and $w^{(k,i)}$ for $1 \leq i \leq L+H$ are the weights on connections that connect the corresponding past i-th hidden node output (i.e., the feedback state signal $\mathbf{s}_{t-1}^{(i)}$ from the previous time step $t-1$) to the current state input. The activation functions $\sigma(\cdot)$ considered in this work are sigmoidal in form

$$\sigma(a) = 1/(1 + \exp(-a)) \tag{2.28}$$

which map the input a from $\mathbb{R}$ into a bounded interval $\Omega = (0, 1)$ of length $|\Omega| = 1$ where $\Omega \subset \mathbb{R}$.

The sum of the states multiplied by the corresponding weights are combined with the bias and are passed through the activation function to produce the output:

$$y_t = \sigma\left(b^{(0)} w^{(0)} + \sum_{j=1}^{H} w_o^{(j)} s_t^{(j)} \right) = \sigma(f(\mathbf{s}_t)) \tag{2.29}$$

where the output bias and corresponding weight is given by $b^{(0)}$ and $w^{(0)}$ respectively. The hidden to output node weights $w_o^{(j)}$ connect the states of the network to the output node y_t. In this thesis, the output node $y_t \in \mathbb{R}^c$ where $c = 1$. The total number of weights in the model is $m = H + 1 + (H + I + 1) \times H$. The overall network model thus becomes a highly nonlinear function $h(\mathbf{w}_t, \mathbf{u}_t)$ of the weights $\mathbf{w}_t$ and inputs $\mathbf{u}_t$.

$$\begin{aligned} d_t &= h(\mathbf{w}_t, \mathbf{u}_t) + \epsilon_t \\ &= y_t + \epsilon_t \end{aligned} \tag{2.30}$$

where the noise ϵ_t is assumed to be independent zero-mean Gaussian with variance σ_y^2 which is unknown: $\epsilon_t \sim \mathcal{N}(0, \sigma_y^2)$.

2.5.2 Williams and Zipser Recurrent Neural Network

The most general recurrent network architecture is the *fully connected recurrent network* as shown in Figure 2.7. We adopt the following notation to describe the fully recurrent network: the state vector $\mathbf{s}_t$ contains the state of each hidden neuron

$$\mathbf{s}_t = [s^{(1)}, s^{(2)}, \ldots, s^{(k)}, \ldots, s^{(H)}] \tag{2.31}$$

and the context vector

$$\mathbf{c}_t = [c^{(1)}, c^{(2)}, \ldots, c^{(k)}, \ldots, c^{(H)}] = \mathbf{s}_{t-1} \tag{2.32}$$

again contains the activations at the previous time step.

The inputs to each neuron consist of the context vector, the input vector for the network $\mathbf{u}_t$, and the bias b. The output activation of each neuron is defined as

$$s^{(k)} = g^{(k)}(\mathbf{u}_t, \mathbf{s}_{t-1}) \tag{2.33}$$

where $1 \leq k \leq H$. If the first neuron (i.e., the output neuron) is indexed, the output of the network is given by

$$y_t = g^{(1)}(\mathbf{u}_t, \mathbf{s}_{t-1}) \tag{2.34}$$

As in the Elman network, the function $z_t(j)$ is as defined in Equation 2.25, and each neuron $g^{(k)}$ computes its activation by first summing the weights times the corresponding input

$$\begin{aligned} \acute{s}_t^{(k)} &= \sum_{j=0}^{L+H} z_t(j) w^{(k,j)} \\ &= b w^{(k,0)} + \sum_{j=1}^{L} w^{(k,j)} \mathbf{u}_t^{(j)} + \sum_{i=L+1}^{L+H} w^{(k,i)} \mathbf{c}_t^{(i-L)} \end{aligned} \tag{2.35}$$

and then passing the sum through the activation function

$$s_t^{(k)} = \sigma(\acute{s}_t^{(k)}) \tag{2.36}$$

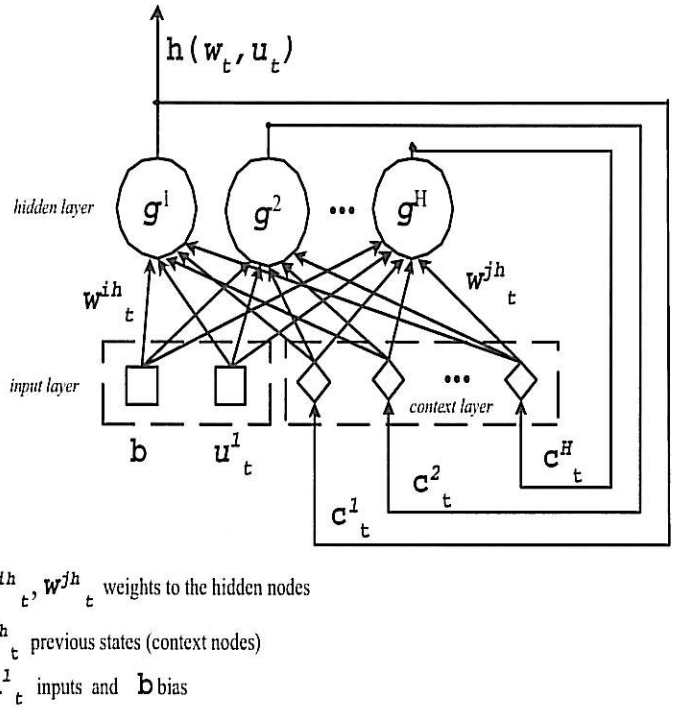

Figure 2.7: A schematic representation of the fully recurrent neural network architecture proposed by Williams and Zipser [228].

where $w^{(k,0)}$ is the bias weight, $w^{(k,j)}$ for $1 \leq j \leq L$ are the input weights, and $w^{(k,i)}$ for $1 \leq i \leq L + H$ are the weights on connections that connect the corresponding past i-th hidden node output (i.e., the feedback state signal $s_{t-1}^{(i)}$ from the previous time step $t - 1$) to the current state input. The overall network weight vector is defined as

$$\mathbf{w} = [w^{(1,0)}, w^{(1,1)}, \ldots, w^{(1,L+H)}, w^{(2,0)}, w^{(2,1)}, \ldots, w^{(2,L+H)}, \ldots, w^{(H,L+H)}] \tag{2.37}$$

The total number of weights for the fully connected network are $m = (H + I + 1) \times H$, and the output node $y_t \in \mathbb{R}^c$ where $c = 1$. The functions $\sigma(\cdot)$ are logistic sigmoidal nonlinearities defined in Equation (2.28). The entire network is referred to as highly nonlinear function $h(\mathbf{w}_t, \mathbf{u}_t)$ of the weights $\mathbf{w}_t$ and input $\mathbf{u}_t$

$$\begin{aligned} d_t &= h(\mathbf{w}_t, \mathbf{u}_t) + \epsilon_t \\ &= y_t + \epsilon_t \end{aligned} \tag{2.38}$$

where the noise ϵ_t is assumed to be independent zero-mean Gaussian with variance σ_y^2: $\epsilon_t \sim \mathcal{N}(0, \sigma_y^2)$.

Chapter 3

Optimization Algorithms for Time Series Modeling

Learning algorithms provide a means for iterative refinement of the model weights (parameters) to reach plausible well generalizing models. There are two main categories of learning/training algorithms: supervised and unsupervised learning algorithms [50]. In unsupervised learning the training algorithms enable the model to extract salient features from the data set without providing a teacher function (i.e., there is no desired response in the training set). On the other hand, supervised learning presents the model with an input and a corresponding output (desired response). Given this relationship, the objective of the learning algorithm is to adapt model parameters to minimize the error committed between the desired response and the output response of the network.

As this thesis focuses on supervised learning for time series modeling, we dichotomize supervised training algorithms into two main categories: fixed point and trajectory learning algorithms [169]. Fixed point learning algorithms train the recurrent neural structure to associate a static input with a static desired response, without specifying how the network arrives at the static response. In fixed point learning the training algorithm adapts model parameters to achieve dynamic stability such that upon presentation of an input, the network output/s eventually relax to the final desired response. Once this fixed point state is reached the system does not change. Trajectory learning is required when more than fixed point behavior is required, such as in time series modeling [169]. A trajectory is a sequence of desired responses, such as a time series. Unlike fixed point learning, trajectory learning adapts the network so that the network output follows the desired sequence

over time. Trajectory learning is a more general learning algorithm in the sense that it specifies how the network output should behave over the intermediate points as well as the final point [169].

3.1 Supervised Learning as Optimization

Training a time series model refers to searching for a set of model parameters $\mathbf{w}$ such that the model describes the data set, $\mathcal{D}$ "well." For one step ahead forecasting, the data set consists of the model inputs $\mathbf{u}_t$ and the corresponding target values d_t, $\mathcal{D} = \{\mathbf{u}_t, d_t\}_{t=1}^{T}$ where T is the number of input-output examples. The inputs are arranged as $\mathbf{u}_t = [d_{t-1}, d_{t-2}, \ldots, d_{t-L}]$. To find a model that describes the data set "well" it seems reasonable to demand that the model describes the observations in some optimal sense. How well the model describes the data can be measured by the error the model commits

$$\begin{aligned} e_t(\mathbf{w}) &= d_t - y_t \\ &= d_t - h(\mathbf{w}, \mathbf{u}_t) \end{aligned} \tag{3.1}$$

where at time t, the target is d_t and the model output is y_t, which is dependent on the input vector $\mathbf{u}_t$ and the model parameters/weights $\mathbf{w}$. For recurrent networks time is implicit in the model so the temporal ordering of the data is important. However, for feed forward networks, the order of the sequence of input/output patterns is not important.

The instantaneous error at particular moment t is:

$$\begin{aligned} E_t(\mathbf{w}) &= \Big(d_t - h(\mathbf{w}, \mathbf{u}_t)\Big)^2 \\ &= e_t^2(\mathbf{w}) \end{aligned} \tag{3.2}$$

At each time step, the standard method in the neural network and signal processing literature for computing the error of the model is defined by Equation 3.2 [78, 114] which relies on the definition of the error in Equation 3.1. From this measure, the model parameters $\mathbf{w}$ can be chosen such that the errors e_t, $t = 1, \ldots, T$ are minimized. The most common criterion for measuring the total error (accuracy) on the data set is the quadratic cost function:

$$C_T(\mathbf{w}) = \frac{1}{2} \sum_{\tau=1}^{T} E_\tau(\mathbf{w}) \tag{3.3}$$

The constant factor $1/2$ is included to allow for simplification in future derivations. The cost function is dependent on the data as well, but to simplify notation it is omitted. The quadratic cost function may be interpreted as the Euclidean distance between the desired output vector and the model outputs. It may also be interpreted in a Maximum Likelihood framework where the likelihood of the parameters/weights and the data $L(\mathbf{w}, \mathcal{D})$ can be maximized via minimization of the cost function $C_T(\mathbf{w})$.

By minimizing the cost function, the training algorithm searches for the "right" set of parameters through an iterative adaptation of the weights. This process is known as "training" or "learning". Learning algorithms can generally be considered as optimization procedures or search strategies. There exist numerous types of learning algorithms such as random search, Evolutionary Algorithms [65], simulated annealing [101], and gradient based search [228]. Each of these optimization algorithms have their advantages and disadvantages for various problems[1]. However, the three mentioned algorithms (random search, evolutionary search, and simulated annealing) are generally for batch/offline learning. For sequential training of neural networks, gradient based optimization methods are commonly utilized as they are able to compute an instantaneous gradient of the cost function. As this thesis focuses on sequential methods, the learning algorithms presented herein generally rely on gradient information.

Following an explanation given in [161], training is considered "an optimization problem in which the object is to find a set of parameters $\hat{\mathbf{w}}$ that minimizes the cost function,"

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} C_T(\mathbf{w}) \tag{3.4}$$

and in the case of $C_T(\mathbf{w})$ being quadratic, minimization of the cost function becomes a least squares optimization problem. Taylor expansion of the error function $C_t(\mathbf{w})$ results in,

$$C_T(\mathbf{w}) \approx C_T(\hat{\mathbf{w}}) + \mathbf{g}_T(\hat{\mathbf{w}})\delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\mathbf{H}_T(\hat{\mathbf{w}})\delta\mathbf{w} \tag{3.5}$$

where $\delta\mathbf{w} = \mathbf{w} - \hat{\mathbf{w}}$ and the first derivatives $\nabla C_t(\mathbf{w})$ are written as

$$\begin{aligned} \nabla C_T(\mathbf{w}) &= \mathbf{g}(\hat{\mathbf{w}}) \\ &= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{w}} \end{aligned} \tag{3.6}$$

[1]The error landscape for nonlinear models is usually characterized by numerous local minima, various training algorithms have been proposed to search this landscape.

and the Hessian (second derivatives) are written as

$$\begin{aligned} \nabla^2 C_T(\mathbf{w}) &= \mathbf{H}_T(\hat{\mathbf{w}}) \\ &= \frac{\partial^2 C_T(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^{\mathrm{T}}} \end{aligned} \tag{3.7}$$

where $\hat{\mathbf{w}}$ is a local minima of $C_T(\mathbf{w})$. A point in error space is a local minima if the following conditions are satisfied [63]:

- $\hat{\mathbf{w}}$ is a stationary point, $\mathbf{g}_T(\hat{\mathbf{w}}) = 0$
- $\mathbf{H}_T(\hat{\mathbf{w}})$ is positive definite, $\mathbf{w}^{\mathrm{T}}\mathbf{H}_T(\hat{\mathbf{w}})\mathbf{w} > 0, \forall \mathbf{w}$

For nonlinear problems, it usually is the case that there is no analytical solution to $\hat{\mathbf{w}}$ of $C_T(\mathbf{w})$, so iterative methods are relied on. To begin the iterative search process, an initial weight vector $\mathbf{w}^{(0)}$ is selected, and the algorithm then proceeds by generating a sequence $\{\mathbf{w}^{(k)}\}$ of parameter estimates that tends to a local optimum $\hat{\mathbf{w}}$. The algorithm produces an iterative estimate $\{\mathbf{w}^{(k)}\}$ in which convergence toward $\hat{\mathbf{w}}$ can be characterized by

$$\lim_{k \to \infty} ||\mathbf{w}^{(k)} - \hat{\mathbf{w}}|| = 0 \tag{3.8}$$

where the symbol $|| \cdot ||$ represents some vector norm. In general, the weight update can be expressed by the following equation

$$\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} + \eta \Delta \mathbf{w}^{(k-1)} \tag{3.9}$$

where $\mathbf{w}^{(k)}$ is the kth iterate of the weight vector, the search direction is $\Delta \mathbf{w}^{(k-1)}$, and η specifies the learning rate.

It is usually the case for neural networks and other nonlinear models that the quadratic cost function contains many weight vectors $\hat{\mathbf{w}}$ which satisfy the conditions for a local optima. The weight vectors $\hat{\mathbf{w}}$ satisfying the condition

$$C_T(\hat{\mathbf{w}}) \leq C_T(\mathbf{w}), \quad \forall \ \mathbf{w} \tag{3.10}$$

are considered global optima (and obtaining these points are the ultimate goal of training RNNs). Unfortunately with gradient based search, there is no general method that guar-

antees reaching the global optimum of the error surface. It turns out that iterative methods are guaranteed to converge to a local optima.

3.2 Derivative Computation

Before a neural network can be used, the weights must be adjusted to minimize the overall error between the desired and actual values for all output nodes over all input patterns. This process is termed "learning" or "training." The most commonly used algorithms for temporal data processing are gradient based methods in which Backpropagation is used to obtain derivative information. This procedure consists of two stages, a forward pass in which a data point (for sequential training) or the entire training set (for batch training) is passed through the network to obtain an overall measure of network performance, and a backward pass to compute error gradients in weight space starting at the output layer and propagating the error backward to successively obtain gradients at each previous layer. The weights are then modified proportionally to the respective error gradients in which the steepest descent method is commonly used.

The first training algorithm for computing RNN weight sensitivities was proposed by Hopfield [8]. The Hopfield RNN had no self feedback connections. The network generally embodied fixed point behavior, although some extensions allowed for oscillatory dynamics as a desired effect. Further training algorithms were developed to train continuous time RNNs, including the training algorithm of Pineda [167]. Similar to Hopfield, the network algorithm of Pineda led to dynamics that ended in fixed point behavior, i.e., $\partial y/\partial t = 0$. The algorithm was not useful for achieving oscillatory dynamics. To allow for a more rich behavior, Doya and Yoshizawa developed a continuous time backpropagation training algorithm [42] for trajectory learning. However, this algorithm adapted hidden weights by considering only its corresponding units direct influence on the error signal.

3.2.1 Backpropagation Through Time

For trajectory learning, the backpropagation through time (BPTT) algorithm [223, 224] was developed to take into account influences from all weights in the network on the output signal over time. The algorithm was presented in the same book that popularized the backpropagation algorithm for feed-forward networks [128]. The algorithm received a

comprehensive treatment in a book chapter by Williams and Zipser [228] which discussed practical variations of BPTT. The authors described the process of BPTT in two main phases, the forward propagation phase, and the backward pass. In the forward phase, network runs forward over the data. Four main quantities are stored, the activations of each neuron in the network $\mathbf{s}_t$, the network output activation y_t, the network inputs $\mathbf{u}_t$, and the error e_t. In the backward pass, the stored information is used to unfold the RNN through time. This unfolding results in an equivalent multi-layer feed-forward network in which each "layer" of the unfolded multi-layer feed-forward network corresponds to an activation state (time step) of the recurrent network. In the unfolded network representation, the network dynamic equations

$$\begin{aligned} \acute{s}_t^{(k)} &= \sum_{j=0}^{L+H} z_t(j) w_{[t]}^{(k,j)} \\ &= b^{(1)} w_{[t]}^{(k,0)} + \sum_{j=1}^{L} w_{[t]}^{(k,j)} \mathbf{u}_t^{(j)} + \sum_{i=L+1}^{L+H} w_{[t]}^{(k,i)} \mathbf{c}_t^{(i-L)} \end{aligned} \tag{3.11}$$

are similar to the original network equations (Equations 2.35) except in the unfolded representation the weights are now labeled with a time index in square brackets as in $w_{[t]}^{(k,j)}$. The square brackets represent the unfolded copies of the weights in time. The weights in successive layers of the unfolded representation of the RNN are thus constant, i.e., the value for the (i,j)th weight is constant between the unfolded layers. This means that the original weight is copied throughout the unfolding $w^{(k,j)} = w_{[t]}^{(k,j)}$ for all $t = 1, \ldots, T$.

After the network is unfolded, the standard backpropagation algorithm is then applied to compute the sensitivities of each weight. The gradient of the cost function $\partial C_T(\mathbf{w})/\partial \mathbf{w}$ can be decomposed in time using using the copied weights as follows:

$$\frac{\partial C_T(\mathbf{w})}{\partial w^{(i,j)}} = \sum_{t=1}^{T} \frac{\partial C_T(\mathbf{w})}{\partial w_{[t]}^{(i,j)}} \frac{\partial w_{[t]}^{(i,j)}}{\partial w^{(i,j)}} = \sum_{t=1}^{T} \frac{\partial C_T(\mathbf{w})}{\partial w_{[t]}^{(i,j)}} \tag{3.12}$$

note: the partial derivative $\partial w_{[t]}^{(i,j)} / \partial w^{(i,j)} = 1$.

Now the unfolded network is like a static feed forward network, and the gradient can

then be expressed as follows

$$\frac{\partial C_T(\mathbf{w})}{\partial w_{[t]}^{(i,j)}} = \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w_{[t]}^{(i,j)}} \tag{3.13}$$

To simplify notation, the standard short hand relations are introduced [228]:

$$\epsilon_t^{(i)} = -\frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \qquad \delta_t^{(i)} = -\frac{\partial C_T(\mathbf{w})}{\partial \acute{\mathbf{s}}_t^{(i)}} \tag{3.14}$$

$$\frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} = \sigma'(\acute{\mathbf{s}}_t^{(i)}) \qquad \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w_{[t]}^{(i,j)}} = z_t(j) \tag{3.15}$$

$$\delta_t^{(i)} = \epsilon_t^{(i)} \cdot \sigma'(\acute{\mathbf{s}}_t^{(i)}) \tag{3.16}$$

Substituting in these relations into the partial derivative $\partial C_T(\mathbf{w})/\partial w_{[t]}^{(i,j)}$ results in

$$\frac{\partial C_T(\mathbf{w})}{\partial w_{[t]}^{(i,j)}} = -\delta_t^{(i)} z_t(j) \tag{3.17}$$

As the weights $w_{[t]}^{(i,j)}$ influence the output directly as well as indirectly (i.e. in the future via the recurrent connections), the expression $\partial C_T(\mathbf{w})/\partial \mathbf{s}_t^{(i)}$ can be thought of as sensitive to the neuronal activations at the present time t as well as any future activations. So $\epsilon_t^{(i)}$ is dependent on $\mathbf{s}_t^{(i)}$, $\bar{\mathbf{s}}_t^{(i)}$, and $\acute{\mathbf{s}}_{t+1}^{(i)}$ [228] where $\bar{\mathbf{s}}_t^{(i)} = \mathbf{s}_t^{(i)}$ for all $t = 0, \ldots, T$.

The direct influence of the neuronal activations $\mathbf{s}_t^{(i)}$ on the cost function is given by the following expression

$$\frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}}|_{direct} = -\frac{\partial C_T(\mathbf{w})}{\partial \bar{\mathbf{s}}_t^{(i)}} \frac{\partial \bar{\mathbf{s}}_t^{(i)}}{\partial \mathbf{s}_t^{(i)}} \tag{3.18}$$

The future activations $\mathbf{s}_{t+1}^{(i)}$ can be thought of as indirectly impacting the cost function, so the expression describing this interaction involves the sensitivities of the cost function $C_T(\mathbf{w})$ with respect to the elements of future neuronal activities $\mathbf{s}_{t+1}^{(i)}$.

$$\frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}}|_{indirect} = -\sum_{l=1}^{T} \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(l)}} \frac{\partial \mathbf{s}_{t+1}^{(l)}}{\partial \mathbf{s}_t^{(i)}} = -\sum_{l=1}^{T} \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(l)}} \frac{\partial \mathbf{s}_{t+1}^{(l)}}{\partial \acute{\mathbf{s}}_{t+1}^{(l)}} \frac{\partial \acute{\mathbf{s}}_{t+1}^{(l)}}{\partial \mathbf{s}_t^{(i)}} \tag{3.19}$$

The change in the cost function with respect to some neuron activation can now expressed in terms of the direct and indirect components.

$$\begin{aligned}
\epsilon_t^{(i)} &= -\frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \\
&= \frac{\partial C_T(\mathbf{w})}{\partial \bar{\mathbf{s}}_t^{(i)}} \frac{\partial \bar{\mathbf{s}}_t^{(i)}}{\partial \mathbf{s}_t^{(i)}} + \sum_{l=1}^{T} \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(i)}} \frac{\partial \mathbf{s}_{t+1}^{(i)}}{\partial \acute{\mathbf{s}}_{t+1}^{(i)}} \frac{\partial \acute{\mathbf{s}}_{t+1}^{(i)}}{\partial \mathbf{s}_t^{(i)}} \\
&= \frac{\partial C_T(\mathbf{w})}{\partial \bar{\mathbf{s}}_{t+1}^{(i)}} + \sum_{l=1}^{T} (-\epsilon_{t+1}^{(i)}) \cdot \sigma'(\acute{\mathbf{s}}_{t+1}^{(i)}) \cdot w_{[t+1]}^{(l,i)} \\
&= e_t^{(i)} + \sum_{l=1}^{T} w_{[t+1]}^{(i,l)} \delta_{t+1}^{(l)}
\end{aligned} \tag{3.20}$$

The above equation is simplified via substitution with previously defined quantities and the following $\partial \acute{\mathbf{s}}_{t+1}^{(i)} / \partial \mathbf{s}_t^{(i)} = w_{[t+1]}^{(i,j)}$.

Neuron outputs generated for any time past $t > T$ have no impact on $C_T(\mathbf{w})$, and so $\epsilon_t^{(i)}$ (which is the partial derivative) is zero at $t = T + 1$, and at $t = T$, $\epsilon_t^{(i)} = e_t^{(i)}$. The weights from the unraveled network $w_{[t]}^{(i,j)}$ are replaced by the original weights $w^{(i,j)}$. This leads to the following:

$$\epsilon_t^{(i)} = \begin{cases} e_t^{(i)} & t = T \\ e_t^{(i)} + \sum_{l=1}^{T} w_{[t+1]}^{(l,i)} \delta_{t+1}^{(l)} & t < T \end{cases} \tag{3.21}$$

Finally, the weight update at time t is defined as

$$\Delta w^{(i,j)} = -\eta \frac{\partial C_T(\mathbf{w})}{\partial w^{(i,j)}} = \eta \sum_{t=1}^{T} \delta_t^{(i)} z_t(j) \tag{3.22}$$

where η is the learning rate. The initial conditions for the algorithm are $z_0(j) = 0$ and the context layer $\mathbf{c}_t^{(i)} = 0$ for $i = 1, \ldots, H$.

3.2.2 Real Time Recurrent Learning

The Real Time Recurrent Learning (RTRL) algorithm [228] uses an instantaneous error measure $C_{t_0,t_0}(\mathbf{w})$ rather than using the entire error measure as in BPTT [228]. The ben-

efits of this approach is its online operation which is useful if the data set is not complete before training begins. The RTRL algorithm computes the gradient by forward propagation of the error gradient in time, while the BPTT algorithm unfolds the network in time and computes the gradient over the unraveled nodes.

In the RTRL algorithm, the partial derivative $\partial C_T(\mathbf{w})/\partial w^{(k,l)}$ is given by

$$\begin{aligned}\frac{\partial C_T(\mathbf{w})}{\partial w^{(k,l)}} &= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \frac{\partial \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} \\ &= \sum_{t=1}^{T} \frac{\partial}{\partial \mathbf{s}_t^{(i)}} E_t(\mathbf{w}) \frac{\partial \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} \\ &= -\sum_{t=1}^{T} e_t(\mathbf{w}) \frac{\partial \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}}\end{aligned} \tag{3.23}$$

now $\partial \mathbf{s}_t^{(i)}/\partial w^{(k,l)}$ can be further expressed as

$$\frac{\partial \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} = \frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} = \sigma'(\acute{\mathbf{s}}_t^{(i)}) \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} \tag{3.24}$$

where change in the summation block $\acute{\mathbf{s}}_t^{(i)}$ with respect to $w^{(k,l)}$ is

$$\begin{aligned}\frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} &= \frac{\partial \left(\sum_{j=1}^{L+H} w^{(i,j)} z_t(j) \right)}{\partial w^{(k,l)}} \\ &= \sum_{j=1}^{L+H} \left[w^{(i,j)} \frac{\partial z_t(j)}{\partial w^{(k,l)}} + z_t(j) \frac{\partial w^{(i,j)}}{\partial w^{(k,l)}} \right] \\ &= \sum_{j=1}^{L+H} \left[w^{(i,j)} \frac{\partial z_t(j)}{\partial w^{(k,l)}} \right] + \delta_{i,k} z_t(l) \\ &= \sum_{j=1}^{H} \left[w^{(i,j)} \frac{\partial z_t(j)}{\partial w^{(k,l)}} \right] + \sum_{j=H+1}^{L+H} \left[w^{(i,j)} \frac{\partial z_t(j)}{\partial w^{(k,l)}} \right] + \delta_{i,k} z_t(l) \\ &= \sum_{j=1}^{L} \left[w^{(i,j)} \frac{\partial \mathbf{u}_t^{(j)}}{\partial w^{(k,l)}} \right] + \sum_{j=L+1}^{L+H} \left[w^{(i,j)} \frac{\partial \mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}} \right] + \delta_{i,k} z_t(l) \\ &= \sum_{j=L+1}^{L+H} \left[w^{(i,j)} \frac{\partial \mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}} \right] + \delta_{i,k} z_t(l)\end{aligned} \tag{3.25}$$

By substituting back into Equation 3.24,

$$\frac{\partial \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} = \sigma'(\acute{\mathbf{s}}_t^{(i)}) \Big[\sum_{j=L+1}^{L+H} w^{(i,j)} \frac{\partial \mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}} \Big] + \delta_{i,k} z_t(l) \tag{3.26}$$

where $\delta_{i,k}$ is the Kronecker delta. The initial conditions for the H dimensional system (Equation 3.26) are

$$\frac{\partial \mathbf{s}_0}{\partial w^{(k,l)}} = 0 \tag{3.27}$$

From this starting point, the partial derivatives can be propagated forward in time via recursively computing Equation 3.26 and stepping the network forward in time via Equations 2.35 and 2.36. Then, the gradient of the cost function can be computed via

$$\frac{\partial C_T(\mathbf{w})}{\partial w^{(i,j)}} = \sum_{t=1}^{T} e_t(\mathbf{w}) \frac{\partial \mathbf{s}_t^{(1)}}{\partial w^{(k,l)}} \tag{3.28}$$

There have been two main ways in which the RTRL algorithm has been used, batch and sequential. In the batch update formulation, the algorithm accumulates the gradient of the error function by Equation 3.28 and then performs the weight update.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \frac{\partial C_T(\mathbf{w})}{\partial w^{(k,l)}} \tag{3.29}$$

where $\eta > 0$ is a small constant known as the learning rate, and $\mathbf{w}^{(\tau)}$ is the weight vector at epoch (batch update) τ.

The alternative is to update the weights after each training pattern. This method has been popular in the literature as it allows for online operation of the learning algorithm. In this formulation, the error is computed at each time step rather than accumulated.

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta e_t(\mathbf{w}_{t-1}) \frac{\partial \mathbf{s}_t}{\partial \mathbf{w}_t} \tag{3.30}$$

This online updating of the weights is known as the RTRL algorithm [228].

To facilitate simpler notation in the following chapters, the partial derivatives of the

output are then collected into a Jacobian vector

$$\mathbf{j}_t = [\frac{\partial s_t^{(1)}}{\partial w^{(1,0)}}, \ldots, \frac{\partial s_t^{(1)}}{\partial w^{(1,L+H)}}, \frac{\partial s_t^{(1)}}{\partial w^{(2,0)}} \cdots, \frac{\partial s_t^{(1)}}{\partial w^{(2,L+H)}}, \ldots, \frac{\partial s_t^{(H)}}{\partial w^{(H,L+H)}}] \tag{3.31}$$

3.2.3 Time Complexity of Derivative Computation

Table 3.1: Computational complexity for derivative computation of RNNs. The complexity is expressed in terms of the size of the hidden layer of the network [105].

Algorithm	Approximate Time Complexity
RTRL	$\mathcal{O}(H^4)$
BPTT	$\mathcal{O}(H^2)$
BPTT-H	$\mathcal{O}(H^2)$
BPTT-WilPeng	$\mathcal{O}(H^2)$
BPTT-RTRL Hybrid	$\mathcal{O}(H^3)$

The time complexity of the algorithms discussed can be found in the first and second rows of Table 3.2.3. The variable H represents the number of nodes in the hidden layer. The BPTT algorithm is considerably faster than the RTRL algorithm, however BPTT is a batch procedure and RTRL is for online training. Various methods for reducing the computational burden have been proposed such as BPTT-H [228] and BPTT-WilPeng [227]. A hybrid BPTT-RTRL algorithm for online training was proposed [188] which reduced the RTRL computational complexity by one order of magnitude for online training.

3.3 Batch vs Sequential Training

There are two main approaches to training neural based models: batch training, and sequential training. It will be helpful to define the data sets; in cases where data is available before training starts we define

$$\begin{aligned} \mathcal{D}^{train} &= \{y_t, d_t\}_{t=1}^{T} \\ \mathcal{D}^{test} &= \{y_t, d_t\}_{t=T+1}^{\Upsilon} \end{aligned} \tag{3.32}$$

where T is the number "in sample" data and Υ is the number of "out of sample" data. In cases where not all data is available before training starts

$$\mathcal{D}^{online} = \{y_t, d_t\} \quad t = 1, 2, \ldots \tag{3.33}$$

the data set is not complete.

A) Batch training: Batch algorithms evaluate all data $\mathcal{D}^{train}$ before updating the weights $\mathbf{w}$ of the RNN. These algorithms have the following general structure

$$\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} + \Delta\mathbf{w}^{(k-1)}(\mathcal{D}^{train}, \mathbf{w}^{(k-1)}) \tag{3.34}$$

where k is the batch index, and $\Delta\mathbf{w}^{(k-1)}(\cdot)$ is the search direction which is a function of the entire data set and the weight estimate of the last batch update. To evaluate the performance of the network, the weights are frozen (no training takes place), and the network predicts over $\mathcal{D}^{test}$. BPTT is typically used to compute the derivatives for this type of update.

B) Sequential training: Sequential algorithms have the general structure

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \Delta\mathbf{w}_{t-1}(\mathcal{D}^{online}, \mathbf{w}_{t-1}) \tag{3.35}$$

where t is the current time instant, and the search direction is a function of the updated weight vector at the last time step, the network input vector, and the target all at the current time step. The main idea is that one update of the weights is performed per time step. RTRL is typically used to compute the derivatives for this type of update.

There are two different ways in which sequential training can be performed, off-line and on-line.

1. Sequential off-line training: the algorithm iterates (possibly multiple times) over the data set $\mathcal{D}^{train}$ and updates the weights as in Equation (3.35) i.e., at each time step. During evaluation of model parameters over $\mathcal{D}^{test}$ the weights can still be updated sequentially as the algorithm can update the weigh vector $\mathbf{w}_t$ as each new measurement arrives (at each time step).

2. Sequential on-line training: the data set does not have to be complete before training starts, i.e., the model is trained and evaluated over $\mathcal{D}^{online}$. The weight updates take place as in Equation (3.35).

3.4 Batch Training Algorithms

First-order gradient descent (known as gradient descent) is the most common approach to minimization of the cost function. Gradient descent uses the error gradient of the cost function to guide the search towards a minimum of the cost function. Gradient descent has been used due to its simplicity, however, it is well known that the procedure is highly ineffective for training RNNs due to long training times, susceptibility to being trapped in local minima, resulting in poor generalizing solutions [145, 172, 213].

3.4.1 First-order Batch Procedures

The goal of optimization is to minimize the cost function $C_T(\mathbf{w})$ through adjustment of the weights/parameters $\mathbf{w}$. For nonlinear models such as neural networks, this process generally takes multiple iterations or epochs. The search direction $\Delta\mathbf{w}^{(k-1)}$ is generally chosen such that $C_T(\mathbf{w}^{(k)}) < C_T(\mathbf{w}^{(k-1)})$. This corresponds to choosing a search direction that points "downhill" where the derivative of the cost function $C_T(\mathbf{w})$ at $\mathbf{w}^{(k)}$ is negative, i.e.

$$\mathbf{g}_T(\mathbf{w}^{(k-1)})^{\mathrm{T}}\Delta\mathbf{w}^{(k-1)} \leq 0 \tag{3.36}$$

Adjustment of the weights in the direction opposite of the gradient turns out to be the direction that decreases the cost function $C_T(\mathbf{w})$ the most rapidly. This direction is commonly known as the direction of steepest-descent. With this choice, the weights of the neural network can be updated at the end of each epoch as

$$\begin{aligned}\mathbf{w}^{(k)} &= \mathbf{w}^{(k-1)} - \Delta\mathbf{w}^{(k-1)} \\ &= \mathbf{w}^{(k-1)} - \eta\mathbf{g}_T(\mathbf{w}^{(k-1)})\end{aligned} \tag{3.37}$$

This method of iterative refinement of the neural network weights has become known as gradient descent learning (sometimes assumed to be used with backpropagation). This method is a first-order optimization technique as only the first-order derivatives of the cost function are used in the optimization process. The gradient of the cost function is computed as follows

$$\mathbf{g}_T(\mathbf{w}) = \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{\tau=1}^{T} e_\tau(\mathbf{w})\frac{\partial \mathbf{s}_\tau}{\partial \mathbf{w}} \tag{3.38}$$

where $\partial \mathbf{s}_\tau / \partial \mathbf{w}$ is the output of the neural network with respect to some weight $\mathbf{w}$. The convergence of this algorithm relies on a parameter η known as the step size or learning rate. Generally η is chosen to be a small positive constant such that each step leads to a decrease in the cost function. This however can be problematic as a small step size may lead to very slow convergence toward a local minimum. On the other hand, the choice of η too large may lead to overstepping the minima, and divergence of the algorithm.

3.4.2 Second-Order Batch Methods

A significant amount of effort has been focused towards improving upon first-order training algorithms such as gradient descent. A plethora of research in the area has evaluated most all well known optimization methods for training neural networks [144]. It was found that the most successful (in terms of speed and generalization capacity) optimization algorithms tended to include second-order information of the cost function. Second-order methods generally use more information about the error surface than first order methods. The main difference between first-order methods and second-order methods is that first order methods direct the search in the direction of maximum change, where as second order methods descend in the direction of the local minimum.

To develop training algorithms based on the information contained in the second derivatives of the cost function, a second-order Taylor series expansion of the cost function around the current weight estimate $\mathbf{w}^k$ is written

$$C_T(\mathbf{w}) = C_T(\mathbf{w}^{(k)}) + \Delta \mathbf{w}^{\mathrm{T}} \mathbf{g}_T(\mathbf{w}^{(k)}) + \frac{1}{2} \Delta \mathbf{w}^{\mathrm{T}} \mathbf{H}_T(\mathbf{w}^{(k)}) \Delta \mathbf{w} \tag{3.39}$$

where $\mathbf{w} = \mathbf{w}^{(k)} + \Delta \mathbf{w}$. The second partial derivatives of $C_T(\mathbf{w})$ are known as the Hessian $\mathbf{H}_T(\mathbf{w})$ which is computed as

$$\mathbf{H}_T(\mathbf{w}) = \frac{\partial^2 C_T(\mathbf{w})}{\partial \mathbf{w}^{\mathrm{T}} \partial \mathbf{w}} = -\sum_{t=1}^{T} \left[e_t(\mathbf{w}) \frac{\partial^2 \mathbf{s}_t}{\partial \mathbf{w} \partial \mathbf{w}^{\mathrm{T}}} - \frac{\partial \mathbf{s}_t}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{s}_t}{\partial \mathbf{w}} \right] \tag{3.40}$$

Second-order optimization methods generally include the above expansion to obtain the search direction. Computation of the exact second-order terms are generally expensive to compute. An R-propagation based procedure for computation of the Hessian is provided in [153]. In the following sections, well known second-order RNN training algorithms are reviewed.

3.4.2.1 Newton's Method

One of the most well known second-order optimization methods is Newton's method. The algorithm chooses the weight estimate at the next time step $\mathbf{w}^{(k)}$ as the minimizer of the second-order expansion of the cost function $C_T(\mathbf{w})$. The search direction $\Delta\mathbf{w}^{(k-1)}$ involves computation of the of the gradient and the inverse of the Hessian matrix

$$\Delta\mathbf{w}^{(k-1)} = -[\nabla^2 C_T(\mathbf{w})]^{-1}\nabla C_T(\mathbf{w}) = -[\mathbf{H}_T(\mathbf{w}^{(k-1)})]^{-1}\mathbf{g}_T(\mathbf{w}^{(k-1)}) \tag{3.41}$$

where $\nabla C_T(\mathbf{w}) = \mathbf{g}_T(\mathbf{w}^{(k-1)}) = \partial C_T(\mathbf{w})/\partial\mathbf{w}$ is the vector of first-order derivatives, and $\nabla^2 C_T(\mathbf{w}) = \mathbf{H}_T(\mathbf{w}^{(k-1)}) = \partial^2 C_T(\mathbf{w})/\partial\mathbf{w}^2$ is the matrix of second-order derivatives called the Hessian.

Assuming that a starting point close to the local minimum $\hat{\mathbf{w}}$ is chosen, then the convergence of Newtons method is quadratic [40].

$$\| \mathbf{w}^{(k+1)} - \hat{\mathbf{w}} \| \leq c \| \mathbf{w}^{(k)} - \hat{\mathbf{w}} \|, \quad c \geq 0 \tag{3.42}$$

However, if the RNN is initialized far from the local minimum the Hessian may not be positive definite. In fact, the Hessian may be negative definite, which can result in the search moving toward a local maxima. Netwon's method is termed locally convergent [114].

Another limiting factor of Netwon's method is the enormous computational burden for calculating the second derivative terms of the inverse Hessian. With these two negative aspects of the algorithm (local convergence and heavy computational cost), Newton's method has not been popular for training RNNs.

3.4.2.2 The Gauss-Newton Method

A large computational burden can be avoided through an approximation of the true Hessian (Equation 3.40) with

$$\mathbf{H}_T(\mathbf{w}^{(k-1)}) = \frac{\partial^2 C_T(\mathbf{w})}{\partial\mathbf{w}^{\mathrm{T}}\partial\mathbf{w}} \approx -\sum_{t=1}^{T} \frac{\partial\mathbf{s}_t}{\partial\mathbf{w}} \cdot \frac{\partial\mathbf{s}_t}{\partial\mathbf{w}^{\mathrm{T}}} \tag{3.43}$$

where the terms involving the second derivatives of the model output are ignored. Assuming that the RNN is capable of approximating the underlying system for the set of

parameters $\hat{\mathbf{w}}$, then as the number of data points T approaches infinity, the approximation to the Hessian will become exact.

Substituting this approximation (Equation 3.43) for the exact Hessian in Equation (3.41) results in the Gauss-Newton search direction. This approach makes the implementation much more simple as only the information from the gradient computation is used.

The Hessian $\mathbf{H}_T(\mathbf{w}^{(k-1)})$ can be written as a function of two components

$$\mathbf{H}_T(\mathbf{w}^{(k-1)}) = \mathbf{J}_T^{\mathrm{T}}(\mathbf{w}^{(k-1)})\mathbf{J}_T(\mathbf{w}^{(k-1)}) + \mathbf{S}_T(\mathbf{w}^{(k-1)}) \tag{3.44}$$

where

$$\mathbf{J}_T(\mathbf{w}^{(k-1)}) = \frac{\partial \mathbf{s}_t}{\partial \mathbf{w}}, \qquad \mathbf{S}_T(\mathbf{w}^{(k-1)}) = \sum_{t=1}^{T} e_t \frac{\partial^2 \mathbf{s}_t}{\partial \mathbf{w} \partial \mathbf{w}^{\mathrm{T}}} \tag{3.45}$$

The batch Jacobian matrix is $\mathbf{J}_T(\mathbf{w}) = [\mathbf{j}_1(\mathbf{w}), \ldots, \mathbf{j}_T(\mathbf{w})]^{\mathrm{T}}$ and the second-order terms are contained in the matrix $\mathbf{S}_T(\mathbf{w})$. In the Gauss Newton approach, the hessian is written as

$$\mathbf{H}_T(\mathbf{w}^{(k-1)}) = \mathbf{J}_T^{\mathrm{T}}(\mathbf{w}^{(k-1)})\mathbf{J}_T(\mathbf{w}^{(k-1)}) \tag{3.46}$$

The weight update then becomes

$$\begin{aligned} \mathbf{w}^{(k)} &= \mathbf{w}^{(k-1)} - \mathbf{H}_T^{-1}(\mathbf{w}^{(k-1)})\mathbf{J}_T(\mathbf{w}^{(k-1)})\mathbf{e}_T(\mathbf{w}^{(k-1)}) \\ &= \mathbf{w}^{(k-1)} - [\mathbf{J}_T^{\mathrm{T}}(\mathbf{w}^{(k-1)})\mathbf{J}_T(\mathbf{w}^{(k-1)})]^{-1}\mathbf{J}_T(\mathbf{w}^{(k-1)})\mathbf{e}_T(\mathbf{w}^{(k-1)}) \end{aligned} \tag{3.47}$$

This simplification eliminates the computations that would have been needed to obtain the $\mathbf{S}_T(\mathbf{w})$ matrix, which makes the method feasible for RNN training. However, a critical assumption is that in Equation (3.46) the approximation to the Hessian is always positive definite.

3.4.2.3 The Levenberg-Marquardt Method

The optimal learning step using second-order information is again restated [15, 77]:

$$\Delta \mathbf{w}^{(k-1)} = -[\nabla^2 C_T(\mathbf{w})]^{-1} \nabla C_T(\mathbf{w}) = [\mathbf{H}_T(\mathbf{w}^{(k-1)})]^{-1} \mathbf{g}_T(\mathbf{w}^{(k-1)}) \tag{3.48}$$

Theoretically, such a learning algorithm should converge to the optimal weight vector, but convergence is not guaranteed since the Hessian is often ill-conditioned. A problem is

said to be ill-conditioned when small changes in the data can lead to large changes in the model output. Conditioning can be measured by the condition number $\mathrm{k}(\mathbf{H}_T(\mathbf{w}^{(k-1)})) = \lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are the largest and smallest eigenvalues of $\mathbf{H}_T(\mathbf{w}^{(k-1)})$. A large condition number indicates ill-conditioning of the Hessian matrix. It is often the case in second-order optimization methods that the condition number may become large. This is especially relevant to equation (3.48), as a large condition number of the Hessian makes the solution sensitive to the numerical precision of the computational hardware which leads to errors in the model parameters [66].

Such difficulties of Hessian-based training can be alleviated through regularization [208]; by adding a damping (regularization) parameter to the diagonal of the Hessian a lower bound on the smallest eigenvalues can be imposed ensuring numerical stability. This leads to the Levenberg-Marquardt algorithm [112, 127] which offers an efficient technique that combines regularization with second-order training. The Levenberg-Marquardt algorithm has become a popular optimization method for Neural Network training. The Levenberg-Marquardt method tends to be more robust to starting conditions far from the minima than the Gauss-Newton method. This robustness can be attributed to the Gradient Descent like behavior of the algorithm far from the minima, and the Newton like behavior close to the minimum.

It capitalizes on the squared error function (3.3) and uses an efficient approximation to the Hessian to circumvent the need for extensive computations of second derivatives. With respect to the total error, it was shown that $\nabla^2 C_T(\mathbf{w}) = \mathbf{J}_T^{\mathrm{T}}(\mathbf{w})\mathbf{J}_T(\mathbf{w}) + \mathbf{S}_T(\mathbf{w})$, where $\mathbf{S}_T(\mathbf{w})$ denotes the part of the Hessian matrix involving second-order information of the error surface, and $\mathbf{J}_T(\mathbf{w})$ is the Jacobian matrix of first-order derivatives with respect to each of the weights per training example $\mathbf{J}_T(\mathbf{w}) = [\mathbf{j}_1(\mathbf{w}), \ldots, \mathbf{j}_T(\mathbf{w})]^{\mathrm{T}}$.

Key to the Levenberg-Marquardt algorithm is the replacement of $\mathbf{S}_T(\mathbf{w})$ by a matrix $\alpha\mathbf{I}$ [112, 127]. This algorithm exploits the convenient property that $\mathbf{J}_T^{\mathrm{T}}(\mathbf{w})\mathbf{J}_T(\mathbf{w})$ is symmetric and non-negative definite. The performance of this algorithm, though relies on the choice of the α parameter, which is usually found through heuristics. This algorithm assumes that the Hessian can be inverted only if it is within a trust region of small radius which can be determined using the eigenvalues $\{\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(M)}\}$ and the eigenvectors $\{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(M)}\}$ of the Hessian matrix as follows:

$$\nabla^2 C_T(\mathbf{w})\mathbf{v}^{(i)} = [\mathbf{H}_T(\mathbf{w}) + \alpha\mathbf{I}]\mathbf{v}^{(i)} = \mathbf{H}_T(\mathbf{w})\mathbf{v}^{(i)} + \alpha\mathbf{v}^{(i)} = (\lambda^{(i)} + \alpha)\mathbf{v}^{(i)} \qquad (3.49)$$

where α is the predefined regularization constant (damping parameter). The penalized matrix $\mathbf{H}_T(\mathbf{w}) + \alpha\mathbf{I}$ can be made potentially invertible (positive definite) by choosing the parameter α such that $(\lambda^{(i)} + \alpha) > 0$ for all i.

There are two problems in the above formulation 1) the batch Levenberg-Marquardt algorithm is unable to incorporate new measurements without re-processing the entire batch; and 2) using one global common regularizer does not help to accurately tune the weights so as to avoid over-fitting [68, 212]. A remedy for the above mentioned problems is to use recursive inversion of the Hessian with an individualized local regularization parameter for each weight, identified through Bayesian inference, all of which is explained in Chapter 4 [137].

3.5 Sequential Training Methods

The previously mentioned training algorithms are all considered "batch" methods for off-fline situations in which the entire data set is available before the training process begins. The model is trained on the data set, and then run over an unseen data set to produce forecasts. No learning takes place during the "forecasting" phase as the model is fully trained before forecasting takes place. This restriction can be disadvantageous as not all information is integrated into the model when producing a forecast at the next time step. In the off-line (batch) mode of training, the weights of the neural model are not updated until the entire data set is processed. Each of these iterations is known as an epoch in neural network jargon.

Adaptation of the weights in a sequential setting happens more frequently than in the batch mode. At each data point, the weights of the model are updated. This allows for learning on sequentially arriving data, i.e. training can start before the entire data set arrives. The model continuously improves itself through recursively updating model weights through time [114]. For learning in a sequential setting, the model parameters are usually updated immediately after a new measurement arrives, and thus a common error function chosen to reflect the instantaneous error of the model is given by,

$$E_t(\mathbf{w}) = \frac{1}{2}e_t^2(\mathbf{w}) \tag{3.50}$$

This type of error measure is useful for gradient based training methods, but for more complex training algorithms based on second-order information of the error function an

accumulated error measure is usually chosen.

The accumulated error function is commonly used for sequential training [114]. Based on this error function, the weight updates are based on the accumulated error given by:

$$C_t(\mathbf{w}) = \frac{1}{t}\sum_{\tau=1}^{t} E_\tau(\mathbf{w}) \tag{3.51}$$

This accumulated error function allows for sequential versions of Gauss-Newton type second order algorithms [114].

3.5.1 First-Order Sequential Methods

3.5.1.1 Sequential Gradient Descent

The most simplistic sequential learning algorithm adds recursion to the batch gradient descent learning rule. In the batch case the gradient of the cost function $\partial C_t(\mathbf{w})/\partial \mathbf{w}$ is computed by adding up the gradients obtained from each training example. In the stochastic gradient approach, each training example is used to estimate the gradient of the cost function $\mathbf{j}_t(\mathbf{w})e_t(\mathbf{w})$ at each time step t, where $\mathbf{j}_t$ is defined in Equation 3.31.

The weights are then updated by the following equation

$$\begin{aligned} \mathbf{w}_t &= \mathbf{w}_{t-1} + \eta \mathbf{j}_t(\mathbf{w}_{t-1})e_t(\mathbf{w}_{t-1}) \\ &= \mathbf{w}_{t-1} + \eta \mathbf{g}_t(\mathbf{w}_{t-1}) \end{aligned} \tag{3.52}$$

where the weight estimate is $\mathbf{w}_t$, and the step-size parameter at time step t is η. The algorithm is called "stochastic gradient" decent [115] because the instantaneous gradients may be viewed as a random sample of $\partial C_t(\mathbf{w})/\partial \mathbf{w}$. Other authors call the method "stochastic approximation" [176]. However, in much of the neural network literature, the stochastic gradient algorithm is usually called back-propagation with sequential gradient descent [128]. There are two advantages to approximating the gradient by its instantaneous value, on-line learning of time-varying systems becomes possible; and in comparison to the (off-line) gradient descent algorithm the on line algorithm may escape from local minima. As the on line gradient descent is stochastic, although the algorithm takes a step to decrease the cost function at each step, the stochastic nature of the gradient

may actually result in an increase of the cost function which allows for the possibility of stepping out of local minima.

3.5.2 Second-Order Sequential Methods

3.5.2.1 Recursive Gauss Newton

The recursive Gauss Newton training algorithm is a sequential second-order training algorithm that has been used for sequential training of RNN time series models [216]. The recursive Gauss Newton method differs from the sequential gradient descent method discussed in Section 3.5.1.1 in that two quantities are sequentially updated, the weight vector and the Hessian matrix (rather than just the weight vector in the sequential gradient descent method). The recursive Gauss Newton method can be though of as a stochastic form of the Gauss-Newton method (from Section 3.4.2.2) applied at each data point. The derivation summarized in the remainder of this section generally follows [114], and a similar presentation in the context of RNNs can be found in [216].

The estimated weight vector $\mathbf{w}_t$ at time t, is an approximation of $\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} C_t(\mathbf{w})$. According to the normal form for recursive estimation (3.35) is written as:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{g}_t(\mathbf{w}_{t-1}) \tag{3.53}$$

where $\mathbf{H}_t(\mathbf{w}_{t-1})$ is the Hessian defined later in Equation 3.57, and the gradient is

$$\begin{aligned} \mathbf{g}_t(\mathbf{w}) &= \frac{1}{2}\frac{\partial C_t(\mathbf{w})}{\partial \mathbf{w}} \\ &= \sum_{\tau=1}^{t} \mathbf{j}_\tau(\mathbf{w}_{t-1}) e_\tau(\mathbf{w}_{t-1}) \end{aligned} \tag{3.54}$$

where $\mathbf{j}_t$ is defined in Equation 3.31, and

$$\begin{aligned} \mathbf{g}_t(\mathbf{w}_{t-1}) &= \sum_{\tau=1}^{t} \mathbf{j}_\tau(\mathbf{w}_{t-1}) e_\tau(\mathbf{w}_{t-1}) \\ &= \sum_{\tau=1}^{t-1} \mathbf{j}_\tau(\mathbf{w}_{t-1}) e_\tau(\mathbf{w}_{t-1}) - \mathbf{g}_t(\mathbf{w}_{t-1}) e_t(\mathbf{w}_{t-1}) \\ &= \mathbf{g}_{t-1}(\mathbf{w}_{t-1}) - \mathbf{j}_t(\mathbf{w}_{t-1}) e_t(\mathbf{w}_{t-1}) \end{aligned} \tag{3.55}$$

Assuming that $\mathbf{w}_{t-1}$ minimizes the cost function $C_{t-1}(\mathbf{w})$ then $\mathbf{g}_{t-1}(\mathbf{w}) = 0$, i.e.,

$$\mathbf{g}_t(\mathbf{w}_{t-1}) = \mathbf{j}_t(\mathbf{w}_{t-1})e_t(\mathbf{w}_{t-1}) \tag{3.56}$$

The pseudo Hessian is given by

$$\begin{aligned} \mathbf{H}_t(\mathbf{w}) =& \frac{1}{2}\frac{\partial^2 C_t(\mathbf{w})}{\partial \mathbf{w}}\mathbf{w}^{\mathrm{T}} \\ =& \sum_{\tau=1}^{t} \mathbf{j}_\tau(\mathbf{w})\mathbf{j}_\tau(\mathbf{w})^{\mathrm{T}} \end{aligned} \tag{3.57}$$

where

$$\begin{aligned} \mathbf{H}_t(\mathbf{w}_{t-1}) &= \sum_{\tau=1}^{t} \mathbf{j}_\tau(\mathbf{w}_{t-1})\mathbf{j}_\tau(\mathbf{w}_{t-1})^{\mathrm{T}} \\ &= \sum_{\tau=1}^{t-1} \mathbf{j}_\tau(\mathbf{w}_{t-1})\mathbf{j}_\tau(\mathbf{w}_{t-1})^{\mathrm{T}} + \mathbf{j}_t(\mathbf{w}_{t-1})\mathbf{j}_t(\mathbf{w}_{t-1})^{\mathrm{T}} \\ &= \mathbf{H}_{t-1}(\mathbf{w}_{t-1}) + \mathbf{j}_t(\mathbf{w}_{t-1})\mathbf{j}_t(\mathbf{w}_{t-1})^{\mathrm{T}} \end{aligned} \tag{3.58}$$

The recursive algorithm involves

$$\begin{aligned} \mathbf{H}_t(\mathbf{w}_{t-1}) &= \mathbf{H}_{t-1}(\mathbf{w}_{t-1}) + \mathbf{j}_t(\mathbf{w}_{t-1})\mathbf{j}_t(\mathbf{w}_{t-1})^{\mathrm{T}} \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{g}_t(\mathbf{w}_{t-1}) \end{aligned} \tag{3.59}$$

The sequential Gauss-Newton method works well for solutions close to the minimum. However, when far from the minimum, the Hessian is likely to be ill-conditioned. This problematic limitation of the algorithm can be alleviated by the addition of regularization parameters to the Hessian to improve numerical stability. Such an approach is the focus of the next chapter.

Chapter 4

Recursive Recurrent Bayesian Levenberg-Marquardt Learning

There are many situations in which RNNs are required to function online such as in situations where data sequences are either nonstationary and/or when data arrives sequentially. In this chapter sequential learning and regularization schemes are developed for improved RNN training. In the sequential learning paradigm network training is reinterpreted in a dynamical systems framework in which state estimates $\mathbf{w}_t$ of the original systems states are computed using a set of measurements $\{\mathbf{u}_{1:t}\}$. The estimation problem is fundamentally dynamic as new measurements arrive at each time step. The newly arriving observations are incorporated into weight estimates at each time step through recursive estimation.

Although much effort has been spent on the development of model parameter estimation schemes for RNNs (as shown in Chapter 4), which make use of first and second-order approximations of the error surface [18, 24, 28, 104, 172, 216], direct optimization (without complexity control) has lead to mediocre results [58, 102, 113]. One reason for this is the ill-posed nature of the problem, i.e. parameter (weight) estimation involves inversion of a nonlinear dynamical system from finite and noisy data which typically is ill-posed [19, 79]. Moreover, research in neural modeling has found that for processing time series, first and second-order batch training algorithms tend to underperform compared to incremental second-order training algorithms [151].

Previous work have reported on increases in generalization capacity through the regularization of RNNs [62, 95, 111, 162, 201, 229]. However, the above mentioned papers

share two common weaknesses: first, they use a uniform regularization parameter for all weights in the model; and second, the computation of the regularization parameter relied on suboptimal methods (i.e. heuristics, cross-validation, or bootstrapping). Cross-validation and bootstrapping use subsets of the data for training and do not use the entire data set for model parameter estimation [122].

A principled approach to estimation of the regularization parameter(s) has been proposed in a Bayesian setting [122, 148]. The probabilistic framework facilitates inference of the regularization hyperparameters, which are viewed as beliefs in the uncertainties of the model parameters. In previous work, the Bayesian evidence procedure [122] has been followed to impose a uniform regularizer when training RNNs [206] in an off-line manner. This involves off-line estimation of the covariance matrix (Hessian), that is computationally inappropriate when handling large numbers of network parameters, or numerous training examples, or both. In such cases the eigenvalues of the Hessian matrix are known to decay to zero causing numerical instabilities (i.e. singularity of the Hessian).

A solution to both above mentioned problems is to use non-uniform regularization with hyperparameters that are adapted to maximize the weight posterior distribution [68, 137, 133]. Following this reasoning a probabilistic approach to recursive second-order training of recurrent networks via a Recursive Bayesian Levenberg-Marquardt (RBLM-RNN) algorithm is developed [137, 133]. The main contributions are: 1) incorporation of individual, non-uniform Bayesian regularization parameters for each weight to account for its uncertainties; 2) handling of the target noise through a specific noise hyperparameter; 3) derivation of a regularized equation for recursive second-order estimation of the weights; 4) formulation of an equation for the recursive computation of the inverted regularized dynamic Hessian matrix; and 5) estimation of Bayesian confidence intervals. Attention is directed to the recursive Levenberg-Marquardt algorithm because it was found to have superior convergence properties than other online algorithms for training neural networks [152], such as the recursive steepest descent, and the recursive Gauss-Newton method.

4.1 Ill-Posed Problems

Learning from temporal data involves a tradeoff between, fitting a model to data by minimizing some loss function, and performance on out-of-sample data. When fitting a model to data, the more flexible the model i.e. the more free parameters there are, *ceteris paribus*

the better the fit. However, as the ultimate goal in inductive learning is to perform well on unseen examples, it follows that smaller models are preferred since networks with many parameters will tend to capture the noise as well as the underlying signal [15]. This problem of learning the noise along with the signal is called over-fitting. To prevent over-fitting constraints such as smoothness must be imposed on the solution in the form of regularization. Regularization is a method to discourage complex models by imposing a penalty term in the cost function [15].

Attempting to model a system from observations is known as an inverse problem. Inverse problems attempt to find unknown causes for known consequences, as opposed to the direct problem of finding unknown consequences to known causes. Inverse modeling amounts to finding model parameters given the observed data. Most inverse problems are ill-posed. Hamard [72] first introduced the notion of ill and well-posed problems and provided a short criteria for determining a well-posed problem:

- *A solution to the problem exists*: For every input vector $\mathbf{u} \in \mathbf{U}$, there exists an output vector $y = h(\mathbf{u})$, where $y \in \mathbf{y}$;
- *The solution is unique*: For any pair of input vectors $\mathbf{u}_1, \mathbf{u}_2 \in \mathbf{U}$, it follows that $h(\mathbf{u}_1) = h(\mathbf{u}_2)$ iff $\mathbf{u}_1 = \mathbf{u}_2$;
- *The solution is stable*: meaning the mapping is continuous, for any $\epsilon > 0$ there exists $\xi = \xi(\epsilon)$ such that the condition $C_{\mathrm{w}}(\mathbf{u}_1, \mathbf{u}_2) < \xi$ implies that $C_{\mathrm{y}}(h(\mathbf{u}_1), h(\mathbf{u}_2)) < \epsilon$, where $\mathrm{C}(\cdot, \cdot)$ represents the distance metric between two arguments in their respective spaces.

All problems that fail to satisfy the above criteria are considered ill-posed. It turns out that most real world problems fail to meet this criteria directly and are considered ill-posed.

Although it is usually assumed that the modeling problem has a solution, the solution is rarely unique or stable. For nonlinear systems, uniqueness of the solution is a difficult constraint to satisfy because there usually exist an almost infinite amount of solutions that can fit the data. To make matters worse, no measurement is free of noise. If the model is inherently unstable, small perturbations in the input due to noise can easily become magnified in the output of the system. Solutions to problems ill-posed in nature have no practical use.

How then do we get around the ill-posed problem? The method of regularization is a technique specifically developed for solving ill-posed inverse problems (ill-posed inverse parameter estimation problems). The main idea of regularization is instead of solving the

ill-posed problem, a set of well-posed problems that approximate the ill-posed problems are solved. Regularization theory was first proposed by Tikonov [207] in 1963.

There are two main types of parameter estimation problems: linear and nonlinear. The general model of a linear estimation problem is defined as a system of equations

$$\mathbf{A}\mathbf{w} \approx \mathbf{d} \tag{4.1}$$

where $\mathbf{w} \in \mathbb{R}^m$ is the weight/parameter vector, $\mathbf{d} \in \mathbb{R}^n$ is the target vector, and the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a matrix with $m \leq n$.

In the case of nonlinear systems, the nonlinear model is described as

$$h(\mathbf{w}, \mathbf{u}) \approx \mathbf{d} \tag{4.2}$$

where the weights/parameters of the model are $\mathbf{w} \in \mathbb{R}^m$, the targets are $\mathbf{d} \in \mathbb{R}^n$, and the function $h(\cdot, \cdot)$ is a nonlinear mapping.

Originally, the method of regularization was developed for solving ill-posed problems in linear systems, but has been extended to nonlinear systems [39]. Smoothness is an essential property for nonlinear dynamical systems, and regularization techniques can be used to demand smoothness from the model. Before further discussing regularization, it would be useful to provide a non-rigorous definition of regularization as: *any method of stabilizing the solution to obtain a meaningful result by modifying the original ill-posed problem.*

There are two main objectives in regularization, achieving stability, and uniqueness in the solution with a much lower sensitivity to noise that the original ill-posed problem. There are two main routes to achieving these objectives, truncation [52] and Tikhonov Regularization [208]:

- Truncation: singular value decomposition or total least squares methods are computed, and a lower dimensional summation of the resulting solution matrix is taken in an attempt to eliminate "high frequency" components leaving the remaining problem in well conditioned form [215]. Truncation methods have been shown to outperform other regularization methods, but problems exist such as to what degree of truncation to perform in discrete ill-posed problems.

- Tikhonov Regularization: through the inclusion of a penalty term to the loss function a balance between fitting the model and smoothness can be imposed on the

solution. This method fits naturally into the neural network training framework as it is cast as an optimization problem [168, 218].

$$\min_{\mathbf{w}} \|\mathbf{A}\mathbf{w} - \mathbf{d}\|^2 + \alpha\|\Omega(\mathbf{w})\|^2 \tag{4.3}$$

where $\alpha > 0$ is the regularization parameter that controls smoothness and where $\|\cdot\|$ is the Euclidean norm. The function $\Omega(\cdot)$ is a penalty function that promotes smoothness from the model.

Regularization can be interpreted deterministically or stochastically. In the deterministic sense regularization is nothing more than performing numerical stabilization. For example Tikhonov regularization transforms the unstable solution

$$\mathbf{A}^T\mathbf{A}\mathbf{w} = \mathbf{A}^T\mathbf{d} \tag{4.4}$$

into the stable solution

$$(\mathbf{A}^T\mathbf{A} + \alpha\mathbf{I})\mathbf{w} = \mathbf{A}^T\mathbf{d} \tag{4.5}$$

in which small singular values are filtered out via the stabilization term $\alpha\mathbf{I}$ [208]. Here the matrix $\mathbf{I}$ is the identity matrix.

On the other hand, the stochastic interpretation of regularization models the noise originating from the observations and the model discrepancy as probability distributions. Bayes rule has been used to estimate the regularization parameters given estimates of these distributions [120].

4.2 Regularization in Neural Network Learning

The complexity of a model is determined by the number of free parameters of the model. By increasing the number of free parameters, the flexibility of the models increases, which increases the danger of overfitting [25, 36, 57]. Conversely decreasing the number of free parameters decreases the flexibility of the model and decreases the chance of overfitting but increases the chance of not having a powerful enough model. Regularization in previous work has been achieved through cross validation training (early stopping), adding noise to the training set, neural ensembles (i.e. combining multiple neural models), and constructive/destructive methods (i.e. adding/removing neurons during training) [53].

Training with noise has been shown to be equivalent to regularization [16, 110, 229]. Noise is added to the training data which tends to provide a range of scattered points around the so called "true point" which prevents the neural network from over-fitting the data. However, it is not clear how much noise to add and when to stop training.

By combining a mixture of models it has been shown that that the performance can be improved [91, 163, 164]. The idea behind using comities of models is that by training individual models with higher variance than bias and then by combining the models, the variances will be averaged over all the models. The main drawback of this approach is the need for an additional modeling step to combine the forecasts [34]. The committee of models may still need to undergo model selection to discard the poor performing models [34]. Furthermore, it is unclear when to stop training the population.

Popular methods for complexity control can be categorized into two main categories, penalized likelihood regression [208] and predictive assessment [15], in which there is a major overlap between the two categories.

Penalized likelihood methods control the complexity of the model through a penalty term $\Omega(\mathbf{w})$ placed in the loss function of the model.

$$C(\mathbf{w}) = E(\mathbf{w}) + \alpha\Omega(\mathbf{w})$$

The loss function controls the tradeoff between two conflicting goals, where the first term $E(\mathbf{w})$ encourages fitting the model closely to the data and the second term $\Omega(\mathbf{w})$ encourages smoothness of the model (i.e. discourages fitting the data precisely).

Predictive assessment methods use some sort of cross validation technique by segmenting the training set into two or more subsets where one or more subsets are saved for testing and the remaining subsets are used for training. Predictive assessment techniques are often used to find the parameters for the penalized likelihood loss functions. Early stopping is the most well known predictive assessment technique that involves breaking the data set into two sets, a training set and a validation set. The model is then trained on the training set and, the model performance is evaluated on the validation set. Training is stopped when the validation error begins to increase. This technique has been shown to be equivalent to regularization [16]. Cross validation in general and early stopping in particular has several drawbacks such as the model can not be trained on the full data set resulting in a biased estimator towards the validation set.

To circumvent the limitations of predictive assessment, a novel algorithm for inferring regularization coefficients (RNN weights) and regularization hyperparameters in a penal-

ized likelihood framework is proposed. First, however, an overview of regularization is provided.

4.3 Regularization for Linear Systems

To illustrate the forward problem, the linear system presented in Equation 4.1 is rewritten as:

$$\mathbf{d} = \mathbf{A}\mathbf{w} + \epsilon \tag{4.6}$$

where ϵ is the uncertainty in the observations (i.e. noise in the observations), and $\mathbf{d}$ is the data. The direct problem involves using the matrix $\mathbf{A}$ and the vector $\mathbf{w}$ to find $\mathbf{d}$. The inverse problem involves finding the regression coefficients $\mathbf{w}$ given $\mathbf{d}$ and matrix $\mathbf{A}$. This operation requires the matrix inverse to solve for $\mathbf{w}$ assuming that $\mathbf{A}$ is a square matrix.

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{d} \tag{4.7}$$

or if $\mathbf{A}$ is not square the standard ordinary least squares procedure is usually employed

$$\mathbf{w} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{d} \tag{4.8}$$

However, due to noise corrupted data, finding the exact solution is not always straightforward. The matrix $\mathbf{A}$ is usually nearly singular, and thus finding $\mathbf{A}^{-1}$ is generally non-trivial. Least squares approaches can result in spurious solutions as $\mathbf{A}^T\mathbf{A}$ is generally ill-conditioned. This means that the inverse can magnify the noise in $\mathbf{d}$ leading to poor estimates of $\mathbf{w}$.

As it has already been shown that the standard solutions to the inverse problem generally fail when uncertainty in the data is present, attention is focused on methods that are robust in the face of uncertainty.

Tikhonov introduced penalized least squares to condition the nonlinear inverse operator [207]. This method has been used successfully to obtain solution estimates for many ill-posed problems as in [196] and references their in. A penalty term and the regularization hyperparameter are augmented to the quasi-solution expression as shown below

$$E(\mathbf{w}) + \alpha\Omega(\mathbf{w})$$

The first term $E(\mathbf{w})$ is the error function, and the second term is a penalty function that promotes constraints (such as smoothness) on the solution $\mathbf{w}$ given that

- $\Omega(\mathbf{w})$ is continuous, nonnegative and everywhere dense in $\mathbf{W}$;
- the true solution $\mathbf{w}_t$ belongs to the domain of $\Omega(\mathbf{w})$;
- for every positive number d, the set of elements $\mathbf{w}$ for which $\Omega(\mathbf{w}) \leq d$ is a compact subset of $\mathbf{W}$.

There have been various functions proposed for the penalty term in the penalized cost function [83, 64, 222]. Weight decay [83] is one of the simplest forms of the regularization component $\Omega(\mathbf{w})$, where the regularization function is defined by:

$$\Omega(\mathbf{w}) = \mathbf{w}^T\mathbf{w} \tag{4.9}$$

Weight decay promotes smoothness in the mapping by penalizing large weights, and forcing the existing small weights towards zero. This is beneficial to generalization as weights with large absolute values can cause rough or abrupt changes in the output of the network. By discouraging large weight magnitudes, a smoother output is obtained. As small weights are forced towards zero, a number of the weights are effectively removed from the network, resulting in a parsimonious model. Simultaneous minimization of $\Omega(\mathbf{w})$ and $E_D(\mathbf{w})$ is possible by minimizing the function

$$C(\mathbf{w}) = E(\mathbf{w}) + \alpha\Omega(\mathbf{w}) \tag{4.10}$$

where α is a nonnegative coefficient called the regularization parameter. The regularization parameter controls the tradeoff between the amount of regularization and the amount of fitting through the minimization of $E(\mathbf{w})$. When $\alpha = 0$, i.e. no regularization, the quasi-solution is obtained and when $\alpha \rightarrow \infty$ the result is a parametric model.

4.4 Regularization for Nonlinear Systems

Regularization theory was applied to neural networks by Poggio and Girosi [168]. For neural networks two principles of regularization are paramount, smoothness and simplicity. Characteristic of neural network models is the usually high dimension of the $\mathbf{w}$ parameter vector which is far from the goal of parsimonious models.

Referring back to Equation 4.2, the data is now described by the nonlinear system contaminated by white noise

$$d_t = h(\mathbf{u}_t, \mathbf{w}_t) + \varepsilon_t \tag{4.11}$$

assuming that $\mathbf{w}$ is obtained through least squares optimization over T training patterns and $\varepsilon_t \sim \mathcal{N}(\mu, \sigma^2)$ is the white noise process that is normally distributed with mean $\mu = 0$ and variance σ^2. The model error variance $\overline{V}_T$ over many different estimates of $\mathbf{w}$ can be approximated by:

$$\overline{V}_T \approx \sigma^2\Big(1 + \frac{m}{T}\Big) \tag{4.12}$$

where m is the number of weights/parameters, and σ^2 is the variance of the white noise. Assuming there is no more data available, the only way to reduce the variance is by reducing the size of m. This is where regularization with weight decay steps into the picture.

By adding a penalty term to the loss function, regularization techniques with a weight decay functional can be used to reduce variance in nonlinear systems

$$C_T(\mathbf{w}) = \sum_{\tau=1}^{T} E_\tau(\mathbf{w}) + \alpha\Omega(\mathbf{w}) \tag{4.13}$$

Assuming the standard sum of squared error function, the regularized loss function can be written as

$$C_T(\mathbf{w}) = \sum_{\tau=1}^{T} E_\tau(\mathbf{w}) + \alpha\Omega(\mathbf{w}) = \mathbf{e}(\mathbf{w})\mathbf{e}^T(\mathbf{w}) + \alpha\mathbf{w}^T\mathbf{w} \tag{4.14}$$

Through regularization with weight decay, the variance of the model can be reduced, assuming that superfluous weights are diminished to values effectively zero. The new model variance is approximated by

$$\overline{V}_\gamma \approx \sigma^2\Big(1 + \frac{\gamma}{T}\Big) \tag{4.15}$$

where $\gamma \leq m$ is the effective number of parameters [15], computed by

$$\gamma = \sum_{i=1}^{m} \frac{\lambda_i}{\lambda_i + \alpha} \tag{4.16}$$

where the ith eigenvalue of the Hessian is λ_i and α is the regularization parameter. The regularization parameter $\alpha > 0$, implies that if $\lambda_i > \alpha$ by a significant amount, the quantity $\lambda_i/(\lambda_i + \alpha)$ will be somewhere near 1. On the other hand, if $\lambda_i < \alpha$ by a significant amount, the quantity $\lambda_i/(\lambda_i + \alpha)$ will approach zero. RNN modeling typically uses a large number of parameters. It turns out that γ, the effective number of parameters, can be be significantly smaller than m, the total number of parameters. Regularization for RNNs have the potential to reduce model variance compared to RNNs trained without regularization.

4.4.1 Determination of Parameter α

Although regularization provides a framework for smooth models, finding the optimal smoothness parameter(s) and the correct number of degrees of freedom is difficult to predetermine in advance, as these quantities are dependent on the number of the training samples, the distribution of the noise in the samples, and the complexity of the underlying phenomenon to be modeled.

The standard method for determining the value of α in linear ill-posed problems is through a trial and error process known as the L curve method. Both components of the penalized error function are minimized, i.e. the error function $E(\mathbf{w})$ and the penalty term $\Omega(\mathbf{w})$ are minimized for various values of α. Then they are plotted against each other in a log-log scale. Generally an L shaped curve is obtained. The L curve shows the tradeoff between regularization and model fitting. The horizontal part of the L plots the errors (or model sensitivity to α) from the regularization and the vertical part of the L plots the errors due to noise contaminated measurements. The optimal setting of α corresponds to the value at the corner of the L.

Bayesian methods have become popular for inferring the regularization parameters in nonlinear models such as neural networks [122]. In the Bayesian regularization framework, the model weights and the regularization hyperparameters are assumed to be described by probability distributions. Through an iterative learning process, the parameters and hyperparameters are updated via Bayes rule resulting in a finely tuned model with characteristics that match the training data in a statistical sense. The next section discusses the Bayesian method for inference of RNN parameters.

4.5 Regularized Bayesian RNN Training

The key principle of Bayesian analysis is that uncertain quantities are modeled as probability distributions, and inference is performed by constructing the posterior probability distributions for all unknown entities in a model, given a data sample. To use the model, marginal distributions are constructed for all entities of interest such as the predictions in non-parametric regression.

The Bayesian approach was first introduced to Neural Networks by [17, 121, 148]. In his thesis, MacKay pointed out that one of the largest weaknesses of neural networks is that when building a neural model, the design "depends on a considerable number of design choices, most of which are currently made by rules of thumb and trial and error." Of these so called "design choices," the most important and difficult is specifying the complexity of the model.

The Bayesian approach to building neural models implicitly handles the design considerations by defining vague (non-informative) priors for the hyperparameters. Through estimation of the hyperparameters, the unknown degree of complexity of the model is determined. Model selection is performed by averaging the resulting model over all model complexities weighted by their posterior probability given the data sample. The model can be allowed to have different complexity in different parts of the model by grouping the parameters to have a common hyperparameter or by assigning a hyperparameter to each parameter.

The goal of RNN modeling is to properly learn the model regions of both low and high nonlinearities in the data. The overfitting may vary in the different regions of the model. Since each weighted term contributes a different curvature to the overall model, there is a need to manipulate the uncertainty of each weight separately in order to adjust the smoothness of the functional mapping. Adapting the weights with local regularization helps to achieve accurate quantification of each term to the overall output, and local control over the particular term nonlinearities so as to tune the model curvature to the data. A set of individually tunable priors $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]$ (which come from a zero mean gaussian [122]) are adopted, with variance $\alpha_i = \sigma_{w_i}^{-2}$ which allow each weight to be influenced by its local regularization hyperparameter [212].

Applied to neural network training the Bayes' rule for training with hyperparameters is [15, 122]:

$$p(\mathbf{w}|d, \mathbf{U}, \boldsymbol{\alpha}, \beta^{-1}) = \frac{p(d|\mathbf{U}, \mathbf{w}, \beta^{-1})p(\mathbf{w}|\boldsymbol{\alpha})}{p(d|\mathbf{U}, \boldsymbol{\alpha}, \beta^{-1})} \tag{4.17}$$

which is proportional to the data likelihood $p(d|\mathbf{U}, \mathbf{w}, \beta^{-1})$, the weight prior $p(\mathbf{w}|\boldsymbol{\alpha})$, and inversely proportional to the evidence $p(d|\mathbf{U}, \boldsymbol{\alpha}, \beta^{-1})$. Here $\mathbf{U}$ denotes the matrix of all input vectors, and β is a measure of the noise contained in the target data d. The evidence approach assumes a Gaussian approximation to the weight posterior through a second-order Taylor expansion of the cost function to enable analytical tractability [122].

The likelihood $p(d|\mathbf{U}, \boldsymbol{\alpha}, \beta^{-1})$ reflects the neural network accuracy on the training set. It is a distribution

$$p(d|\mathbf{u}) = \mathcal{N}(h(\mathbf{w}, \mathbf{u}), \sigma_y^2) \tag{4.18}$$

with variance σ_y^2, which is represented by a noise hyperparameter $\beta = \sigma_y^{-2}$. Assuming zero-mean normally distributed noise implies that the likelihood of the entire data set is:

$$p(d|\mathbf{U}, \mathbf{w}, \beta^{-1}) \simeq \frac{1}{Z(\beta)} \exp\Big(-\frac{1}{2}\beta E_t(\mathbf{w})\Big) \tag{4.19}$$

where $Z(\beta) = (2\pi\beta^{-1})^{t/2}$ is a normalizing constant [15, 122].

The weight prior $p(\mathbf{w}|\boldsymbol{\alpha})$ accounts for the subjective beliefs σ_{w_i} about the weights and the shape of the network mapping. The individual hyperparameters α_i are given a zero-mean normal distribution with inverse variance $\boldsymbol{\alpha}^{-1}$, that is:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \mathcal{N}(\mathbf{w}|0, \boldsymbol{\alpha}^{-1}) \tag{4.20}$$

This leads to the following zero-mean factorized prior [212]:

$$p(\mathbf{w}|\boldsymbol{\alpha}) \simeq \prod_{i=1}^{m} \frac{1}{Z_w(\alpha_i)} \exp\Big(-\frac{1}{2}\alpha_i w_i^2\Big) \tag{4.21}$$

where $Z_w(\alpha_i) = 2\pi/\alpha_i^{m/2}$ is a normalizing constant [15, 122].

The instantaneous Gaussian weight posterior at a particular time step then becomes:

$$p(\mathbf{w}|d, \mathbf{U}, \boldsymbol{\alpha}, \beta^{-1}) \simeq \frac{1}{Z(\boldsymbol{\alpha}, \beta)} \exp -\frac{1}{2}\Big(-\beta E_t(\mathbf{w}) + \mathbf{w}^{\mathrm{T}} \operatorname{diag}(\boldsymbol{\alpha})\mathbf{w}\Big) \tag{4.22}$$

where Z is a normalizer produced by $Z_D(\beta)$ and $Z_w(\alpha_i)$ [15, 122].

The individual hyperparameters α_i control the inference of the weight distribution, which impacts the weight value and the variance of their distribution. Individual hyperparameters are more advantageous to simple weight decay regularization (which uses a uni-

form hyperparameter) as each weight can be individually tuned. Simple uniform weight decay imposes uniform penalty in the cost function, which is less efficient for achieving good generalization. Moreover, the uniform weight decay regularization is inconsistent with the scaling properties of the mappings [15].

The weight posterior distribution embodies all statistical information about the model parameters. However, neural networks are highly nonlinear models where the posterior distribution of the weights may have several modes. One approach to overcome this difficulty is to assume that the weight posterior is sharply peaked, and to approximate it using a local Taylor expansion of the cost function (up to the second-order) [122].

The intention to perform sequential estimation requires to consider the cost accumulated from the beginning of the time interval up to the current moment t. The performance criterion is reformulated to accommodate Bayesian regularization, which leads to the following cost function:

$$C_t(\mathbf{w}) = \frac{1}{2}\Big(\sum_{\tau=1}^{t} \beta E_\tau(\mathbf{w}) + \mathbf{w}^{\mathrm{T}}(\alpha_i \mathbf{R}_t)\mathbf{w}\Big) \tag{4.23}$$

where $\mathbf{R}_t$ is a special matrix (mxm) having zeros everywhere except on the diagonal at positions ii, that is $[\mathbf{R}_t]_{ii} = z_i^{-1}$. The quantity $z_i = (T \text{ div } m)$ if $t < m * (T \text{ div } m)$ or a very large number otherwise [153], and the operator div is the integer division operator.

4.5.1 Recursive Weight Estimation

The training objective in temporal modeling is to evolve the weight posterior $p(\mathbf{w}_t|d_t, \mathbf{U}, \boldsymbol{\alpha}, \beta^{-1})$ progressively with the arrival of the next training vector. This can be accomplished by sequential minimization of the Bayesian cost function. A distinguishing characteristic of the algorithm is the incremental distribution of the regularization over the entire training set while optimizing the weights. Such a technique is necessary because the total regularization effect has to be distributed in parts when processing sequentially arriving data. This can be achieved through augmentation of the covariance matrix by fractions of the total regularization.

At the minimum of the cost function, the optimal weight estimate of the gradient at time $t-1$ is zero. Under the assumption that this is an instantaneous minimum, the

following recursive Bayesian weight training rule is obtained:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \beta \mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{j}_t(\mathbf{w}_{t-1})e_t(\mathbf{w}_{t-1}) - \alpha_i \mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{R}_t\mathbf{w}_{t-1} \tag{4.24}$$

where $\mathbf{w}_t$ is the weight vector, $\mathbf{H}_t^{-1}(\mathbf{w}_{t-1})$ is the inverted regularized Hessian, and α_i is the prior hyperparameter with index $i = (t+1) \bmod m$, where mod is the modulus operator. The term $\mathbf{j}_t(\mathbf{w}_{t-1})$ includes the model derivatives which can be computed using the RTRL algorithm, given by equations (3.31).

The error $e_t(\mathbf{w}_{t-1})$, which is embedded in the second term of Equation 4.24 determines the local search direction, while the noise hyperparameter β impacts its magnitude. The third term (in Equation 4.24) $\alpha_i \mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{R}_t\mathbf{w}_{t-1}$ accounts for the regularization that is applied partially until the number of training examples reaches $m * (T \; div \; m)$. The index of the next hyperparameter α_i, $1 \leq i \leq m$ is such that $i = (t+1) \bmod m$ and the value of the hyperparameter is scaled by the matrix $\mathbf{R}_t$.

Equation (4.24) provides a general rule for second-order recursive Bayesian training that can be applied to training RNNs. It requires several iterations to tune the hyperparameters. The generality of this weight update equation can be explained as follows: 1) if one global regularization hyperparameter α is used and the noise variance is assumed $\beta = 1.0$, it degenerates to the classical Levenberg-Marquardt method; 2) if small regularizers are used $\alpha_i \rightarrow 0$ and $\beta = 1.0$ it reduces to Newton's method; and 3) when regularizers are very large $\alpha_i \rightarrow \infty$ it becomes equivalent to gradient descent.

4.5.2 Recursive Covariance Update

Central to the Levenberg-Marquardt method is the estimation of the inverted regularized Hessian matrix $\mathbf{H}_t^{-1}(\mathbf{w}_{t-1})$. The inverse Hessian is recursively estimated at each time step as a function of its past values $\mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1})$ and the current values of the regularization hyperparameters α_i. The dynamic Hessian is updated via:

$$\mathbf{H}_t(\mathbf{w}_{t-1}) = \beta \mathbf{J}_{t-1}^{\mathrm{T}}(\mathbf{w}_{t-1})\mathbf{J}_{t-1}(\mathbf{w}_{t-1}) + \mathbf{\Lambda}_t \tag{4.25}$$

Here $\mathbf{\Lambda}$ is the distributed version of the regularization matrix $\mathbf{\Lambda}_t^{-1} = [1 \;\; 0; 0 \;\; \alpha_i[\mathbf{R}_t]_{ii}]$ where ii is the diagonal position of the matrix $\mathbf{R}$.

The sequential modification of the inverted Hessian is derived by writing it in the form

$$\mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) = \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) + \beta \mathbf{j}_{t-1}^{*}(\mathbf{w}_{t-1}) \mathbf{\Lambda}_{t-1}^{-1} \mathbf{j}_{t-1}^{*\mathrm{T}}(\mathbf{w}_{t-1}) \tag{4.26}$$

The notation $\mathbf{j}_{t-1}^{*}(\mathbf{w}_{t-1})$ specifies the augmented column vector $\mathbf{j}_{t-1}^{\mathrm{T}}(\mathbf{w}_{t-1})$ by a column vector with zero elements except for the element at position $i = t - 1 \bmod m$, and more precisely

$$\mathbf{j}_{t-1}^{*}(\mathbf{w}_{t-1}) = [\mathbf{j}_{t-1}^{\mathrm{T}}(\mathbf{w}_{t-1}); 0 \ldots 0 \;\; 1_i \ldots 0]^{\mathrm{T}} \tag{4.27}$$

After applying the matrix inversion lemma [127] and some simplifications, the following regularized recursive inverse Hessian update is reached (see Appendix C):

$$\mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) = \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) - \beta \mathbf{H}_{t-1}(\mathbf{w}_{t-1}) \mathbf{j}_{t-1}^{*}(\mathbf{w}_{t-1}) \mathbf{S}_{t-1}^{-1} \mathbf{j}_{t-1}^{*\mathrm{T}}(\mathbf{w}_{t-1}) \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) \tag{4.28}$$

where

$$\mathbf{S}_{t-1} = \left(\beta \mathbf{j}_{t-1}^{*\mathrm{T}}(\mathbf{w}_{t-1}) \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) \mathbf{j}_{t-1}^{*}(\mathbf{w}_{t-1}) + \mathbf{\Lambda}_t \right) \tag{4.29}$$

and the inverted distributed regularization matrix is $(\mathbf{\Lambda}_t^{-1})^{-1}$ defined above. The initial values on the diagonal of $[\mathbf{H}_t^{-1}(\mathbf{w}_{t-1})]_{ii}$ are set to small values.

It should be noted that the proposed partial regularization negligibly increases the computational complexity of the algorithm because $\mathbf{\Lambda}_t$ is a two dimensional matrix whose inversion can be computed analytically (with numerically stable formula) [96].

4.5.3 Bayesian Hyperparameter Updates

The Bayesian inference aims at maximization of the weight posterior distribution with plausible hyperparameters. The most plausible hyperparameters are those that maximize their likelihood conditioned on the data, known as the evidence for the hyperparameters [122]. The procedure for evidence maximization in the case of normal, Gaussian data leads to analytical formula for the hyperparameters.

The weight prior hyperparameters are adjusted to maximize the probability of the weight prior distribution as follows [122]:

$$\alpha_i = \frac{\gamma_i}{w_i^2} \tag{4.30}$$

where γ_i is defined as follows

$$\gamma_i = 1 - \alpha_i[\mathbf{H}_T^{-1}(\mathbf{w}_{t-1})]_{ii} \tag{4.31}$$

Note that for each w_i, there is one local α_i that is computed at the end of the interval. Each such α_i is sequentially distributed while passing through the series.

The output noise hyperparameter is updated to adjust the model to the characteristics of the training data by the equation [122]:

$$\beta = \frac{T - \gamma}{\mathbf{e}(\mathbf{w})\mathbf{e}(\mathbf{w})^{\mathrm{T}}} \tag{4.32}$$

where $\mathbf{e}(\mathbf{w}) = [e_1(\mathbf{w}), \ldots, e_T(\mathbf{w})]$ is the vector of errors, and γ is the effective number of parameters (which is less than the actual number of parameters) computable as [137]

$$\gamma = m - \sum_{i=0}^{m} \alpha_i[\mathbf{H}_T^{-1}(\mathbf{w}_{t-1})]_{ii} \tag{4.33}$$

This probabilistic training procedure requires several passes through the data until reaching an acceptably low error. After each pass, the hyperparameters are re-estimated while keeping the weights fixed.

4.5.4 Bayesian Network Pruning

The evidence framework [122] provides a means for automatic relevance determination (ARD) of the weights (i.e. automatic identification of the significant weighs). The suggestion is that when a local hyperparameter shrinks its corresponding weight toward zero, this weight can be pruned.

The approach employs a Bayesian pruning procedure which has the advantage of evaluation the significance of all weights considered together, pointing out the individual weights that are irrelevant in the context of the model as a whole. The overall effect from pruning is increasing the smoothness of the neural network mapping, and thus improving its generalization [15].

RBLM-RNN training is an iterative process of three steps: 1) computing the optimal network weights, 2) re-estimation of the prior hyperparameters and the noise hyperparameter, and 3) pruning the network topology by setting the statistically insignificant weights

to zero. Weight pruning takes place when the prior hyperparameter becomes greater than a predefined threshold.

4.5.5 Bayesian Confidence Interval Estimation

The presented algorithm provides useful partial results that can be taken to estimate confidence intervals, necessary for statistical diagnostics. The confidence intervals show the variance of the network predictive distribution, and thus offer information about the belief in the learned model. Here Bayesian confidence intervals are considered in order to quantify the degree of belief in the model, through the uncertainties in the weights and the uncertainty in the data [146]. These uncertainties are the hyperparameters computed by the training algorithm.

Bayesian confidence intervals are defined as: $h(\mathbf{w}, \mathbf{u}_t) \pm z_{.025}\sigma_{MP}$, where σ_{MP} is the variance of the forecast distribution, and $z_{.025}$ is the critical point of the standard normal distribution. These are bands within which the true underlying process is expected to fall with 95% confidence. Such error bands of the model can be obtained as follows [137]:

$$h(\mathbf{w}, \mathbf{u}_t) \pm z_{.025} \cdot \sqrt{\frac{1}{\beta} + \mathbf{j}_{t-1}^{\mathrm{T}}(\mathbf{w}_{t-1})\mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1})\mathbf{j}_{t-1}(\mathbf{w}_{t-1})} \tag{4.34}$$

where β is the output noise variance, $\mathbf{j}_{t-1}(\mathbf{w}_{t-1})$ is the gradient vector, and $\mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1})$ is the inverted regularized Hessian matrix at the particular time step. The confidence intervals can be generated sequentially at each training example by computing the variance of the prediction error $\mathbf{j}_{t-1}^{\mathrm{T}}(\mathbf{w}_{t-1})\mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1})\mathbf{j}_{t-1}(\mathbf{w}_{t-1})$ [99]. The calculation of this quantity uses partial Jacobian-Hessian-Jacobian products that are precomputed during the recursive covariance update (4.28).

4.6 Experimental Results

The objective of this chapter was to develop an improved dynamic RNN for time series modeling. Tests were conducted to evaluate the performance of the proposed model in terms of its predictive ability and computational cost of training. Studies into time series modeling were carried out using: a group of chaotic nonlinear dynamical systems, the benchmark laser intensity series from data set A of the Santa Fe time series prediction competition, and a real-word electricity spot price series (publicly available

at `www.nordpool.no`). The following experiments are all cases of nonlinear regression for single-input-single-output (SISO) systems.[1] The residuals were measured by the normalized mean squared error (nMSE) computed by $nMSE = (1/(T\sigma_y^2)) \sum_{t=1}^{T}(d_t - h(\mathbf{w}, \mathbf{u}_t))^2$, where σ_y is the standard deviation of the training data.

The performance of the Recursive Bayesian Levenberg-Marquardt RNN (RBLM-RNN) is related to other neural time series models, such as the feed-forward multi-layer perceptron (MLP) trained with the extended Kalman filter (EKF), a feed-forward neural network trained with the Bayesian Levenberg-Marquardt (MLP-BLM) algorithm, an RNN trained with RTRL, an RNN trained with BPTT, and an RNN trained with the EKF (using RTRL derivatives). In all simulations, the weights of the networks were initialized with random uniformly distributed values in the range of $[-0.1, 0.1]$. The computational cost of training each algorithm was measured by the average time it took for each model to reach a pre-specified training error. For simulations concerned with evaluating computational cost, each run began with the same initial conditions (weight vector, state vector, and number of nodes) for each of the RNNs. Similarly, all feed forward networks began with the same initial conditions (weight vector and number of nodes). The structure of the networks in the computational cost simulations were the same as in the prediction simulations. The average time to convergence was taken for 50 runs where at each run a different set of initial conditions was randomly chosen.

4.6.1 Modeling Chaotic Processes

The first experiment studied the RBLM-RNN performance on a group of benchmark equation based nonlinear dynamical systems. The group of chaotic processes included the Henon map [81], the Ikeda map [90], the Lorenz equations [117] and the Rössler attractor [180]. Plots of the attractors are shown in Figures 4.1 and 4.2. Each training set was 1000 time steps long, and each test set was 250 time steps long. White noise with standard deviation of 0.05 was added to the training sets of each series.

The RBLM-RNN and the other recurrent networks, RNN-BPTT, RNN-RTRL, and RNN-EKF, were initialized with 3 hidden neurons, 1 input and 1 output neuron. For the RBLM-RNN and MLP-BLM models, the initial hyperparameters were: $\alpha_i = 10^{-4}$ and $\beta = 1.0$ (these values were chosen so as to constrain the initial weight vector to small numbers). The feed-forward networks were constructed with 3 hidden neurons, and used

[1] The derivations in this chapter can be extended for mulitple-input-mulitple-output (MIMO) systems in a straightforward way.

an input window chosen according to the embedding dimension of the series (determined by false nearest neighbors). The initial diagonal elements of the covariance matrix for all filters were set to 10^3 [76].

Table 4.1: Residuals of the in-sample fitting and out-of-sample predictions on benchmark chaotic time series.

Model	Metric	Chaotic Attractors							
		Henon		Ikeda		Rossler		Lorenz	
		train	test	train	test	train	test	train	test
MLP-EKF	nMSE	0.00037	0.00086	0.00035	0.00132	0.00025	0.00193	0.00023	0.00162
	weights	25		25		25		37	
MLP-BLM	nMSE	0.00032	0.00081	0.00049	0.00127	0.00047	0.00101	0.00033	0.00096
	weights	21		24		23		33	
RNN-BPTT	nMSE	0.00055	0.00110	0.00058	0.00162	0.00070	0.00311	0.00056	0.00185
	weights	19		19		19		19	
RNN-RTRL	nMSE	0.00058	0.00103	0.00056	0.00159	0.00071	0.00312	0.00057	0.00172
	weights	19		19		19		19	
RNN-EKF	nMSE	0.00043	0.00072	0.00043	0.00101	0.00060	0.00191	0.00036	0.00121
	weights	19		19		19		19	
RBLM-RNN	nMSE	0.00036	0.00068	0.00048	0.00081	0.00057	0.00092	0.00036	0.00090
	weights	16		14		17		16	

Table 4.1 gives the experimental results of training and single-step ahead forecasting on the chaotic benchmark series. These results indicate that the RBLM-RNN outperforms the other competing models in the sense of nMSE. It can be observed that the RBLM-RNN is more sparse than the other networks, which was achieved by pruning weights whose hyperparameter values were greater than a predefined threshold $\alpha_{MAX} > 10^3$ (this parameter was found through multiple trials, too small a setting and no weights get pruned, and too large a setting and too many weights get pruned), while doing 10 iterative cycles over the data. The computational cost of training in terms of cpu time is given in Table 4.2. Although the RBLM-RNN is not the lowest in terms of computations per epoch, its total training time is similar to training times of the feed-forward networks.

4.6.2 Laser Intensity Series

The second experiment studied data taken from the Santa Fe time series competition (Nh3 laser intensity pulses) [88] in order to test the effectiveness of the RBLM-RNN on "real world" data. Building a model from this series is challenging due to the chaotic nature of the laser and only three occurrences of the so called "intensity collapse" in the training set. A plot of this series and its attractor are shown in Figure 4.3. All recurrent networks:

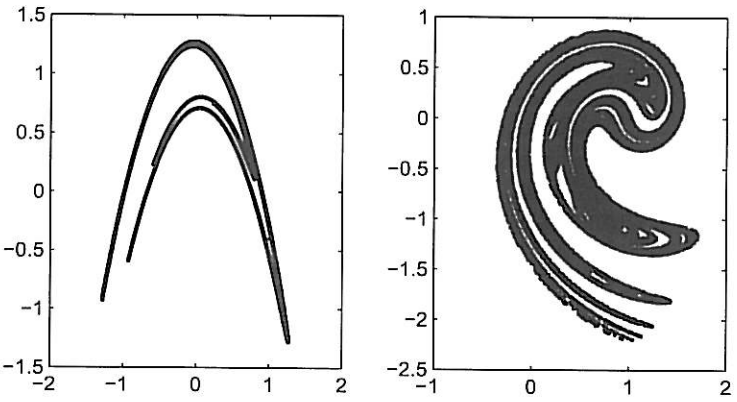

Figure 4.1: Plots of the Henon and Ikeda map. The Henon map is a one dimensional iterated function system with correlation dimension estimated at 1.258 with parameters of the governing equation set to $a = 1.4$ and $b = 0.3$ The Ikeda map is a system of equations representing the plane wave interactivity in an optical ring laser. The parameters of the equation were set to $a = 1.0$, $b = 0.9$, $c = 6.0$ and $\phi = 0.4$ and the correlation dimension was estimated at 1.59. Both correlation dimensions were estimated from the time series data through the Grassberger-Procaccia method [69].

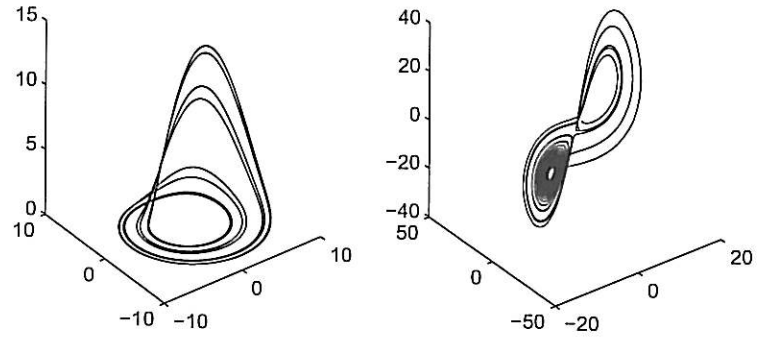

Figure 4.2: Plots of the Rössler and Lorenz attractors. The Rössler attractor is generated by a system of three differential equations having coefficients $a = 0.2$, $b = 0.2$, and $c = 4.6$ which lead to an estimated correlation dimension of 2.013. The Lorenz attractor is a nonlinear three dimensional system that provides a simplified model of convection in the atmosphere. For the parameter settings (Prandtl number $\sigma = 10$, and the Rayleigh number $\rho = 28$ and $\beta = 8/3$) used in this experiment, the system exhibits chaotic behavior with correlation dimension estimated at 2.062. The Grassberger-Procaccia method was used for the estimation of the correlation dimensions of both series.

Table 4.2: Computational cost of training on benchmark chaotic time series.

Model	Metric	Chaotic Attractors							
		Henon		Ikeda		Rossler		Lorenz	
		mean	std dev	mean	std dev	mean	std dev	mean	std dev
MLP-EKF	CPU Time	17.77	2.18	14.68	1.96	15.45	2.03	18.99	2.39
	Epochs	77.28	12.75	63.84	10.31	67.2	10.09	73.90	12.73
MLP-BLM	CPU Time	14.71	1.46	12.15	0.96	12.78	1.17	16.60	1.91
	Epochs	50.72	5.30	41.90	4.97	44.1	4.15	48.51	4.98
RNN-BPTT	CPU Time	21.28	11.42	22.47	10.65	26.88	13.42	21.05	10.58
	Epochs	150.15	59.76	96.82	71.30	82.92	68.43	99.96	80.92
RNN-RTRL	CPU Time	29.62	10.67	28.47	10.93	25.76	9.31	28.33	11.26
	Epochs	185.15	42.55	152.95	34.45	161.01	46.34	177.1	49.51
RNN-EKF	CPU Time	33.04	4.87	27.29	1.81	28.72	1.89	31.60	2.16
	Epochs	86.94	15.90	71.82	13.34	75.6	14.21	83.16	14.12
RBLM-RNN	CPU Time	19.83	1.35	14.71	2.26	15.02	2.41	19.10	2.12
	Epochs	45.06	4.11	38.92	4.13	41.7	4.89	43.27	4.67

RBLM-RNN, RNN-BPTT, RNN-RTRL and RNN-EKF were implemented for this task with 3 hidden neurons, 1 input and 1 output neuron. The initial hyperparameters for the RBLM-RNN and MLP-BLM were: $\alpha_i = 10^{-4}$ and $\beta = 1.0$. The feed-forward networks were constructed with 4 hidden neurons and input vector of size 10, i.e. the network is initialized with 49 weights. After multiple trials, it was found that the the RBLM-RNN and MLP-BLM perform best on the laser series with diagonal elements of the covariance matrix initialized to values of 10^4. Figures 4.4 and 4.5 provide plots of the predictions and the prediction errors of this series respectively. Training took place on the first 1000 data points and the networks were evaluated in a one-step-ahead prediction task over the next 100 data points.

The results from the conducted experiments are given in Tables 4.3 and 4.4. All models fit the series well, however the RBLM-RNN performs best on out-of-sample forecasts. One may be inclined to think that the main reason for this is its good generalization due to the Bayesian tuning of the regularization hyperparameters. Without Bayesian regularization, the MLP-EKF, RNN-BPTT, RNN-RTRL and RNN-EKF tend to overfit the training series because they achieve very low training error, but high prediction error. The feed-forward network trained by the Bayesian Levenberg-Marquardt algorithm is second best after the RBLM-RNN. The mean convergence time for training the RBLM-RNN on the laser series is higher than the feed-forward models, but the lowest of the recurrent models. This is partly due to the fact that the RTRL algorithm requires significantly more computations per epoch than ordinary back propagation. Yet, the computational times of

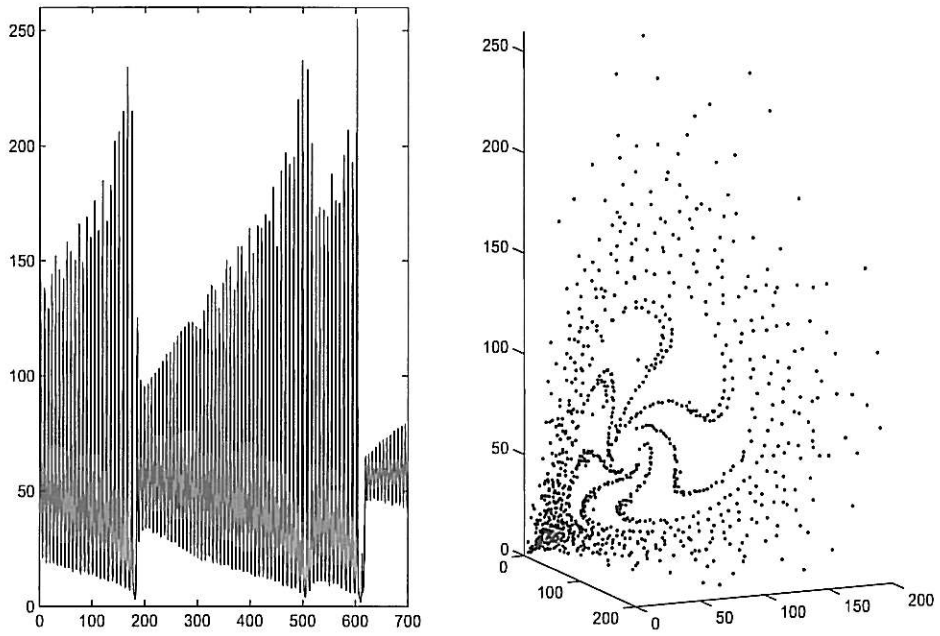

Figure 4.3: Laser time series and return map.

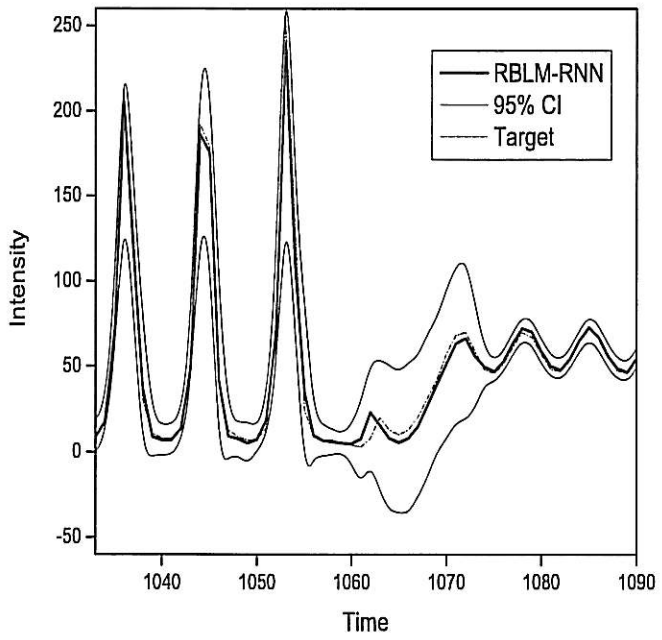

Figure 4.4: Out-of-sample one-step-ahead forecast of the RBLM-RNN and confidence intervals on the Laser time series. The predictions begin at point 1001, and continue for 100 steps into the future

the RBLM-RNN and the feed-forward networks are similar due to the small number of free parameters required to model the laser for the RNN (delay window of size 1) and large number of free parameters required for the same task in the feed forward network. Further lowering the total computational time of the RBLM-RNN is the fast convergence of the RBLM algorithm, which had the lowest mean number of epochs for convergence of all the algorithms considered.

Table 4.3: Residuals from modeling Laser intensities.

Model	weights	nMSE(*training*)	nMSE(*testing*)
MLP-EKF	49	0.00031	0.00468
MLP-BLM	37	0.00088	0.00093
RNN-BPTT	19	0.00032	0.01092
RNN-RTRL	19	0.00036	0.00876
RNN-EKF	19	0.00035	0.00436
RBLM-RNN	17	0.00045	0.00060

Table 4.4: Computational cost of training on the laser time series.

Model	CPU Time		Epochs	
	mean	stdev	mean	stdev
MLP-EKF	26.26	2.61	101.96	16.69
MLP-BLM	21.76	1.88	68.11	7.37
RNN-BPTT	37.47	29.73	179.48	108.95
RNN-RTRL	44.65	16.09	235.51	53.03
RNN-EKF	46.33	6.59	113.79	20.17
RBLM-RNN	36.72	2.43	72.84	7.18

4.6.3 Forecasting Electricity Spot Prices

A current challenge to the energy industry is the efficient management of electricity networks. In the final experiment, the predictive ability of the RBLM-RNN on noisy electricity spot price data taken from the Nordpool power exchange is evaluated. The spot prices are measured in Euro/MegaWatt per hour. The studied models were trained on daily average prices from 1/1/2004 to 10/31/2005. A plot of a segment of the in-sample data is shown in Figure 4.6. Two out-of-sample segments were then used to test the models generalization capacity. In the first out-of-sample segment, prices from the next day 11/1/2005 until 11/30/2006 were considered for forecasting. In the second out-of-sample

forecast, prediction began on 1/1/2006 until 1/31/2006, which gives a gap of 2 months between training and forecasting. A small data set of 18 days prior to 1/1/2006 were fed to the RNNs to "prime" the networks states before prediction began (no weight adaptation took place during "priming"). Once the RNNs were primed, prediction began on 1/1/2006.

Table 4.5: Residuals on Electricity Spot Price Fitting and Prediction.

Model	Weights	nMSE(training)	nMSE(testing 1)	nMSE(testing 2)
MLP-EKF	65	0.0680	0.0962	0.1034
MLP-BLM	43	0.05024	0.0899	0.1066
RNN-BPTT	29	0.0497	0.2016	0.2791
RNN-RTRL	29	0.0473	0.1828	0.2376
RNN-EKF	29	0.0458	0.0991	0.1093
RBLM-RNN	19	0.0590	0.0608	0.0831

Table 4.6: Computational cost of training on the electricity spot price time series.

Model	CPU Time		Epochs	
	mean	stdev	mean	stdev
MLP-EKF	20.16	2.27	96.31	15.38
MLP-BLM	17.64	1.26	62.87	6.41
RNN-BPTT	25.57	19.87	198.45	87.63
RNN-RTRL	32.2	11.72	231.13	58.75
RNN-EKF	38.88	5.84	107.96	19.61
RBLM-RNN	26.89	1.83	61.75	6.23

All recurrent networks: RBLM-RNN, RNN-BPTT, RNN-RTRL and RNN-EKF were implemented for this task with 4 hidden neurons, 1 input and 1 output neuron. The covariance matrices were initialized with elements 10^3. The initial hyperparameters for the RBLM-RNN and MLP-BLM were: $\alpha_i = 10^{-4}$ and $\beta = 1.0$. The MLP-EKF was initialized with 4 hidden neurons, but used input vector with time lag 14 from the series[2], i.e. there are 65 weights.

The results from the in-sample fitting and out-of-sample forecasts are given in Table 4.5 and the computational costs of training are given in Table 4.6. The plots of the out-of-sample forecasts and errors are shown in Figures 4.7 and 4.8. The RBLM-RNN

[2]After multiple trials, it was found that it was necessary to include two weeks of past data in the input window to achieve accurate predictions. This is possibly due to a weekly seasonality effect in the electricity series.

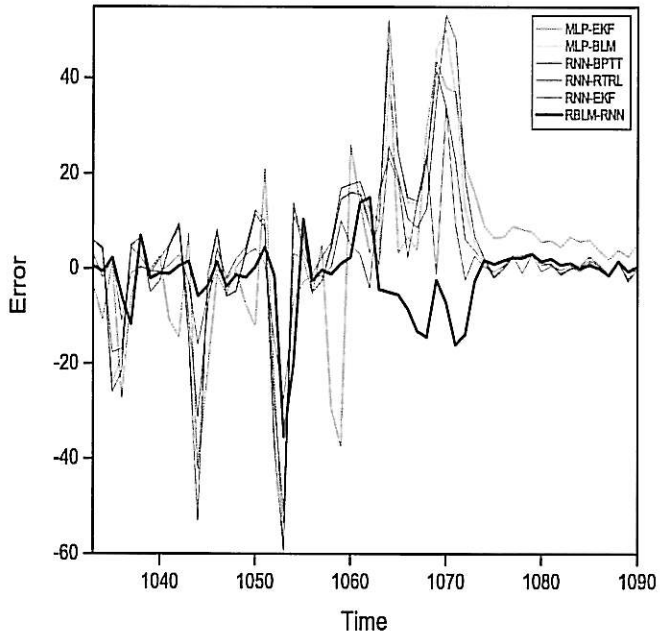

Figure 4.5: Plots of the errors on the Laser time series for all models considered in the study. The errors are for the one-step-ahead out-of-sample forecast.

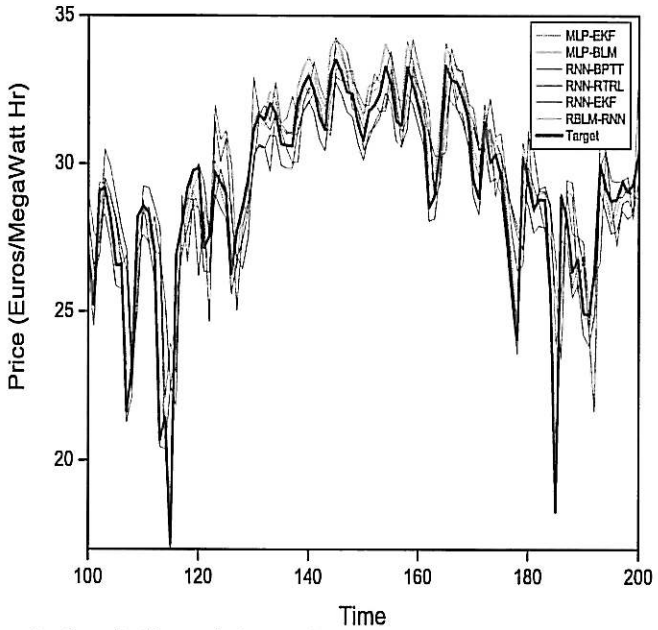

Figure 4.6: In-sample fit of all models on the a segment of the Nordpool electricity price series.

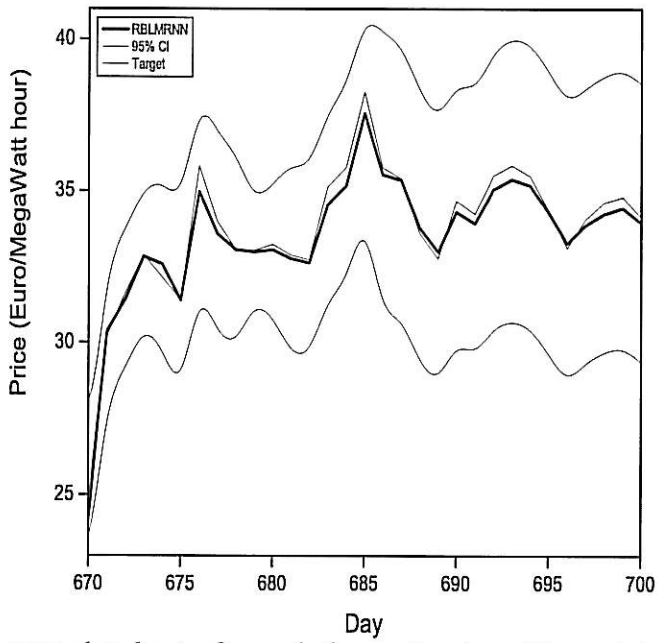

Figure 4.7: A one-step-ahead out-of-sample forecast and confidence intervals of the Nord-pool electricity price series. The plot here shows the RBLM-RNN performance in forecasting a segement of the unseen data.

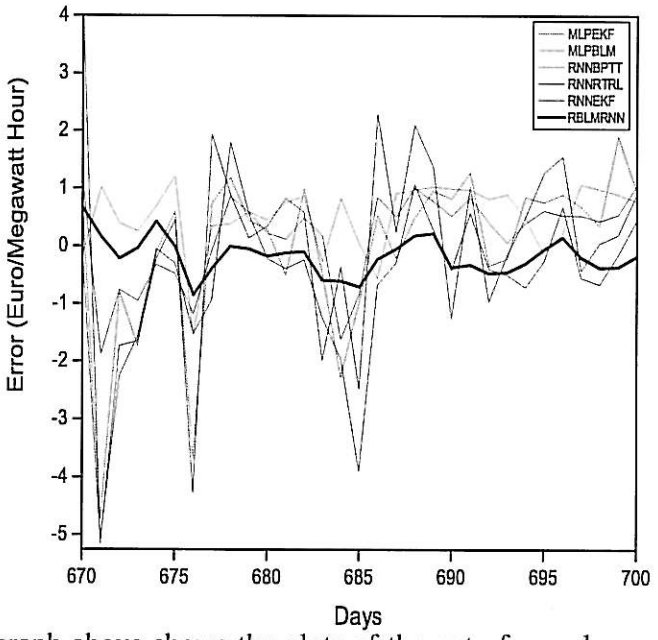

Figure 4.8: The graph above shows the plots of the out-of-sample errors for all models considered in this study evaluated on a segment of the unseen data.

does not seem to fit the noise too severely in this series as it does not show the lowest training error, however the RBLM-RNN performs best on forecasting. It seems that the Bayesian modeling of the noise characteristics as well as the Bayesian tuning of the regularization hyperparameters may have reduced the overfitting in the training phase. Without Bayesian regularization, the MLP-EKF, RNN-RTRL and RNN-EKF seem to overfit the training series because they achieve very low training error, but very high prediction error. The MLP-BLM is second best after the RBLM-RNN. The model with the lowest average (mean) number of epochs to convergence in this experiment was the RBLM-RNN. Again the mean training time for the RBLM-RNN was lower than that of the RNN-EKF and RNN-RTRL training algorithms.

4.7 Conclusion

The development of Bayesian regularized neural models has led to three key contributions. First it was found that Bayesian regularization for recurrent neural networks is an effective means to control model complexity by automatically selecting model hyperparameters without the need for cross-validation or boot-strapping. Second, through the use of automatic relevance determination by providing one regularization hyper-parameter for each weight, sparse recurrent neural models have been achieved. Lastly, via the extended Kalman filter, the direct computation of the hessian has been avoided for the Bayesian regularization of recurrent neural models. Experimental findings on standard benchmark time series, Mackey-Glass and sun spots, show significant improvement in prediction accuracy and reduction in network size over the control group (MLP-EKF, MLP-BLM, RNN-RTRL, RNN-EKF).

Overfitting has stood in the way of well generalizing recurrent neural models for time series prediction. This chapter presented an approach to addressing this problem through a probabilistic recursive training algorithm for dynamic recurrent neural models which features a sequential approach to regularization in the Bayesian framework. This approach is unique from previous attempts to regularize RNNs in the sense that the regularization is carried out sequentially over the training data. The advantage of this is that over large data sets, ill conditioning of the Hessian matrix is avoided. This chapter has demonstrated that the sequential Bayesian regularization of RNNs has led to improvements in prediction accuracy when processing time series. The presented results are encouraging as they

show that the RBLM-RNN has the capacity to outperform feed-forward neural networks and other recurrent networks on dynamic modeling tasks in terms of forecast errors.

Chapter 5

Sequential Bayesian RNN Filtering

This chapter develops sequential RNN training algorithms following the sequential Bayesian framework. The Bayesian paradigm is widespread in recursive estimation [12, 82, 109, 175] and some strong proponents of Bayesian theory claim that the Bayesian paradigm provides a unifying theory of recursive estimation [82]. The Bayesian framework was first introduced into the field of recursive estimation by Ho and Lee in 1964 [82]. Since then the Bayesian view of recursive estimation has grown immensely and is well accepted in the literature [11, 56, 108, 175, 198]. In the Bayesian view, the desired quantity of interest is the posterior conditional density of the system state given the measurements. As new measurements arrive, knowledge about the posterior is recursively updated.

The sequential Bayesian estimation framework have been found to be highly effective for training RNNs [26, 78, 103, 126, 155, 165, 171, 173]. This is due to the regularized second-order optimization properties of the sequential filtering framework. During the optimization, the filter minimizes a penalized cost function, which implicitly applies regularization during training [9]. An added benefit of sequential Bayesian filters is the adaptive learning rates individually tuned for each weight. These learning rates are computed in the Kalman gain [189]. Although these methods are second-order optimizers, they compute suboptimal posterior estimates of the RNN weights which turn out to be beneficial to RNN training as they discourage overfitting of the training data [9]. Finally, the filters are sequential so they are less likely to get trapped in local minima [9].

There has been extensive research done in training RNNs with sequential Bayesian filters [9, 23, 78, 170, 171]. The first work was proposed by Puskurios and Feldkamp [172]. This work showed that the Kalman filter trained RNN can significantly outperform gradi-

ent based methods. Other work by Haykin [78] explored how the filters can learn complex chaotic dynamics and was suitable for reconstruction of nonlinear dynamical systems. Further work was done on modeling time varying systems [87]. Various filters in the Kalman filtering class have been proposed, including the Uncented Kalman Filter [27], and the Cubature Kalman filter [9]. It was shown that each filter has its own characteristics and improvements can be made from using more "advanced" filtering methods than the standard extended Kalman filter. This is generally due to the various ways nonlinearities are handled in each filter. One of the main limitations to the original Kalman filter is the problem of divergence during situations of high nonlinearity [78, 156, 157, 172]. These problems can usually be traced to improper linearization which biases the posterior estimates of the RNN weights, or the use of incorrect noise covariances (noise covariance estimation is addressed in Chapter 6).

Improvements have been reported through the use of filters that avoid first-order approximation of the RNN [78, 156]. However, solutions that have been proposed have a significant drawback, the computational cost of training. This chapter explores various alternatives to the leading filtering algorithms for training RNNs. It proposes the SEEK filter as a method to reduce divergence [166]. However, this method relies on first-order linearization of the RNN, which carries a large computational burden. To remedy this problem, a training method based on Monte Carlo sampling is proposed, which avoids the need for computation of the derivatives all together [134, 136]. Although it may seem that Monte Carlo methods may negate this savings from avoiding derivative computation, it turns out that the filter maintains a population of samples whose statistics alleviate the need for the direct computation of the covariance of the errors, which results in a drastic reduction in computational complexity. This method of computationally efficient online Monte Carlo RNN training was pioneered by the author in [134, 136]. The achievements are a reduction in computational complexity for online training of RNNs and an improvement in out-of-sample forecasts.

These methods are evaluated in terms of how well the filters are able to produce well generalizing models (i.e. out-of-sample forecasts), as well as a reduction in the computational burden of RNN weight estimation. The next section provides an overview of the theory of sequential Bayesian filtering, which is followed by a derivation of the closed form solution to the discrete time linear optimal filtering problem, which is known as, the Kalman Filter. Then, novel non-linear sequential Bayesian filtering algorithms for training RNNs are presented.

5.1 Sequential Bayesian Inference

The objective of the sequential Bayesian inference in RNN training is the computation of the posterior probability of the RNN weights, which is a combination of information about the RNN weights before new measurements are obtained, and information from the newest observation [12, 93, 175]. Computation of the posterior corresponds to inverting a noisy sequence of observations $\{d_1, \ldots, d_t\}$ to infer the system state, which for RNN training corresponds to estimation of the RNN weights $\{\mathbf{w}_1, \ldots, \mathbf{w}_t\}$.

A naive solution to the inverse problem of finding the weights given the observations would be to plug the distribution of the weights $p(\mathbf{w}_{0:t}) = p(\mathbf{w}_0, \ldots, \mathbf{w}_t)$ and the joint distribution $p(\boldsymbol{d}_{0:t}|\mathbf{w}_{0:t})$ into Bayes rule

$$\begin{aligned} p(\mathbf{w}_0, \ldots, \mathbf{w}_t|d_1, \ldots, d_t) &= \frac{p(d_1, \ldots, d_t|\mathbf{w}_0, \ldots, \mathbf{w}_t)p(\mathbf{w}_0, \ldots, \mathbf{w}_t)}{p(d_1, \ldots, d_t)} \\ &\propto p(d_1, \ldots, d_t|\mathbf{w}_0, \ldots, \mathbf{w}_t)p(\mathbf{w}_0, \ldots, \mathbf{w}_t) \end{aligned} \tag{5.1}$$

which results in the posterior density of the weights given the observations. The density of the weights $p(\mathbf{w}_{0:t})$, known as the prior density, provides information about the weights before the observations $\boldsymbol{d}_{0:t}$ are obtained. The information from the observations $\boldsymbol{d}_{0:t}$ are implicitly contained in the likelihood density $p(\boldsymbol{d}_{0:t}|\mathbf{w}_{0:t})$. The combination of these two densities via Bayes rule leads to the posterior $p(\mathbf{w}_{0:t}|\boldsymbol{d}_{0:t})$. In theory this formulation will work, but in practice, for online applications, the computations will grow unboundedly.

Rather than computing the full joint distribution of the entire history of the RNN weights (i.e. Equation 5.1), given the data, it is much more efficient to compute what is known as the filtering distribution [93]

$$p(\mathbf{w}_t|\boldsymbol{d}_{1:t}) \tag{5.2}$$

which uses constant computations per time step, allowing for realtime online filtering. To do so, the discrete-time state space framework is generally employed [56]:

$$\begin{aligned} \mathbf{w}_t &\sim p(\mathbf{w}_t|\mathbf{w}_{t-1}) \\ d_t &\sim p(d_t|\mathbf{w}_t) \end{aligned} \tag{5.3}$$

where $\mathbf{w}_t \in \mathbb{R}^m$ is the weight vector of the RNN, which corresponds to the state of the

underlying system at time step t, and the observations are $d_t \in \mathbb{R}^1$. The RNN weight transition dynamics is assumed to be governed by a first-order stochastic process represented by the probability density $p(\mathbf{w}_t|\mathbf{w}_{t-1})$. The measurement model $p(d_t|\mathbf{w}_t)$ is a functional relationship between the state and what is observed. This relationship is stochastic in the sense that there could be a distribution of possible values for any given state.

This state space framework relies on the Markov properties of the state process which assumes that the sequence of RNN weights $\{\mathbf{w}_t : t = 1, 2, \ldots\}$ forms a Markov chain. This implies that the current RNN weight vector $\mathbf{w}_t$ and any other weight vector in the future (i.e. $\mathbf{w}_{t+1}, \mathbf{w}_{t+2}, \ldots$) given the past weight vector $\mathbf{w}_{t-1}$ is independent from anything prior to that:

$$p(\mathbf{w}_t|\mathbf{w}_{1:t-1}, \boldsymbol{d}_{1:t-1}) = p(\mathbf{w}_t|\mathbf{w}_{t-1}) \tag{5.4}$$

Furthermore, the probability distribution of the RNN weight vector $\mathbf{w}_t$ given the previous weight vector $\mathbf{w}_{t-1}$, i.e. $p(\mathbf{w}_t|\mathbf{w}_{t-1})$ is conditionally independent from all other weight vectors in the past $p(\mathbf{w}_t|\mathbf{w}_{1:t-1}) = p(\mathbf{w}_t|\mathbf{w}_{t-1})$. The probability density function (PDF) of the RNN weight vector, $p(\mathbf{w}_{0:t})$, can then be written as

$$p(\mathbf{w}_0, \ldots, \mathbf{w}_t) = p(\mathbf{w}_0) \prod_{i=1}^{t} p(\mathbf{w}_i|\mathbf{w}_{i-1}) \tag{5.5}$$

It is also assumed that the measurements d_t are conditionally independent of the previous weight vectors $\mathbf{w}_{1:t}$ and also of the previous observations $\boldsymbol{d}_{1:t-1}$

$$p(d_t|\mathbf{w}_{1:t}, \boldsymbol{d}_{1:t-1}) = p(d_t|\mathbf{w}_t) \tag{5.6}$$

hence, the joint distribution of the observations given the RNN weights is given by

$$\begin{aligned} p(d_1, \ldots, d_t|\mathbf{w}_0, \ldots, \mathbf{w}_t) &= p(d_0|\mathbf{w}_0) \prod_{i=1}^{t} p(d_i|\mathbf{w}_i) \\ &= p(d_0) \prod_{i=1}^{t} p(d_i|\mathbf{w}_i) \end{aligned} \tag{5.7}$$

The recursive filtering framework consists of applying Bayes rule and marginaliza-

tion [12, 175]. Starting with the initial prior defined as

$$\begin{aligned} p(\mathbf{w}_0|d_0) &= \frac{p(d_0|\mathbf{w}_0)p(\mathbf{w}_0|d_0)}{\int p(d_0|\mathbf{w}_0)p(\mathbf{w}_0|d_0)\mathrm{d}\mathbf{w}_0} \\ &= \frac{p(d_0)p(\mathbf{w}_0)}{p(d_0)} \\ &= p(\mathbf{w}_0) \end{aligned} \tag{5.8}$$

recursive calculations are possible given the transition probability function, $p(\mathbf{w}_t|\mathbf{w}_{t-1})$, the likelihood density which contains previous knowledge of the system $p(\mathbf{w}_t|\boldsymbol{d}_{1:t-1})$, current information of the system $p(d_t|\mathbf{w}_t)$, and the initial prior $p(\mathbf{w}_0)$. With these quantities, the joint density of the RNN weights given all measurements from zero to time t can be computed via Bayes rule

$$\begin{aligned} p(\mathbf{w}_t|\boldsymbol{d}_{1:t}) &= \frac{p(d_t|\boldsymbol{d}_{1:t-1}, \mathbf{w}_t)p(\mathbf{w}_t|\boldsymbol{d}_{1:t-1})}{p(d_t|\boldsymbol{d}_{1:t-1})} \\ &= \frac{p(d_t|\mathbf{w}_t)p(\mathbf{w}_t|\boldsymbol{d}_{1:t-1})}{p(d_t|\boldsymbol{d}_{1:t-1})} \end{aligned} \tag{5.9}$$

where the measurement update is computed by

$$p(d_t|\boldsymbol{d}_{1:t-1}) = \int p(d_t|\mathbf{w}_t)p(\mathbf{w}_t|d_{1:t-1})\mathrm{d}\mathbf{w}_t \tag{5.10}$$

Based on the Markov properties in Equation 5.4, the effect of each time step is given by

$$\begin{aligned} p(\mathbf{w}_t, \mathbf{w}_{t-1}|\boldsymbol{d}_{1:t-1}) &= p(\mathbf{w}_t|\mathbf{w}_{t-1}, \boldsymbol{d}_{1:t-1})p(\mathbf{w}_{t-1}|\boldsymbol{d}_{1:t-1}) \\ &= p(\mathbf{w}_t|\mathbf{w}_{t-1})p(\mathbf{w}_{t-1}|\boldsymbol{d}_{1:t-1}) \end{aligned} \tag{5.11}$$

Where integration with respect to $\mathbf{w}_{t-1}$ gives the well known Chapman-Kolmogorov time update equation

$$p(\mathbf{w}_t|\boldsymbol{d}_{1:t-1}) = \int p(\mathbf{w}_t|\mathbf{w}_{t-1})p(\mathbf{w}_{t-1}|\boldsymbol{d}_{1:t-1})\mathrm{d}\mathbf{w}_{t-1} \tag{5.12}$$

Equations 5.9, and 5.12 are recursively iterated to compute the posterior RNN weight update $p(\mathbf{w}_t|\boldsymbol{d}_{1:t})$. To see this, consider the posterior as the time step advances; i.e., the

current posterior becomes $p(\mathbf{w}_{t-1}|\boldsymbol{d}_{1:t-1})$. This quantity is advanced forward in time via the Chapman-Kolmogorov equation, Equation 5.12. Then, after receiving a measurement d_t, the likelihood (computed via Equation 5.12) can then be fed back into Equation 5.9 to compute the new posterior. From this recursion, at each time step a pair of distributions $p(\mathbf{w}_t|\boldsymbol{d}_{1:t-1}), p(\mathbf{w}_t|\boldsymbol{d}_{1:t})$ are computed each time a new observation d_t becomes available.

In practice the weight posterior $p(\mathbf{w}_t|\boldsymbol{d}_{1:t})$ is only exactly obtainable for linear Gaussian systems. The most widely known algorithm for computing this exact solution is the Kalman filter which is discussed in the next section. For nonlinear systems the integral Equations 5.10 and 5.12 have few closed form solutions and approximations need to be made. Approximation techniques discussed in this chapter include linearization (discussed in Section 5.3), and Monte Carlo approximation (discussed in Section 5.5).

5.2 The Kalman Filter

The well known Kalman filter [93] provides a solution for linear systems of the kind:

$$d_t = \mathfrak{H}\mathbf{w}_t + \nu_t, \qquad \nu_t \sim \mathcal{N}(0, R) \qquad \text{measurement equation,} \tag{5.13}$$

$$\mathbf{w}_t = \mathfrak{F}\mathbf{w}_{t-1} + \omega_t, \qquad \omega_t \sim \mathcal{N}(0, \mathbf{Q}) \qquad \text{process equation} \tag{5.14}$$

where $\mathfrak{H}$ is the measurement matrix that maps the state vector $\mathbf{w}_t$ to the observations, R is the measurement error covariance matrix, $\mathbf{Q}$ is a noise covariance matrix, and $\mathfrak{F}$ represents the system dynamic matrix, which from here on is assumed to be the identity matrix as is the norm for neural network training [78].

In a sequential setting the observable and the state are time variant, where $\boldsymbol{d}_{1:t} = [d_1, \ldots, d_t]$ and $\mathbf{w}_{0:t} = [\mathbf{w}_0, \ldots, \mathbf{w}_t]$. It is assumed that the conditional probability of the weights given the observations is normally distributed with mean $\mathbf{w}_t^f$ and covariance matrix $\mathbf{P}_t^f$

$$p(\mathbf{w}_t|\boldsymbol{d}_{1:t-1}) \sim \mathcal{N}(\mathbf{w}_t^f, \mathbf{P}_t^f) \propto \exp\Big[-\frac{1}{2}(\mathbf{w}_t - \mathbf{w}_t^f)^{\mathrm{T}}(\mathbf{P}_t^f)^{-1}(\mathbf{w}_t - \mathbf{w}_t^f)\Big] \tag{5.15}$$

The Kalman filter can be expressed in two stages. The first stage is known as the forecast stage, in which the forecast state is given as follows:

$$\mathbf{w}_t^f = \mathbf{w}_{t-1}^a \tag{5.16}$$

For the purposes of this thesis, it is assumed that $\mathfrak{F}$ is the identity matrix and thus is dropped from the notation. The forecast error-covariance at the time t is provided next. The correlation coefficient between $\mathbf{w}_t$ and the system noise, ω_t, is assumed to be zero mean and hence $\mathbb{E}\Big[(\mathbf{w}_t^a - \mathbf{w}_t)\omega_t^{\mathrm{T}}\Big] = 0$. Using this assumption and further simplification the analysis-error covariance is given as follows:

$$\begin{aligned}
\mathbf{P}_t^f &= \mathbb{E}\Big[(\mathbf{w}_t^f - \mathbf{w}_t)(\mathbf{w}_t^f - \mathbf{w}_t)^{\mathrm{T}}\Big] \\
&= \mathbb{E}\Big[(\mathbf{w}_{t-1}^a - \mathbf{w}_{t-1} - \omega)(\mathbf{w}_{t-1}^a - \mathbf{w}_{t-1} - \omega)^{\mathrm{T}}\Big] \\
&= \mathbb{E}\Big[((\mathbf{w}_{t-1}^a - \mathbf{w}_{t-1}) - \omega)((\mathbf{w}_{t-1}^a - \mathbf{w}_{t-1}) - \omega)^{\mathrm{T}}\Big] \\
&= \mathbb{E}\Big[(\mathbf{w}_{t-1}^a - \mathbf{w}_{t-1})(\mathbf{w}_{t-1}^a - \mathbf{w}_{t-1})^{\mathrm{T}}\Big] + \mathbf{Q} \\
&= \mathbf{P}_{t-1}^a + \mathbf{Q}
\end{aligned} \tag{5.17}$$

The second stage is known as the analysis stage, where the analysis state update, $\mathbf{w}_t^a$, and the analysis error-covariance, $\mathbf{P}_t^a$ at the next times step, t, are derived. It is also assumed the observations follow a normal distribution with a mean d_t and covariance R.

$$p(d_t|\mathbf{w}_t) \sim \mathcal{N}(d_t, R) \propto \exp\Big[-\frac{1}{2}(\mathfrak{H}\mathbf{w}_t - d_t)^{\mathrm{T}} R^{-1}(\mathfrak{H}\mathbf{w}_t - d_t)\Big] \tag{5.18}$$

where $\mathfrak{H}$ is the linear operator linking the state of the system to the observations.

Bayes' rule specified in Equation 5.9 is then applied the to Equation 5.15 and Equation 5.18 resulting in the posterior distribution

$$p(\mathbf{w}_t|d_{1:t}) = \exp\Big[-\frac{1}{2}(\mathbf{w}_t - \mathbf{w}_t^f)^{\mathrm{T}}(\mathbf{P}_t^f)^{-1}(\mathbf{w}_t - \mathbf{w}_t^f) - \frac{1}{2}(\mathfrak{H}\mathbf{w}_t - d_t)^{\mathrm{T}} R^{-1}(\mathfrak{H}\mathbf{w}_t - d_t)\Big] \tag{5.19}$$

The maximum of the Equation 5.19 corresponds to the minimum of the negative log of the cost function $C(\mathbf{w}_t)$, where the cost function $C(\mathbf{w}_t)$,is defined as follows

$$C(\mathbf{w}_t) = \frac{1}{2}\Big[(\mathbf{w}_t - \mathbf{w}_t^f)^{\mathrm{T}}(\mathbf{P}_t^f)^{-1}(\mathbf{w}_t - \mathbf{w}_t^f) + (\mathfrak{H}\mathbf{w}_t - d_t)^{\mathrm{T}} R^{-1}(\mathfrak{H}\mathbf{w}_t - d_t)\Big] \tag{5.20}$$

The second term $(\mathfrak{H}\mathbf{w}_t - d_t)^{\mathrm{T}} R^{-1}(\mathfrak{H}\mathbf{w}_t - d_t)$ can be thought of as the sum of squares error measure. The first term $(\mathbf{w}_t - \mathbf{w}_t^f)^{\mathrm{T}}(\mathbf{P}_t^f)^{-1}(\mathbf{w}_t - \mathbf{w}_t^f)$ can be interpreted as a regularization term which provides a distance metric between two RNNs models via the "weighted

Euclidean distance between their corresponding weight vectors" [9]. This regularization property holds for all sequential Bayesian filters discussed in this thesis.

The analysis weight vector, $\mathbf{w}_t^a$, is a solution to $\frac{\partial C(\mathbf{w}_t)}{\partial \mathbf{w}_t}|_{\mathbf{w}_t=\mathbf{w}_t^a} = 0$, which implies that

$$\mathbf{P}_t^{f^{-1}}(\mathbf{w}_t^a - \mathbf{w}_t^f) + \mathfrak{H}^{\mathrm{T}} R^{-1}(\mathfrak{H}\mathbf{w}_t^a - d_t) = 0 \tag{5.21}$$

and

$$\mathbf{w}_t^a = (\mathbf{P}_t^{f^{-1}} + \mathfrak{H}^{\mathrm{T}} R^{-1}\mathfrak{H})^{-1}\Big[(\mathbf{P}_t^f)^{-1}\mathbf{w}_t^f + \mathfrak{H}^{\mathrm{T}} R^{-1} d_t\Big] \tag{5.22}$$

Via the matrix inversion lemma [66] the Kalman filter state update equation and Kalman gain can be expressed as follows [56]

$$\mathbf{K}_t = \mathbf{P}_t^f \mathfrak{H}^{\mathrm{T}}(\mathfrak{H}\mathbf{P}_t^f \mathfrak{H}^{\mathrm{T}} + R)^{-1} \quad \text{Kalman gain} \tag{5.23}$$

$$\mathbf{w}_t^a = \mathbf{w}_t^f + \mathbf{K}_t(d_t - \mathfrak{H}\mathbf{w}_t^f) \quad \text{analysis update} \tag{5.24}$$

The accuracy of the prediction of the analysis state, $\mathbf{w}_t^a$, using Equation 5.24 depends on the accuracy of the forecast-error covariance, $\mathbf{P}_t^f$, and the observation covariance, R. Equation 5.24 shows how the optimal analysis state $\mathbf{w}_t^a$ is estimated. This is simply a correction of the forecast $\mathbf{w}_t^f$. The prediction error weighted by Kalman gain, $\mathbf{K}_t(d_t - \mathfrak{H}\mathbf{w}_t^f)$, is used to correct the forecast state, $\mathbf{w}_t^f$. The analysis-error covariance $\mathbf{P}_t^a$ is derived in the following equations. The definition of the analysis covariance is

$$\mathbf{P}_t^a = \mathbb{E}\Big[(\mathbf{w}_t - \mathbf{w}_t^a)(\mathbf{w}_t - \mathbf{w}_t^a)^{\mathrm{T}}\Big] \tag{5.25}$$

By using Equation 5.13 and Equation 5.22 the following holds [92]

$$\begin{aligned} \mathbf{w}_t - \mathbf{w}_t^a &= \mathbf{w}_t - \mathbf{w}_t^f - \mathbf{K}_t(d_t - \mathfrak{H}\mathbf{w}_t) \\ &= (\mathbf{I} - \mathbf{K}_t\mathfrak{H})(\mathbf{w}_t - \mathbf{w}_t^f) - \mathbf{K}\nu_t \end{aligned} \tag{5.26}$$

It is assumed that the observation and forecast errors are uncorrelated and hence, $\mathbb{E}\Big[(\mathbf{w}_t - \mathbf{w}_t^f)\nu_t^{\mathrm{T}}\Big] = 0$. Using this assumption and further simplification via substitution of Equa-

tion 5.26, the analysis-error covariance, $\mathbf{P}_t^a$ can be written as follows

$$\begin{aligned}
\mathbf{P}_t^a &= \mathbb{E}\Big[(\mathbf{w}_t - \mathbf{w}_t^a)(\mathbf{w}_t - \mathbf{w}_t^a)^{\mathrm{T}}\Big] \\
&= \mathbb{E}\Big[(\mathbf{I} - \mathbf{K}_t\mathfrak{H})(\mathbf{w}_t - \mathbf{w}_t^f)(\mathbf{w}_t - \mathbf{w}_t^f)^{\mathrm{T}}(\mathbf{I} - \mathbf{K}_t\mathfrak{H})^{\mathrm{T}} + \mathbf{K}\nu_t\nu_t^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}\Big] \\
&= (\mathbf{I} - \mathbf{K}_t\mathfrak{H})\mathbf{P}_t^f(\mathbf{I} - \mathbf{K}_t\mathfrak{H}) + \mathbf{K}_t R\mathbf{K}_t^{\mathrm{T}} \\
&= \mathbf{P}_t^f - \mathbf{K}_t\mathfrak{H}\mathbf{P}_t^f - (\mathbf{K}_t\mathfrak{H}\mathbf{P}_t^f)^{\mathrm{T}} + \mathbf{K}_t(\mathfrak{H}\mathbf{P}_t^f\mathfrak{H}^{\mathrm{T}} + R)\mathbf{K}_t^{\mathrm{T}}
\end{aligned} \tag{5.27}$$

Simplification through the use definition of the Kalman gain $\mathbf{K}_t$ from Equation 5.23, the analysis covariance can be written as

$$\begin{aligned}
\mathbf{P}_t^a &= \mathbf{P}_t^f - \mathbf{P}_t^f\mathfrak{H}^{\mathrm{T}}(\mathfrak{H}\mathbf{P}_t^f\mathfrak{H}^{\mathrm{T}} + R)^{-1}\mathfrak{H}\mathbf{P}_t^f \\
&= (\mathbf{I} - \mathbf{K}_t\mathfrak{H})\mathbf{P}_t^f
\end{aligned} \tag{5.28}$$

which is the well known update formula for the analysis covariance.

5.3 EKF Training of Recurrent Neural Networks

The Kalman filter is a popular recursive Bayesian state estimation algorithm which provides the optimal estimate in a minimum variance and maximum likelihood sense, assuming linearity in the process and measurement equations and Gaussian noise terms. In situations with nonlinearity in the state space equations, the EKF [93] has been proposed as a Kalman filter extension to nonlinear systems which provides a suboptimal estimate. For EKF training of RNNs (or more generally for Sequential Bayesian training of RNNs) the RNN is expressed compactly by the functional form

$$y_t = h(\mathbf{w}_t, \mathbf{u}_t) \tag{5.29}$$

where $\mathbf{w}_t \in \mathbb{R}^m$ is the RNN weight vector and $\mathbf{u}_t \in \mathbb{R}^L$ is the input vector to the RNN. The outputs of the RNN a scalar quantity y_t. This formulation of the RNN facilitates treatment of the RNN in state space form for sequential Bayesian estimation of the RNN weights.

To achieve sequential Bayesian training, the RNN weights $\mathbf{w}_t$ are modeled as a stochas-

tic process which allows for modeling with uncertainty in their estimate. From this assumption it is then possible to write the state space model of the RNN training problem as follows

$$d_t = h(\mathbf{w}_t, \mathbf{u}_t) + \nu_t, \qquad \nu_t \sim \mathcal{N}(0, R) \qquad \text{measurement equation,} \tag{5.30}$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \omega_t, \qquad \omega_t \sim \mathcal{N}(0, \mathbf{Q}) \qquad \text{processs equation} \tag{5.31}$$

The weights $\mathbf{w}_t$ follow a random walk, where the stochasticity originates from the random variable ω_t which is normally distributed with covariance matrix $\mathbf{Q}$. The RNN $h(\mathbf{w}_t, \mathbf{u}_t)$ represents the nonlinear measurement function with output y_t that maps the process equation (Equation 5.30) to the measurement. The desired output d_t is taken to be the measurement with zero-mean Gaussian noise ν_t with variance R. The sequential Bayesian framework uses the state space model defined in Equations 5.30, 5.31 to infer the weights $\mathbf{w}_t$ given measurements d_t.

The Kalman filter can not be applied to the nonlinear state space model of Equations 5.30, 5.31 due to the nonlinearity of the RNN, i.e., the measurement function is now $h(\mathbf{w}_t, \mathbf{u}_t)$ rather than the matrix $\mathfrak{H}$. A simple solution to this problem is to use linearization techniques to make the nonlinear state space model approximately linear. The process of sequential linearization of $h(\mathbf{w}_t, \mathbf{u}_t)$ leads to the well known extended Kalman filter, which has been found to provide successful results via RTRL [23].

After linearization of the measurement operator (the RNN) around the most recent weight estimate, the Kalman filtering equations can be applied to update the RNN weights in two phases, the forecast stage and the analysis stage. In the forecast stage the weights and covariance are propagated forward in time. In the analysis stage the RNN weights and error covariance are updated in light of the newly arrived measurement.

The forecast stage: It is again assumed the observation and forecast errors are uncorrelated and hence, $\mathbb{E}[(\mathbf{w}_t^a - \mathbf{w}_t)\omega_t^{\mathrm{T}}] = 0$. By applying Equation (5.16), the forecast error-covariance is derived in the same way as in Equation (5.17)

Analysis stage: In the same way as in the case of Kalman filter, the analysis state and error-covariance are given as follows:

The analysis state

$$\begin{aligned} \mathbf{w}_t^a &= \mathbf{w}_t^f + \mathbf{k}_t(d_t - h(\mathbf{w}_t^f, \mathbf{u}_t)) \quad \text{analysis update} \qquad (5.32) \\ \mathbf{k}_t &= \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} (\mathbf{j}_t \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} + R)^{-1} \quad \text{Kalaman gain} \qquad (5.33) \end{aligned}$$

where $\mathbf{j}_t$ is the gradient of the measurement (which is defined in Equation 3.31), $h(\mathbf{w}_t^f, \mathbf{u}_t)$, around $\mathbf{w}_t^f$. The Kalman gain vector is $\mathbf{k}_t$ and the analysis error-covariance is given by

$$\begin{aligned} \mathbf{P}_t^a &= \mathbf{P}_t^f - \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} (\mathbf{j}_t \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} + R)^{-1} \mathbf{j}_t \mathbf{P}_t^f \\ &= (\mathbf{I} - \mathbf{k}_t \mathbf{j}_t) \mathbf{P}_t^f \end{aligned} \qquad (5.34)$$

This process is repeated each time a new data point is received. The EKF has become the standard algorithm for online training of RNNs [23, 78, 171, 172]. However, the algorithm relies on linearized estimates of the nonlinearities computed via differentiation [78]. Biases and/or errors from the linearization process have the potential to deleteriously affect covariance calculations leading to a degradation of filter performance [12]. This may cause the filter to diverge during situations of high nonlinearity, leading to biased estimates of the RNN weights [187].

5.4 Singular Evolutive Extended Kalman filter

Although the Kalman filter has been effective in sequential estimation, divergence can become a problem in situations of high nonlinearity [157]. The Singular Evolutive Extended Kalman filter (SEEK) [166] has been shown to outperform the EKF in various estimation tasks. The SEEK filter assumes that the error-covariance matrix, $\mathbf{P}$, is symmetric with real entries. The SEEK algorithm decomposes $\mathbf{P}$ into a real orthogonal, matrix $\mathbf{L}$ such that $\mathbf{U} = \mathbf{L}^{\mathrm{T}}\mathbf{P}\mathbf{L}$ is diagonal. As $\mathbf{L}$ is orthogonal the matrix $\mathbf{P}$ can be rewritten as follows:

$$\mathbf{P} = \mathbf{L}\mathbf{U}\mathbf{L}^{\mathrm{T}} \qquad (5.35)$$

At the beginning of training the initial matrix $\mathbf{U}_0$ is defined as

$$\mathbf{U}_0 = [\mathbf{L}\mathbf{j}_0^{\mathrm{T}} R^{-1} \mathbf{j}_0 \mathbf{L}]^{-1} \qquad (5.36)$$

in which the decomposition of the covariance $\mathbf{P}$ leads to the $m \times m$ diagonal matrix $\mathbf{U}$ which contains the eigenvalues.

The initialization is based on the first observation d_0, in which the initial weight vector is computed via:

$$\mathbf{w}_0^a = \mathbf{w}_0 + \mathbf{L}_0\mathbf{U}_0\mathbf{L}_0^T\mathbf{j}_0 R_0^{-1}(d_0 - h(\mathbf{w}_0, \mathbf{u}_0)) \tag{5.37}$$

From this point on, the EKF analysis equations are slightly modified to reach the SEEK filtering algorithm. The main objective of the filter for training RNNs is to compute the quantities $\mathbf{w}^a$ given $\mathbf{w}^f$ and the observations. The gain $\mathbf{k}_t$ is the vector that scales the update of the analysis weights against the error committed by the RNN.

As in the Kalman filter, the new observation d_t at time t is used to correct the forecast according to:

$$\mathbf{w}_t^a = \mathbf{w}_t^f + \mathbf{k}_t(d_t - h(\mathbf{w}_t^f, \mathbf{u}_t)) \tag{5.38}$$

However, the gain $\mathbf{k}_t$, is expressed much differently from the original Kalman filter, as

$$\mathbf{k}_t = \mathbf{L}_t\mathbf{U}_t\mathbf{L}_t^{\mathrm{T}}\mathbf{j}_t^{\mathrm{T}}R^{-1} \tag{5.39}$$

where $\mathbf{L}_t = \mathbf{L}_{t-1}$ and $\mathbf{U}_t$ is defined as follows:

$$\mathbf{U}_t^{-1} = \rho[\mathbf{U}_{t-1}^{-1} + (\mathbf{L}_t^{\mathrm{T}}\mathbf{L}_t)^{-1}\mathbf{L}_t^{\mathrm{T}}\mathbf{Q}\mathbf{L}_t(\mathbf{L}_t^{\mathrm{T}}\mathbf{L}_t)^{-1}]^{-1} + \mathbf{j}_t^{\mathrm{T}}\mathbf{L}_t^{\mathrm{T}}\mathbf{L}^{-1}\mathbf{j}_t\mathbf{L}_t. \tag{5.40}$$

where $\mathbf{j}_t$ is the RTRL gradient defined in Equation 3.31 and $0 < \rho \leq 1$ is the forgetting factor.

The SEEK filter has been shown to be effective for state estimation of complex non-linear high dimensional models in oceanography [166], as well as improved performance over the EKF for training RNNs [157]. A useful feature of the model in the climate sciences community is that the filter can be run in reduced rank, which can save a significant amount of computations during state estimation. However, the reduced rank estimation was found to degrade RNN prediction performance during training, so the filter has only been run at full rank of the covariance matrix.

5.5 EnKF Training of RNNs

The Ensemble Kalman Filter (EnKF) [47] has made a great impact for data assimilation of highly nonlinear large scale models [48, 97]. This section focuses on the ensemble filtering approach for estimation of RNN weights for online time series modeling. The ensemble Kalman filter (EnKF) was first proposed by Evensen [46] for estimation in dynamical systems. The EnKF takes a unique approach to filtering by using an ensemble forecast to approximate the error covariance matrix $\mathbf{P}^f$, and the estimate of the state [47]. The distinguishing feature of the EnKF is that it avoids the computation of the derivatives of the observation function $h(\mathbf{w}_t, \mathbf{u}_t)$ altogether. The EnKF approximates the integrals of the distributions of interest by discrete summation, thus computing efficiently their moments, which results in a reduction of computational complexity. The proposed algorithm has superior convergence properties to gradient descent learning and EKF filtering [136].

The main difference between the EKF and the EnKF is that the EKF approximates the nonlinearity of the underlying system through linearization via differentiation, whereas the EnKF represents the system nonlinearity through integration of a statistical sample of weight estimates. The sample, or ensemble, is used to compute the error covariance which leads to computation of a single Kalman gain. The Kalman gain is then used to update the statistical sample (ensemble) in the analysis step. There is a second benefit of using an ensemble; namely reduced computational complexity. The EnKF has the most favorable computational complexity of all online training methods for RNNs [136]. A major savings comes from the absence of a separate covariance matrix to be evolved and updated. Another major savings comes from the use of the full nonlinearity of the model which circumvents the need to compute derivatives of the RNN.

The algorithm starts by randomly generating an ensemble of plausible weight vectors within a predefined interval. The initial ensemble contains a number of independent weight samples from the distribution of the state which are all equally important:

$$\mathbf{W}_t^f = [\mathbf{w}_{t,1}^f, \mathbf{w}_{t,2}^f, \ldots, \mathbf{w}_{t,n}^f] \tag{5.41}$$

where $\mathbf{W}_t^f \in \mathbb{R}^{(m \times n)}$ in which there are n samples in the ensemble, and m number of RNN weights. The EnKF does not resample the ensemble, rather it only updates the members of the ensemble (weights) after the arrival of new information.

Each ensemble member is accessible via the operator $\mathbf{W}_t^f(i) = \mathbf{w}_{t,i}^f$. The ensemble forecast is defined as $y_{t,i} = h(\mathbf{W}_t^f(i), \mathbf{u}_t)$ where the forecast vector is is given by

$$\mathbf{y}_t = [y_{t,1}, y_{t,2}, \ldots, y_{t,n}] \tag{5.42}$$

Statistics of the ensemble can be computed via simple operations. The most useful are the mean, which is computed by

$$\bar{\mathbf{w}}_t^f = \frac{1}{n} \sum_{i=1}^{n} \mathbf{w}_{t,i}^f \tag{5.43}$$

To simplify notation, it will be convenient to introduce the ensemble perturbation matrix

$$\grave{\mathbf{W}}_t^f = \frac{1}{\sqrt{m-1}} \left(\mathbf{w}_{t,1}^f - \bar{\mathbf{w}}_t^f, \mathbf{w}_{t,2}^f - \bar{\mathbf{w}}_t^f, \ldots, \mathbf{w}_{t,n}^f - \bar{\mathbf{w}}_t^f \right) \tag{5.44}$$

The error covariance matrix of the ensemble is then computed as

$$\mathbf{P}_t^f = \grave{\mathbf{W}}_t^f (\grave{\mathbf{W}}_t^f)^{\mathrm{T}} \tag{5.45}$$

Restating the definition of the Kalman gain from the EKF:

$$\mathbf{k}_t = \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} (\mathbf{j}_t \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} + R)^{-1} \tag{5.46}$$

Assuming the derivative $\mathbf{j}_t$ was available, then each weight vector in the ensemble could be updated through this equation as in the standard EKF. However, it is costly to compute $\mathbf{j}_t$ (with RTRL). Alternatively, the statistics from the ensemble can be used to compute the Kalman gain, which alleviates the need for linearization with the Jacobian $\mathbf{j}_t$.

This can be achieved by first, computing the mean of the predicted outputs by

$$\overline{y}_t = \frac{1}{n-1} \sum_{i=1}^{n} y_{t,i} = \frac{1}{n-1} \sum_{i=1}^{n} h(\mathbf{w}_{t,i}^f, \mathbf{u}_t) \tag{5.47}$$

and then computing the following covariance matrix and variance by

$$\mathbf{p}_t^{wy} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{w}_{t,i}^b - \overline{\mathbf{w}}_t^b)(y_{t,i} - \overline{y}_t)^{\mathrm{T}} \tag{5.48}$$

$$p_t^{yy} = \frac{1}{n-1}\sum_{i=1}^{n}(y_{t,i} - \overline{y}_t)(y_{t,i} - \overline{y}_t)^{\mathrm{T}} \tag{5.49}$$

This covariance matrix and variance may be considered as approximations of the corresponding quantities in standard filters

$$\mathbf{p}_t^{wy} \approx \mathbf{P}_t\mathbf{j}_t \tag{5.50}$$

and

$$p_t^{yy} \approx \mathbf{j}_t^{\mathrm{T}}\mathbf{P}_t\mathbf{j}_t \tag{5.51}$$

The advantage of computing these quantities through the ensemble is the ability to approximate the Kalman gain without resorting to the use of numerical derivatives [47]. This allows the gain update to be computed by the standard Kalman equation

$$\begin{aligned}\mathbf{k}_t &= \mathbf{P}_t^f\mathbf{j}_t^{\mathrm{T}}(\mathbf{j}_t\mathbf{P}_t^f\mathbf{j}_t^{\mathrm{T}} + R)^{-1} \\ &\approx \mathbf{p}_t^{wy}(R + p_t^{yy})^{-1}\end{aligned} \tag{5.52}$$

where R is the output noise variance.

Then the updating of the individual weights in the ensemble is carried out by adding the newly arrived observation to this ensemble mean via the Kalman gain

$$\overline{\mathbf{w}}_t^a = \overline{\mathbf{w}}_t^f + \mathbf{k}_t(d_t - h(\overline{\mathbf{w}}_t^f, \mathbf{u}_t)) \tag{5.53}$$

where $\overline{\mathbf{w}}_t^a$ denotes the mean of the posterior $p(\mathbf{w}_t|\mathbf{d}_t)$.

5.5.1 Computational Complexity of RNN Ensemble Filtering

The standard approach to online training in RNNs entails computation of the gradient of the error function with respect to the weights via the RTRL algorithm [228]. The RTRL algorithm takes $O(H^4)$ computations (where H is the number of neurons), which

Computation	Cost
$y_{t,i} = h(\mathbf{W}_t^f(i), \mathbf{u}_t)\ \forall i$	$\mathcal{O}(n)$
$\overline{y}_t$	$\mathcal{O}(n)$
$y_{t,i} - \overline{y}_t\ \forall i$	$\mathcal{O}(n)$
p_t^{yy}	$\mathcal{O}(n)$
$(R + p_t^{yy})^{-1}$	$\mathcal{O}(n)$
$\mathbf{k}_t$	$\mathcal{O}(n)$
$\overline{\mathbf{w}}_t^a$	$\mathcal{O}(mn^2)$

Table 5.1: The table provides the number of computations for each step of the algorithm. The computational complexity depends linearly on the number of RNN weights m and quadratically on the number of samples in the ensemble n [125]

is impractical for training large networks[1]. Various methods have been proposed to speed up online learning of RNNs, but usually with a tradeoff.

One of the first approaches to complexity reduction was through a modification of the RTRL algorithm which proposed sub-grouping each output neuron together with an arbitrary number of neurons in the hidden layer [233]. Through this strategy, non-overlapping sub-networks were effectively created, leading to drastic savings in computations in the order of $O(H^4/g^3)$, where g denotes the number of sub-groups. However, as g increases a tradeoff is incurred; less crossover of training information flows between the sub-groups resulting in a significant degradation of the network's capabilities.

Other variants of the algorithm were proposed, such as dynamic sub-grouping [45], but again the algorithm relies on arbitrary reduction in the sensitivity matrix which also degrades the networks modeling power when sub-grouping increases. The most prominent limitation of the sub-grouping strategy is the assumption of multiple output neurons. When applied to single output systems, such as univariate time series modeling, the sub-grouping strategy is not valid for online applications, and a post processing stage is necessary.

A Hybird BPTT/RTRL scheme has been put forward by [188], which reduced the computational complexity to $O(H^3)$. The method relies on segmenting the training set and running back propagation through time on each segment and then using RTRL to forward propagate the gradient history before the start of the next time step. Other methods that similarly reduce computational complexity to $O(H^3)$ have been proposed by [202],

[1]The literature has "traditionally" stated the computational complexity of RNN training algorithms in terms of the size of the Hidden layer. The relationship between the number of weights m and the size of the hidden layer H is $m \approx H^2$

which make use of Greens function. A common thread through all work mentioned is estimation of the RNN weights through differentiation (i.e. previous work focused on finding efficient ways to compute the derivatives of the RNN).

The EnKF trained RNN takes a different approach to complexity reduction in online learning of RNNs through a sequential Bayesian filtering framework and proposes a Markov Chain Monte Carlo approach for derivative free RNN weight estimation. The key difference of the proposed approach is estimation of the RNN weights via integration, rather than differentiation. The EnKF provides an online training solution that can reduce the computational complexity by two orders of magnitude from the original RTRL algorithm without sacrificing the modeling potential of the network [136].

The extended Kalman filter (EKF) [172], and the sigma-point (unscented and central difference) Kalman filters (SPKF) [27, 51, 170] have been proposed as online training algorithms for RNNs. Although the filtering techniques offer superior convergence properties to gradient descent, the computational complexity per time step for the EKF is equivalent to RTRL (and it also depends on RTRL derivatives). The SPKF family has computational complexity of approximately $O(H^6)$, which is much higher than RTRL, and that is why they are not discussed further.

The RNN-EnKF provides a theoretical computational complexity of $O(n^2H^2)$ where n is the ensemble size [125, 150]. The computational computational complexity of each operation is provided in Table 5.1. The parameter n is treated as a constant, in the sense that the parameter n typically varies from $20 - 200$ depending on the problem (for data assimilation in high dimensional problems) [124]. The experimental results show that n can be set to 20 and really does not improve the forecast errors (i.e. accuracy of the model) much beyond 20 (for online learning). It was suggested that the ensemble size may have dependence on the dimension of the attractor of the system that is being modeled [141, 142], and not necessarily a function of the number of free parameters of the model. This relationship is not fully understood, so a graph of the ensemble size vs number of neurons vs model error for one epoch is provided in Figure 5.1. The plot suggests that low model errors are achievable with small ensemble sizes. Lower errors are possible if the filter is run for multiple epochs. As the ensemble size n is treated as a constant[2], the theoretical time complexity of the algorithm with respect to the amount of neurons is approximately $O(H^2)$. At the time of publication of this Thesis, there is no online training algorithm for

[2]Assuming that n is constant, its contribution to the growth in computations is linear, and is dominated by the higher order term H and is thus disregarded.

RNNs with a lower computational complexity. In the following sections, the performance of the proposed algorithm is evaluated.

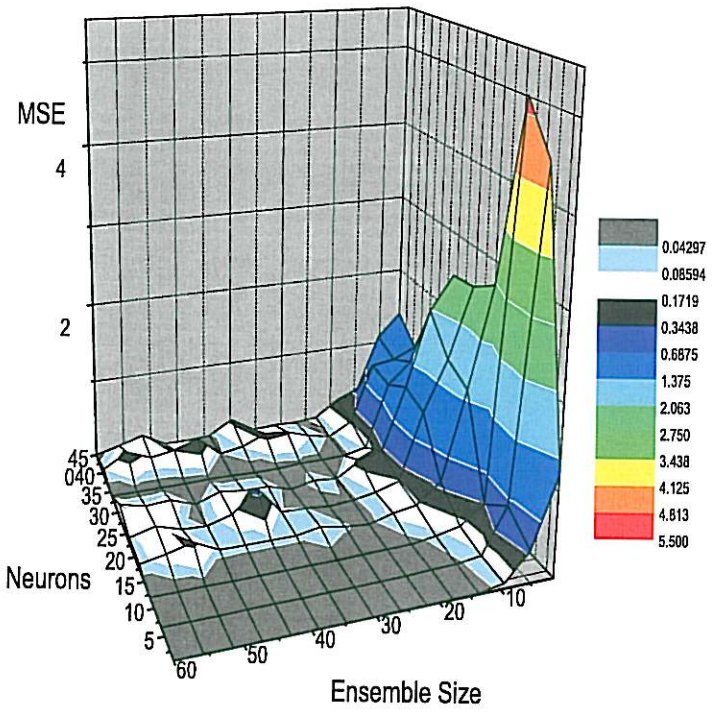

Figure 5.1: Neurons vs Ensemble Size vs MSE obtained after one pass over the data, averaged over 100 runs. Each color band indicates the expected MSE for a range of corresponding network configurations. For a given computationally feasible network configuration the plot suggests to use an ensemble size of 20 in order to achieve reasonable MSE values. The plot also shows that large ensemble sizes do not necessarily lead to low MSE values.

5.6 Experimental Results

The objective of the presented work is to develop computationally efficient methods for sequential training of RNNs for use on time series modeling. To provide a comparison of model performance in terms of computational cost and accuracy, several popular learning algorithms for recurrent networks are compared; the RTRL algorithm [228], and the Extended Kalman Filter for RNNs (RNN-EKF) [23].

Studies into time series modeling and forecasting were carried out using two real world data sets; the first are measurements of the volume of the Great Salt Lake, and the second are hourly averages of the electricity spot prices in the Spanish market. The Salt Lake volume series consists of 3463 average measurements of the lake volume, from which the first 3000 measurements are used for training. The Spanish electricity spot price series consisted of 1152 hourly averages of electricity spot prices corresponding to 48 days. In section 5.5.1, it has already been established that the proposed algorithm has a lower theoretical computational complexity than the existing methods for on-line training of RNNs. However, lower theoretical complexity does not necessarily mean a faster training algorithm in practice. The following experiments assess the speed of the algorithm in terms of CPU time on practical on-line time series forecasting tasks. For these experiments, 7 error thresholds in units of MSE were selected in the range of $[.01, .007]$ and the algorithms were timed to reach these thresholds. The simulations were repeated 50 times at each threshold starting from different initial wight vectors, and the averages were plotted. All simulations were carried out on a Pentium-4 3.6-GHz, 1024-MB RAM workstation. Also the suitability of the algorithm in batch training mode for forecasting out-of-sample (i.e. freezing the weights after training and predicting one step ahead over a predefined future unseen interval) is assessed. Below, the results on the average convergence times, in-sample performance (fitting), and one step ahead forecasting on the out-of-sample segment are reported. Forecast errors are measured by the root mean squared error (rMSE) computed by $rMSE = ((1/(T)) \sum_{t=1}^{T} (d_t - y_t)^2)^{1/2}$, where T is the length of the data sample. In all simulations, the weights of the networks were initialized with random uniformly distributed weights in the range of $[-2, 2]$.

5.6.1 Modeling the Volume of the Salt Lake

The volume of the Salt Lake in Utah, USA, has been used as a proxy for climate variability in the region as the much of the regions precipitation collects into the lake basin. The Salt Lake has a length of about 113km, a width of about 48km, and an average depth of 5m. This large surface area and shallow depth make the lake volume sensitive to changes in the climate. The average volume of the Salt Lake was shown to be chaotic [3], which provides an interesting test for the models.

For this application, all the recurrent networks were initialized with 3 hidden neurons, one input and one output neuron. The RNN-EKF, RNN-SEEK, and the RNN-EnKF filter was initialized with R set to $1.0e^{-5}$. The diagonal elements of $\mathbf{Q}$ in the RNN-EKF, and

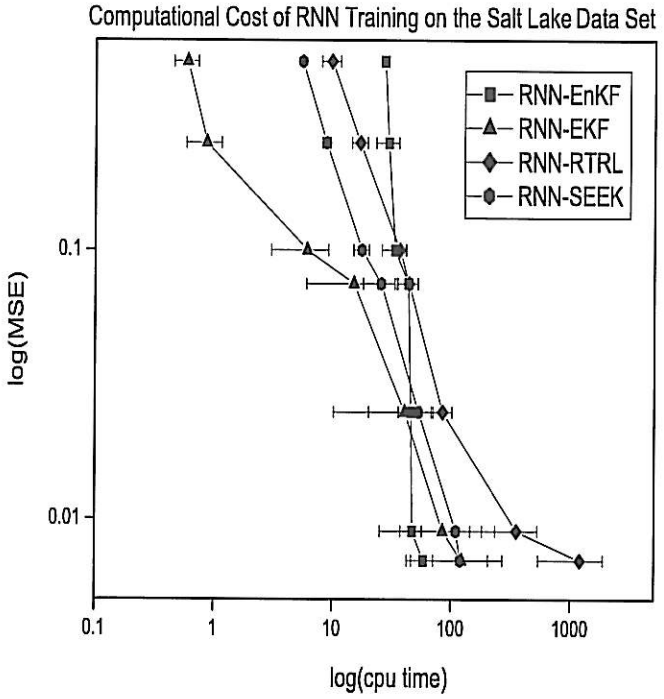

Figure 5.2: Convergence performance of the three online training algorithms applied to the Elman RNN on the Salt Lake data set.

RNN-SEEK were set to $1.0e^{-3}$. The learning rate for the RNN-RTRL was set to .05. The computational cost of training RNNs over the Salt Lake data set is provided in Figure 5.2. The plot clearly illustrates that for relatively high error tolerances, the RNN-EnKF is much slower than the RNN-EKF, RNN-SEEK, and the RNN-RTRL algorithm. However, when a low MSE is required, the RNN-EnKF is the fastest because the other algorithms take significantly more epochs to reach convergence than the RNN-EnKF. The the average mse/epoch for all given algorithms is provided in the second column of Table 5.2. On average the EnKF achieved lower average epochs to reach convergence than the other algorithms which suggests that the RNN-EnKF is more suitable for online learning than the RTRL, SEEK and EKF algorithms. As stated above, a small network size was used for the simulations, due to the practicality of completing the simulations in a timely manner. The small network size favors the RNN-RTRL and RNN-EKF algorithms, however the RNN-EnKF still managed to converge the fastest when high accuracy was demanded.

The numerical results of the out-of-sample forecasts of the studied algorithms on the Salt Lake data set is provided in Figure 5.3. The third and fourth columns of Table 5.2

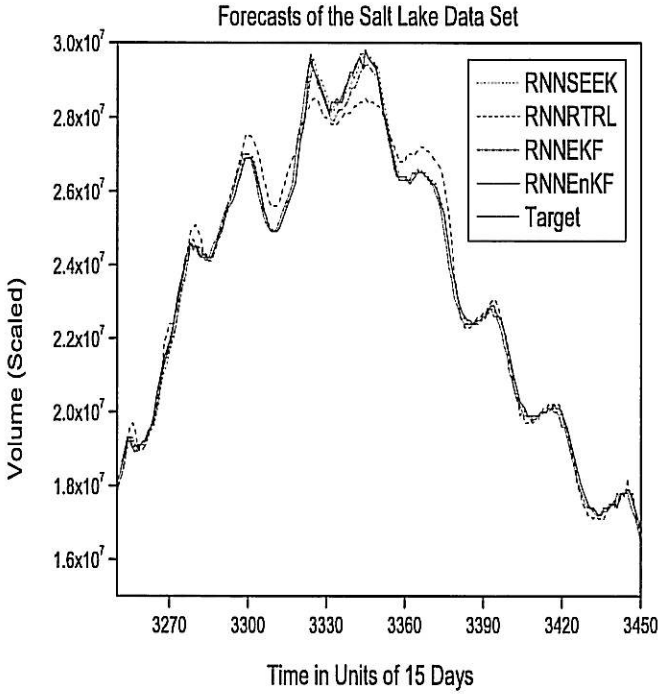

Figure 5.3: Out-of-sample predictions by the RNNs (given by the network output y_t) on the average volume of the Salt Lake (which corresponds to d_t), where each time step is 15 days.

summarize the forecasting results for single step ahead model fitting and prediction on the Salt Lake data set. The in-sample performance of EnKF, EKF, and SEEK algorithms show quite similar ability to fit the in-sample data. However, in out-of-sample forecasting, the EnKF-RNN outperforms the other recurrent network training algorithms including the recurrent network trained by the EKF, SEEK, and the gradient descent RNN algorithm.

Table 5.2: Experimental Results on the Salt Lake Data Set

Model	Mean (Stdev) Epochs	rMSE(*training*)	rMSE(*testing*)
RNN-RTRL	346.8 (173.8)	1.834e5	5.277e5
RNN-EKF	12.4 (6.4)	1.931e5	2.341e5
RNN-SEEK	11.6 (5.9)	1.920e5	2.193e5
RNN-EnKF	3.2 (3.7)	1.917e5	2.035e5

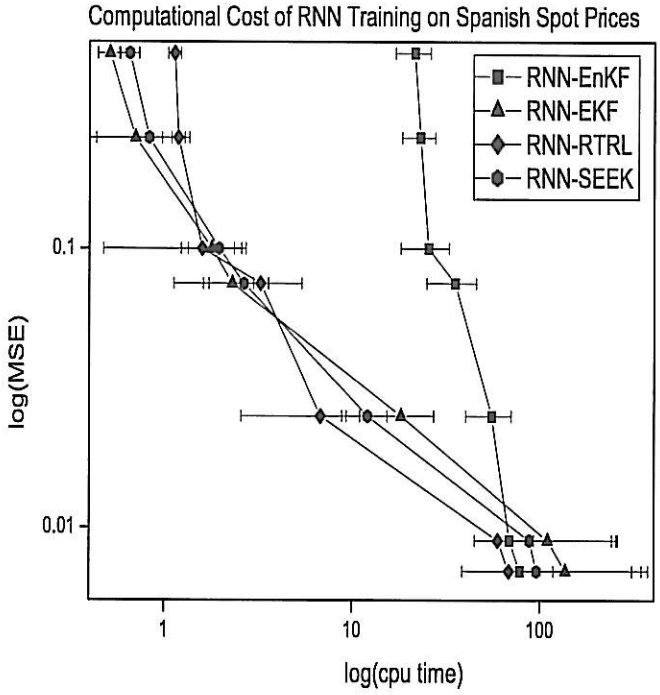

Figure 5.4: Average convergence times of three online training algorithms applied to the Elman RNN evaluated on the Spanish electricity spot price data set.

5.6.2 Modeling Electricity Spot Prices of the Spanish Market

The second study investigates the performance of the proposed model on forecasting hourly prices from the Spanish electricity market. Here the hourly spot prices recorded from June 26, 2008 to August 31, 2008 are considered, from which the first 48 days (June 26, 2004 to August 12, 2004) were used for training and the remaining two weeks (August 13, 2004 to August 26, 2008) were used for out-of-sample forecasting[3].

The RNN-EnKF and the other recurrent networks, RNN-RTRL, RNN-SEEK and RNN-EKF, were initialized with 3 hidden neurons, and one input neuron and one output neuron. The initial diagonal elements of the covariance matrix for $\mathbf{Q}$ of the RNN-EKF, and RNN-SEEK and R for all filters were set to $1.0e^{-3}$ and $1.0e^{-2}$ respectively.

A plot of the average training times to reach pre-specified error targets on the Spanish electricity spot price data set for all algorithms are provided in Figure 5.4. Similar to the previous experimental section, the RNN-EnKF is found much slower than the competing

[3]The data is publicly available from www.omel.es

algorithms when the convergence target is set to relatively high MSE values. As the convergence target is lowered, the RNN-EnKF outperforms the RNN-EKF, RNN-SEEK and similar to the RNN-RTRL. One should take note that only 3 hidden neurons were used in this task, and superior RNN-EnKF performance can be expected as the hidden layer is expanded. As in the previous experiment, the RNN-EnKF is able to achieve convergence in fewer epochs than the other algorithms, and the mean number of epochs for convergence for a MSE of .009 was 2.4 epochs, as shown in the second column of Table 5.3. The RNN-EnKF again shows superior convergence properties and also comparable computational time for training with respect to the RNN-EKF, RNN-SEEK and RNN-RTRL.

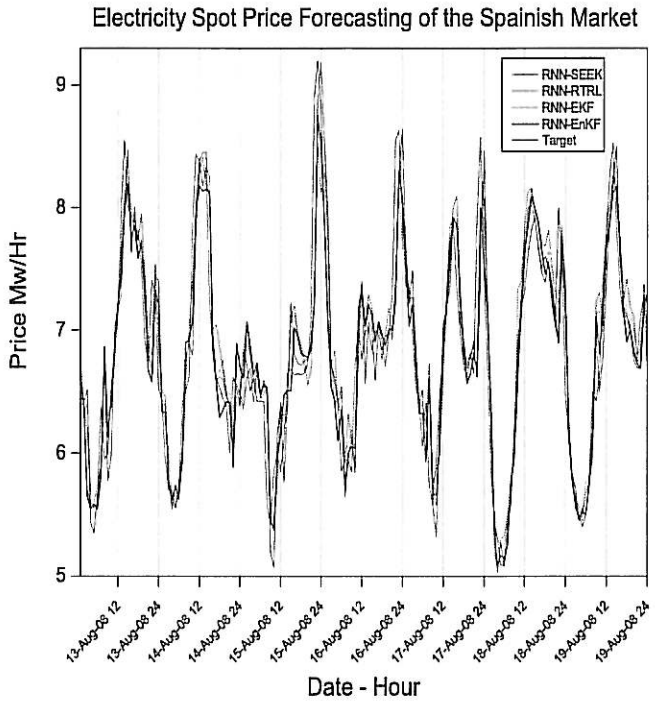

Figure 5.5: One hour ahead forecasts results of the considered RNN models on forecasting hourly average spot prices of the Spanish electricity market (where the forecast is taken from the output of the network y_t and the Target is d_t).

The third and fourth columns of Table 5.3 give the experimental results of training and single-step ahead forecasting on the Spanish spot price series. These results demonstrate that the RNN-EnKF algorithm performs similar to or outperforms the standard RNN training algorithms in terms of training and forecasting. A plot of the out-of-sample predictions are given in Figure 5.5. This plot illustrates improvement in predictive ability

Table 5.3: Numerical Results on the Spanish Electricity Spot Prices

Model	Mean (Stdev) epochs	rMSE(*training*)	rMSE(*testing*)
RNN-RTRL	508.7 (950.1)	0.346	0.385
RNN-EKF	21.6 (1.8)	0.344	0.383
RNN-SEEK	17.8 (1.8)	0.341	0.379
RNN-EnKF	2.4 (1.9)	0.336	0.376

through the use of the EnKF over the use of the EKF for training neural networks. Similar performance was achieved between the RNN-EnKF and RNN-SEEK algorithms.

5.7 Conclusion

Recurrent neural networks are known to be highly powerful forecasting tools, but one of the largest obstacles to their practical use is the severe computational complexity of their on-line training algorithms. The research community has proposed algorithms that significantly reduce the computational complexity for on-line applications, but all algorithms so far have encountered two main disadvantages; the first being a degradation of model performance as the complexity is reduced, and second, the algorithms assume the RNN has multiple outputs.

This chapter reviewed the sequential Bayesian framework as a possible solution to the costly online RNN training problem. Various sequential Bayesian estimation algorithms such as the Kalman filter and the EKF are derived and discussed. One of the major problems of RNN training with the EKF is divergence during situations of high nonlinearity (as well as the high computational cost of training). The SEEK filter was proposed as a derivative based method for RNN training that may reduce divergence. However, the method is based on linearization (RTRL derivatives) which is expensive to compute.

A novel sequential Monte Carlo estimation algorithm known as the EnKF was then proposed for on-line learning of RNNs. The approach proposed in this chapter circumvents these limitations of previously proposed research (decrease in prediction accuracy as computational complexity is reduced, and assumptions of multiple output neurons) through the use of Monte Carlo based stochastic sampling. The main theoretical result is an online algorithm for training RNNs with a computational complexity of approximately $O(H^2)$. Further experimental studies show that in practice the EnKF performance (in terms of training times, and out-of-sample prediction errors) is superior to popular

training algorithms such as the RNN-EKF and the RNN-RTRL. These results show that the proposed online ensemble based training algorithm reduces computational complexity without sacrificing the modeling capability of the RNN.

Chapter 6

Sequential Maximum Likelihood Learning for RNNs

The models discussed in the previous chapter are useful for sequential learning tasks, assuming *a priori* knowledge of the noise distributions over the process and measurement equations. In cases when these quantities are unknown, extensive tuning must take place or, the hyperparameters may be estimated during the training of the model (the second approach being more efficient).

This chapter elaborates on a method proposed to overcome the need for *a priori* knowledge of the noise distributions when building models from time series data with RNNs. This is achieved through the use of the sequential Bayesian estimation framework, and an iterative technique for computing Maximum Likelihood estimates of the models hyper-parameters, known as the Expectation Maximization (EM) algorithm [192, 193]. The approach capitalizes on the strengths of extended Kalman filtering and smoothing for dynamic estimation, and the probabilistic iterative parameter re-estimation algorithm for training RNNs. This approach allows for the estimation of both the model uncertainty and the noise in the data [192], and was introduced to machine learning in [33, 61]. The Kalman filter and smoother are utilized for parameter estimation, and the EM algorithm is used for hyper-parameter estimation. This alleviates the need to tune hyper-parameters to fit the model as they are adapted in the maximization step, which leads to improved out-of-sample performance on time series forecasting tasks [135, 138, 139].

6.1 Maximum Likelihood Learning via Expectation Maximization

In the Maximum likelihood estimation framework, it is assumed that the data $\mathcal{D}_t = (\mathbf{u}_t, d_t)$ are independently and identically distributed (i.i.d.). Given i.i.d. data, the complete likelihood can then be written as

$$L(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T, d_1, d_2, \ldots, d_T|\boldsymbol{\theta}) \propto \prod_{t=1}^{T} p(d_t|\mathbf{u}_t) \tag{6.1}$$

where $\boldsymbol{\theta} = [R, \mathbf{Q}, \boldsymbol{\mu}, \boldsymbol{\Sigma}]$ is the vector of the unknown hyperparameters, where R and $\mathbf{Q}$ are the measurement variance and process covariance in the Kalman filter, and $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ are the initial mean and variance of the Kalman filter. For time series modeling, the observations may not be be independent, so the likelihood is rewritten as [75]

$$L(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T, d_1, d_2, \ldots, d_T|\boldsymbol{\theta}) \propto \prod_{t=1}^{T} p(d_t|d_{1:t-1}\mathbf{u}_t) \tag{6.2}$$

In this formulation the probability density function $p(d_t|d_{1:t-1}\mathbf{u}_t)$ is written so that the distribution of d_t is now conditioned on the past $d_{1:t-1}$.

These assumptions are convenient in a computational sense as the mean and covariance of this conditional distribution can be computed via the family of sequential Bayesian filters, (i.e., Kalman filters) [192].

The innovations form of the log-likelihood function of the parameters $\mathbf{w}$ and hyperparameters $\boldsymbol{\theta}$ can be written down as [70]

$$\log L(\boldsymbol{\theta}) \propto -\frac{1}{2}\sum_{t=1}^{T} \log|R + \mathbf{P}_t^f| - \frac{1}{2}\sum_{t=1}^{T}(d_t - h(\mathbf{w}_t, \mathbf{u}_t))^{\mathrm{T}}(\mathbf{P}_t^f + R)^{-1}(d_t - h(\mathbf{w}_t, \mathbf{u}_t)) \tag{6.3}$$

The task is then to maximize the log-likelihood through optimization of the RNN parameters (weights) and hyperparameters. However the likelihood function is a highly nonlinear function of the unknown parameters and hyperparameters. To solve for the hyperparameters $\boldsymbol{\theta}$, the expectation maximization algorithm can be utilized [192, 193]. For completeness, the derivation of the EM algorithm is provided following an explanation given in [193]. The framework starts with the indirect maximization of the joint or

complete data likelihood

$$L(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_T, d_1, d_2, \ldots, d_T|\boldsymbol{\theta}) \tag{6.4}$$

of the observed measurements $d_1, \ldots, d_T$ and the unobserved hidden states $\mathbf{w}_1, \ldots, \mathbf{w}_T$. In the EM framework, the set of observed measurements $d_{1:T}$ along with the unobserved hidden states $\mathbf{w}_{1:T}$ make up the complete data set. The EM algorithm is a general framework for finding maximum likelihood estimates of this incomplete data set. The method was introduced by Dempster et al. [38] as a unified framework of several iterative procedures proposed by other authors, e.g.[74, 205] as discussed in [107].

The above joint likelihood $L(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta})$ is a function of the complete data set given the unknown but constant vector of hyperparameters $\boldsymbol{\theta}$. With the RNN weights $\mathbf{w}_{1:T}$ treated as unobserved variables, the log-likelihood function cannot be decomposed any further. To obtain the marginal probability the following must be solved

$$\log L(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta}) = \log \int p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta}) d\mathbf{w}_{1:T} \tag{6.5}$$

The EM algorithm maximizes the above equation indirectly, through a two step iterative process. In the first step of the EM algorithm, the RNN weights $\mathbf{w}_{1:T}$ are estimated given the current set of hyperparameters $\boldsymbol{\theta}$ (i.e , the integral in Equation 6.5 is computed). The second step then estimates the hyperparameters $\boldsymbol{\theta}$ (that were used in the previous step) given the newly updated RNN weights. Then, the process starts all over again by re-estimating the RNN weights with the new hyperparameters $\boldsymbol{\theta}$.

Assuming some arbitrary distribution q over the RNN weights $\mathbf{w}_{1:T}$, the log-likelihood of the observed data $\log L(d_{1:T}|\boldsymbol{\theta}) = \log \int p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta}) d\mathbf{w}_{1:T}$ can then be expanded as

follows:

$$\begin{aligned}
\ln \int p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta}) d\mathbf{w}_{1:T} &= \ln \int q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) \frac{p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta})}{q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta})} d\mathbf{w}_{1:T} \\
&\geq \int q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) \ln \frac{p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta})}{q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta})} d\mathbf{w}_{1:T} \\
&= \int q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) \ln p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta}) d\mathbf{w}_{1:T} \\
&\quad - \int q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) \ln q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) d\mathbf{w}_{1:T} \\
&= \mathcal{F}(q, \boldsymbol{\theta})
\end{aligned} \tag{6.6}$$

where in the first line, the likelihood function is rewritten by multiplying by

$$q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) / q(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) \tag{6.7}$$

The inequality on the middle line is Jensen's inequality [15]. This leads to the lower bound $\mathcal{F}(q, \boldsymbol{\theta})$ on the third line, which is just the negative of the Kullback-Leiber divergence [38]. Defining energy as $-\log p(d_{1:T}, \mathbf{w}_{1:T}|\boldsymbol{\theta})$ and starting with the configuration $\{d_{1:T}, \mathbf{w}_{1:T}\}$, then the lower bound is the free energy $\mathcal{F}(q, \boldsymbol{\theta})$, which is the expected energy under q minus the entropy of q [149]. The EM algorithm alternates between maximization of the lower bound $\mathcal{F}(q, \boldsymbol{\theta})$ with respect to q and $\boldsymbol{\theta}$, holding one or the other fixed.

Starting with some initial condition for the hyperparameters $\boldsymbol{\theta}^{old} = \boldsymbol{\theta}_0$ the Expectation and Maximization steps are as follows

$$\begin{aligned}
q^{new} &= \arg\max_{q} \mathcal{F}(q, \boldsymbol{\theta}^{old}) \\
\boldsymbol{\theta}^{new} &= \arg\max_{\boldsymbol{\theta}} \mathcal{F}(q^{new}, \boldsymbol{\theta})
\end{aligned} \tag{6.8}$$

The Expectation step is maximized when

$$q^{new}(\mathbf{w}_{1:T}|d_{1:T}, \boldsymbol{\theta}) = p(\mathbf{w}_T|d_T, \boldsymbol{\theta}^{old}) \tag{6.9}$$

At its maximum, the lower bound is $\mathcal{F}(q, \boldsymbol{\theta}) = \log L(\mathcal{D}_T|\boldsymbol{\theta})$.

The optimum (maximum) in the maximization step, is computed by maximizing

$$\int q(\mathbf{w}_{1:T}|d_{1:T},\boldsymbol{\theta}) \ln p(d_{1:T},\mathbf{w}_{1:T}|\boldsymbol{\theta}) d\mathbf{w}_{1:T} \tag{6.10}$$

which leads to

$$\begin{aligned} \boldsymbol{\theta}^{new} &= \arg\max_{\boldsymbol{\theta}} \int \log p(d_{1:T},\mathbf{w}_{1:T}|\boldsymbol{\theta}) p(\mathbf{w}_{1:T}|\mathcal{D}_{1:T},\boldsymbol{\theta}) d\mathbf{w}_{1:T} \\ &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}\Big[\log p(\mathbf{w}_{1:T},d_{1:T}|\boldsymbol{\theta}^{old})\Big] \end{aligned} \tag{6.11}$$

which is the form found in [38] used to describe the EM algorithm. The name of the algorithm, EM originates from the two step iterative process that is used to increase the log-likelihood function $\log L(d_T|\boldsymbol{\theta})$, i.e., computing the expectation and then maximizing the resulting quantity. This process can be summarized as follows:

- Initialization Step: The parameters are initialized and the hyperparameters are set to $\boldsymbol{\theta}^{old} = \boldsymbol{\theta}_0$.
- Expectation-step: The expectation of the complete data conditioned on the current values of the hyperparameters $\boldsymbol{\theta}^{old}$ is computed

$$Q(\boldsymbol{\theta},\boldsymbol{\theta}^{old}) = \mathbb{E}\Big[\log p(\mathbf{w}_{1:T},d_{1:T})|d_{1:T},\mathbf{u}_{1:T},\boldsymbol{\theta}^{old}\Big] \tag{6.12}$$

- Maximization-step: The hyperparameters $\boldsymbol{\theta}^{new}$ are solved for such that $Q(\boldsymbol{\theta},\boldsymbol{\theta}^{old})$ is maximized

$$\theta^{new} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta},\boldsymbol{\theta}^{old}) \tag{6.13}$$

  This is achieved via differentiation of $Q(\boldsymbol{\theta},\boldsymbol{\theta}^{old})$ with respect to the hyperparameters $\boldsymbol{\theta} = [R,\mathbf{Q},\boldsymbol{\mu},\Sigma]$ and solving for the desired quantities. The process then repeats it self.

The proposed approach for training RNNs follows that of [61, 192] in that the E-step uses the Kalman filtering and smoothing recursions. In the E-step, an estimate of the state $\mathbf{w}$ given the data $d_{1:T}$ and hyper-parameters $\boldsymbol{\theta} = [R,\mathbf{Q},\boldsymbol{\mu},\Sigma]$ is produced. The noise hyperparameters R, and $\mathbf{Q}$ of the EKF are assumed to be unknown *a priori*. In the M-step, the parameters $\boldsymbol{\theta}$ are then estimated given the new weight/parameter estimate.

The M-step uses the likelihood function to solve for the hyperparameters. One of the beneficial properties of the EM algorithm is that the log-likelihood always increases or stays constant after each cycle of E- and M-steps [38].

The set of hyperparameters $\boldsymbol{\theta}$ is initialized with fixed constant prior values. Via the Markov properties and independence assumptions introduced in Section 5.1, the joint likelihood of the data (Equation 6.4) is written as

$$L(\mathbf{w}_0, \ldots, \mathbf{w}_T, d_1, \ldots, d_T|\boldsymbol{\theta}) = p(\mathbf{w}_{0:T}, \boldsymbol{d}_{1:T}|\boldsymbol{\theta}) \tag{6.14}$$

where the conditional probability can be expanded as [43]:

$$p(\mathbf{w}_{0:T}, \boldsymbol{d}_{0:T}|\boldsymbol{\theta}) = p(\mathbf{w}_{0:T}|\boldsymbol{\theta})p(\boldsymbol{d}_{0:T}|\mathbf{w}_{0:T}, \boldsymbol{\theta}) \tag{6.15}$$

The right hand side of Equation 6.15 can be broken down further as

$$p(\boldsymbol{d}_{0:T}|\mathbf{w}_{0:T}, \boldsymbol{\theta}) = \prod_{\tau=1}^{T} p(d_\tau|\mathbf{w}_\tau, \boldsymbol{\theta}) \tag{6.16}$$

and

$$p(\mathbf{w}_{0:T}|\boldsymbol{\theta}) = p(\mathbf{w}_0, \boldsymbol{\theta}) \prod_{\tau=1}^{T} p(\mathbf{w}_\tau|\mathbf{w}_{\tau-1}, \boldsymbol{\theta}) \tag{6.17}$$

putting these expressions together results in the the joint likelihood of the complete data

$$p(\mathbf{w}, \boldsymbol{d}_{1:T}|\boldsymbol{\theta}) = p(\mathbf{w}_0|\boldsymbol{\theta}) \prod_{\tau=1}^{T} p(\mathbf{w}_\tau|\mathbf{w}_{\tau-1}, \boldsymbol{\theta}) \prod_{\tau=1}^{T} p(d_\tau|\mathbf{w}_\tau, \boldsymbol{\theta}) \tag{6.18}$$

It is assumed that the following probabilities are Gaussian:

$$p(\mathbf{w}_0|\boldsymbol{\theta}) = \frac{1}{(2\pi)^{m/2}|\boldsymbol{\Sigma}|^{1/2}} exp\Big[-\frac{1}{2}(\mathbf{w}_0 - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{w}_0 - \boldsymbol{\mu}) \Big] \tag{6.19}$$

$$p(\mathbf{w}_t|\mathbf{w}_{t-1}, \boldsymbol{\theta}) = \frac{1}{(2\pi)^{m/2}|\mathbf{Q}|^{1/2}} exp\Big[-\frac{1}{2}(\mathbf{w}_t - \mathbf{w}_{t-1})^{\mathrm{T}} \mathbf{Q}^{-1} (\mathbf{w}_t - \mathbf{w}_{t-1}) \Big] \tag{6.20}$$

$$p(d_t|\mathbf{w}_t, \boldsymbol{\theta}) = \frac{1}{(2\pi)^{c/2}|R|^{1/2}} exp\Big[-\frac{1}{2}(d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2 R^{-1} \Big] \tag{6.21}$$

where the initial value of $\mu = \mathbf{w}_1^T$, the initial covariance is defined as $\Sigma = \mathbf{P}_1^T$.

The "complete data" $\{\mathbf{w}_{1:T}, \mathcal{D}_{1:T}\}$ which includes the set all of RNN weights and the set of all observations is used to maximize the likelihood of the RNN weights and the observations given the hyperparameters [192]

$$\begin{aligned}\ln p(\mathbf{w}, \mathcal{D}_{1:T}|\boldsymbol{\theta}) = L_1 - \frac{T}{2}\ln|R| - L_2 - \frac{T-1}{2}\ln|\mathbf{Q}| \\ - L_3 - \frac{1}{2}\ln|\Sigma| - \frac{T(m+c)}{2}\ln(2\pi)\end{aligned} \tag{6.22}$$

where the constants c, and m are the dimension of the output of the model and the number of weights respectively. The substitutions are:

$$\begin{aligned}L_1 &= -\sum_{t=1}^{T}\left[\frac{1}{2}(d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2 R^{-1}\right] \\ L_2 &= -\sum_{t=1}^{T}\left[\frac{1}{2}(\mathbf{w}_t - \mathbf{w}_{t-1})^{\mathrm{T}}\mathbf{Q}^{-1}(\mathbf{w}_t - \mathbf{w}_{t-1})\right] \\ L_3 &= -\frac{1}{2}(\mathbf{w}_0 - \boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\mathbf{w}_0 - \boldsymbol{\mu})\end{aligned} \tag{6.23}$$

6.1.1 Maximization Step

To find the set of hyperparameters θ that maximize the likelihood of the "complete data," the expectation of Equation 6.24 is taken.

$$\begin{aligned}\mathbb{E}[\ln p(\mathbf{w}_{0:T}, \mathcal{D}_{1:T}|\boldsymbol{\theta})] = -\sum_{t=1}^{T}\frac{1}{2}\mathbb{E}[(d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2 R^{-1}] - \frac{T}{2}\ln|R| \\ - \frac{1}{2}\sum_{t=1}^{T}\mathbb{E}[(\mathbf{w}_t - \mathbf{w}_{t-1})^{\mathrm{T}}\mathbf{Q}^{-1}(\mathbf{w}_t - \mathbf{w}_{t-1})] - \frac{T}{2}\ln|\mathbf{Q}| \\ - \frac{1}{2}\mathbb{E}[(\mathbf{w}_0 - \mu)\pi(\mathbf{w}_0 - \mu)] - \frac{1}{2}\ln|\pi| - \frac{Tc + (T+1)m}{2}\ln(2\pi)\end{aligned} \tag{6.24}$$

which can be written as

$$
\begin{aligned}
Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = &- \sum_{t=1}^{T} \frac{1}{2} \mathrm{tr}[(d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2 R^{-1}] - \frac{T}{2} \ln |R| \\
&- \frac{1}{2} \sum_{t=1}^{T} \mathrm{tr}[(\mathbf{w}_t - \mathbf{w}_{t-1})^{\mathrm{T}} \mathbf{Q}^{-1} (\mathbf{w}_t - \mathbf{w}_{t-1})] - \frac{T}{2} \ln |\mathbf{Q}| \\
&- \frac{1}{2} \mathrm{tr}[(\mathbf{w}_0 - \mu) \pi (\mathbf{w}_0 - \mu)] - \frac{1}{2} \ln |\pi| - \frac{Tc + (T+1)m}{2} \ln(2\pi)
\end{aligned}
\tag{6.25}
$$

where tr is trace operator. To compute the above conditional expectations, the EKF, with hyperparameters set to $\boldsymbol{\theta}^{old}$, is utilized (this happens in the Expectation step). The above equations are then solved analytically for the hyperparameters in the maximization step.

This happens through differentiating the resulting expectation of the log-likelihood with respect to R^{-1}, yielding

$$
\begin{aligned}
\frac{\partial}{\partial R^{-1}} \mathbb{E}[\ln p(\mathbf{w}, \mathcal{D}_{1:T} | \boldsymbol{\theta})] &\approx \frac{1}{2} \frac{\partial}{\partial R^{-1}} \Big(\frac{T}{2} \ln |R^{-1}| - L_4 \Big) \\
&= R \frac{T}{2} - \sum_{t=1}^{N} \frac{1}{2} \Big[\mathbf{j}_t^{\mathrm{T}} \mathbf{P}_t^T \mathbf{j}_t + (d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2 \Big]
\end{aligned}
\tag{6.26}
$$

where the substitution of L_4 is defined as

$$
L_4 = \sum_{t=1}^{T} tr \Big(R^{-1} [\mathbf{j}_t^{\mathrm{T}} \mathbf{P}_t^T \mathbf{j}_t + (d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2] \Big) \tag{6.27}
$$

Setting the resulting solution equal to zero and solving for R results in

$$
R = \frac{1}{T} \sum_{t=1}^{T} \Big[\mathbf{j}_t^{\mathrm{T}} \mathbf{P}_t^T \mathbf{j}_t + (d_t - h(\mathbf{w}_t, \mathbf{u}_t))^2 \Big] \tag{6.28}
$$

Similarly, differentiating with respect to $\mathbf{Q}$ yields:

$$
\frac{\partial}{\partial \mathbf{Q}^{-1}} \mathbb{E}[\ln p(\mathbf{w}, \mathcal{D}_{1:T} | \boldsymbol{\theta})] \approx \frac{T-1}{2} \mathbf{Q} - \frac{1}{2} (\mathbf{C} - 2\mathbf{B}^{\mathrm{T}} + \mathbf{A}^{\mathrm{T}}) \tag{6.29}
$$

which after equating to zero and solving for $\mathbf{Q}$ leads to

$$\mathbf{Q} = \frac{1}{N-1}(\mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^{\mathrm{T}}) \tag{6.30}$$

where the matrix quantities are

$$\begin{aligned}
\mathbf{A} &= \sum_{t=1}^{T} \left[\mathbf{P}_{t+1}^{T} + \mathbf{w}_{t-1}^{T}(\mathbf{w}_{t-1}^{T})^{\mathrm{T}}\right] \\
\mathbf{B} &= \sum_{t=1}^{T} \left[\mathbf{P}_{t,t+1}^{T} + \mathbf{w}_{t}^{T}(\mathbf{w}_{t-1}^{T})^{\mathrm{T}}\right] \\
\mathbf{C} &= \sum_{t=1}^{T} \left[\mathbf{P}_{t}^{T} + \mathbf{w}_{t}^{T}(\mathbf{w}_{t}^{T})^{\mathrm{T}}\right]
\end{aligned} \tag{6.31}$$

It is also possible to solve for the initial conditions in the M step, via differentiation of the expectation of the log-likelihood function with respect to the initial mean

$$\frac{\partial}{\partial \boldsymbol{\mu}}\mathbb{E}[\ln p(\mathbf{w}, \mathcal{D}_{1:T}|\boldsymbol{\theta})] \approx \frac{1}{2}\Sigma^{-1}(-2\mathbf{w}_1^T + 2\boldsymbol{\mu}) \tag{6.32}$$

which yields the initial value of $\boldsymbol{\mu} = \mathbf{w}_1^T$ and similarly for the covariance, taking the derivative with respect to Σ yields

$$\frac{\partial}{\partial \Sigma^{-1}}\mathbb{E}[\ln p(\mathbf{w}, \mathcal{D}_{1:T}|\boldsymbol{\theta})] \approx \frac{1}{2}\Sigma - \frac{1}{2}(\mathbf{w}_1^T - \boldsymbol{\mu})^{\mathrm{T}}(\mathbf{w}_1^T - \boldsymbol{\mu}) + \mathbf{P}_1^T \tag{6.33}$$

which leads to the initial covariance $\Sigma = \mathbf{P}_1^T$.

The algorithm alternates between two steps, the E-step and the M step. Starting off with an initial guess for $\boldsymbol{\theta}$, in the E-step, the expected values of $\mathbf{w}_t^T$, $\mathbf{P}_t^T$ and $\mathbf{P}_{t,t-1}^T$ are obtained from their current estimates through extended Kalman filtering and smoothing. Then, in the M-step, the new values of $\boldsymbol{\theta}$ are obtained using the above equations.

6.1.2 Expectation Step

For general nonlinear systems the conditional density $p(\mathbf{w}_{0:T}|y_{1:T}, \boldsymbol{\theta})$ is non-Gaussian. A classical and computationally efficient approach from engineering and control literature which provides a Gaussian approximation to the joint posterior, is the extended Kalman

filter and smoother. The extended Kalman filter and smoother is based on a first order Taylor series expansion of the nonlinearity $h(\cdot,\cdot)$ about the Kalman filter predictor $\mathbf{w}_{t-1}^a$ leading to time variant dynamics.

Training the RNN using the EKF requires computation of the Jacobian $\mathbf{j}_t$ which is a vector of first order derivatives of the error with respect to the weights of the RNN computed at each time step. The Jakobian is computed via Equation 3.31 as in [23]. The following equations describe the EKF training algorithm:

$$\begin{aligned}
\mathbf{w}_t^f &= \mathbf{w}_{t-1}^a \\
\mathbf{P}_t^f &= \mathbf{P}_{t-1}^a + \mathbf{Q} \\
\mathbf{k}_t &= \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} [\mathbf{j}_t \mathbf{P}_t^f \mathbf{j}_t^{\mathrm{T}} + R]^{-1} \\
\mathbf{w}_t^a &= \mathbf{w}_t^f + \mathbf{k}_t (d_t - h(\mathbf{w}_t^f, \mathbf{u}_t)) \\
\mathbf{P}_t^a &= \mathbf{P}_t^f - \mathbf{k}_t \mathbf{j}_t \mathbf{P}_t^f
\end{aligned} \tag{6.34}$$

where the vector $\mathbf{k}_t$ is the Kalman gain. The EKF is a suboptimal estimator based on linearization of the nonlinearity of the underlying system. It provides an approximation of the state mean $\mathbf{w}_t$ and the state covariance $\mathbf{P}_t$.

After computing the estimates $\mathbf{w}_t$ and $\mathbf{P}_t$ by Equations (6.34) the Rauch-Tung-Striebel smoother [174] is utilized for recursively computing corrections to the EKF estimates. This is achieved through the following backward recursions:

$$\begin{aligned}
\mathbf{S}_{t-1} &= \mathbf{P}_{t-1}^a (\mathbf{P}_t^f)^{-1} \\
\mathbf{w}_{t-1}^T &= \mathbf{w}_{t-1}^a + \mathbf{S}_{t-1} (\mathbf{w}_t^T - \mathbf{w}_t^f) \\
\mathbf{P}_{t-1}^T &= \mathbf{P}_{t-1}^a + \mathbf{S}_{t-1} (\mathbf{P}_t^T - \mathbf{P}_t^f) \mathbf{S}_{t-1}^{\mathrm{T}} \\
\mathbf{P}_{t,t-1}^T &= \mathbf{P}_t^a \mathbf{S}_{t-1}^{\mathrm{T}} + \mathbf{S}_t (\mathbf{P}_{t+1,t}^T - \mathbf{P}_t^a) \mathbf{S}_{t-1}^{\mathrm{T}}
\end{aligned} \tag{6.35}$$

with boundary condition $\mathbf{P}_{T,T-1}^T = (\mathbf{I} - \mathbf{k}_T \mathbf{j}_T) \mathbf{P}_{T-1}^{T-1}$.

6.2 Experimental Results

The practical task addressed using EM trained RNNs is modeling and forecasting hourly electricity spot prices. To test the effectiveness of the proposed model, data from two power exchanges is used for model validation. The data sets include prices from the hourly Ontario energy price (HOEP), and the Spanish power exchange. In both ex-

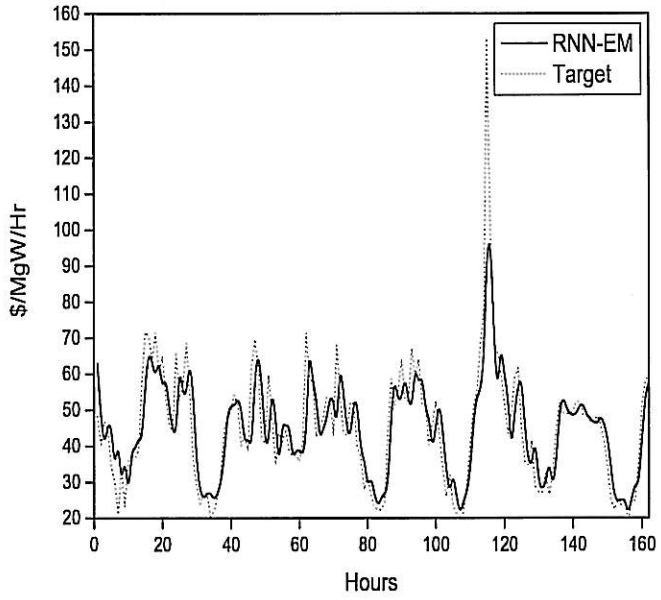

Figure 6.1: Out-of-sample predictions of the Ontario spot prices. The model forecasts 1 hour ahead, for 7 days, starting from April 26, 2004, which corresponds to 168 hours of forecasted values shown above.

periments, the data considered was hourly Market Clearing Prices (MCP). Although the use of exogenous variables other than MCPs (such as electricity load, temperatures, etc.) have been proposed in previous studies, it has been inconclusive as to which variables, if any, contribute to increased explanatory power of a model [29, 67, 154]. This chapter utilizes a single time series of previous prices as the input to the RNN. In order to allow for comparison of the proposed model to previous work, 48 days of data prior to the test set was used to construct the training series, which corresponds to 1152 training examples as in previous studies [7, 29]. The models were then evaluated on the subsequent data set, which consisted of 7 consecutive days of unseen data (which is called the out of sample

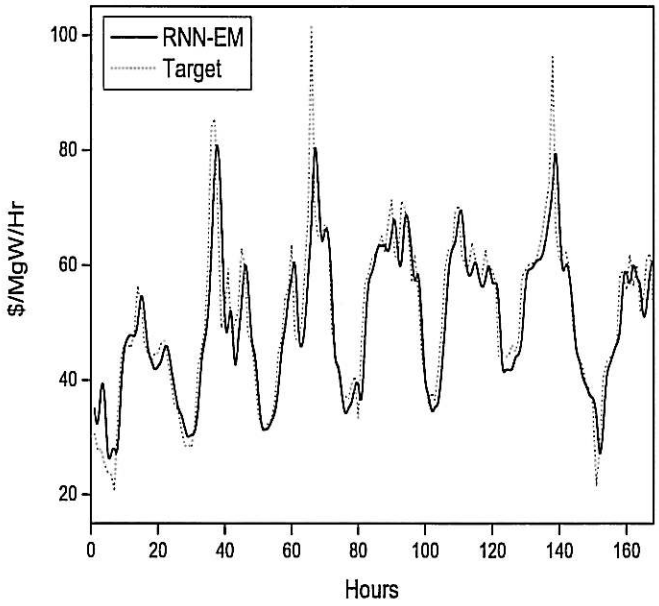

Figure 6.2: Out-of-sample forecasts of the second segment of the Ontario data set. The models forecast 1 hour ahead, for 7 days, starting from August 2, 2004, which corresponds to 168 hours of forecasted values shown above.

data set), corresponding to the next 168 hourly measurements. A test set length of 7 days was used so as to make results comparable to previous studies.

To provide a further comparison with other nonlinear time series models, several well known training algorithms for recurrent networks relevant to the RNN-EM have been implemented, including the Extended Kalman Filter for recurrent networks (RNN-EKF) [23]. These algorithms are compared with a standard (feed-forward) MLP neural network trained with the Extended Kalman Filter (MLP-EKF), and also an MLP neural network trained with the EM algorithm [33]. All data sets were pre-normalized between the values $[0, 1]$. In all simulations, the weights of the networks were initialized with random uniformly distributed weights in the range of $[-2, 2]$. The simulations were carried out on a Pentium-4 2.6-GHz, 1024-MB RAM workstation. The Mean Absolute Percentage Error (MAPE) is a standard error measure found in the electricity demand/spot price forecasting literature [35, 204], and is used here to report errors. Errors from previous studies on the same data set featured in this section were reported in MAPE [29]. So to make results comparable, errors are reported in MAPE, computed by $MAPE = (100/(168)) \sum_{i=1}^{168} |d_i - y_i|/d_i$ where d_i is the actual price and y_i is forecasted

price.

6.2.1 Ontario Electricity Market Price Forecasting

In the first experiment, the predictive performance of all implemented networks are evaluated on the Ontario spot price data series. The Ontario Electricity Market (OEM) is one of the newest and largest electricity markets North America. It became a competitive market in 2002. The OEM provides power to more that 12 million people and is linked up to other power pools such as the PJM power pool. The MCPs in the OEM are determined every

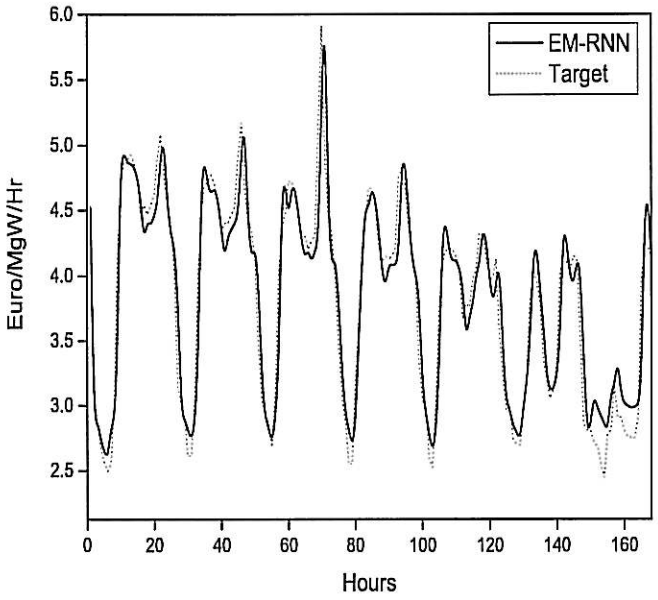

Figure 6.3: Out-of-sample one hour ahead forecasts on the Spring week of the Spain data set starting from May 20, 2004 hour 1 and continues for 168 hrs. The bold solid line represents the forecast, and the dotted line represents the actual spot price, in euros per megawatts per hour

five minutes, and hourly averages are then computed. These hourly averages are called the hourly Ontario energy price (HOEP). As a service to its users, the OEM provides hourly forecasts of the HOEP, and sends the forecast to all market participants. However, the OEM provided forecasts are known to contain significant errors [178]. Here the HOEP prices from three seasons Spring, Summer, and Winter are considered. To facilitate comparability of the proposed model to previous work, the test sets of this section follow the

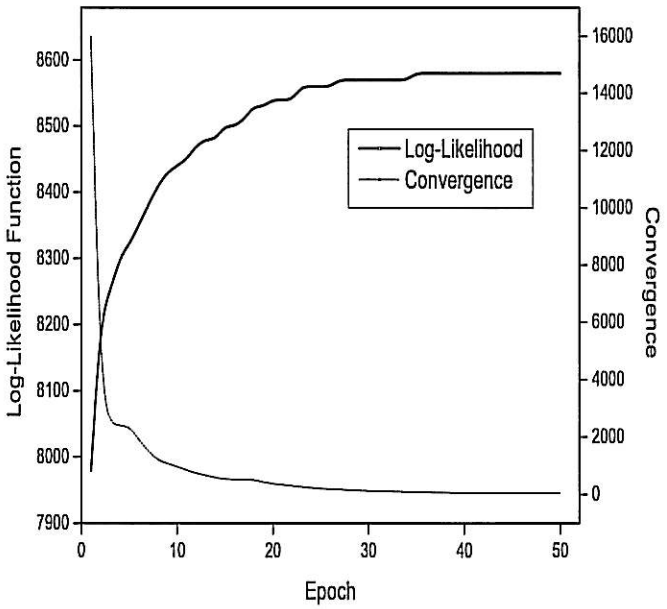

Figure 6.4: Per-Epoch convergence and log-likelihood of the model, evaluated on spring Ontario spot prices from March 11, to April 25, 2004. The bold solid curve shows an increasing likelihood of the model as training progresses. The dashed line shows the convergence of the model (rate of change of the likelihood).

same dates as tested in [4, 232]. Each training set consisted of 48 days (1152 hourly measurements) followed by an out-of-sample test set consisting of two weeks of price data (168 hourly measurements)[1]. The errors of the test sets were reported per week. The spring data set included price measurements from March 11 to May 9, 2004 from which the first 48 days (March 11, to April 25, 2004) were used for training and the remaining two weeks (April 26 to May 9, 2004) were used for out-of-sample forecasts. The Summer data set consisted of data ranging from June 8 to August 8, 2004. The training set ran from June 8 to July 25, 2004 and the subsequent two weeks of July 26 to August 8 were used for the test set. The final data set consisted of data points from October 26, to December 26, 2004, in which the dates of October 26, to December 12, 2004 belonged to the training set. The test set included the dates of December 13 to December 26, 2004. The two recurrent networks were initialized with 3 hidden neurons, one output neuron, and input parameters $\tau = 6$ and $d = 5$. The feed-forward networks were constructed with 6 hidden neurons, and 6 inputs. All models trained with the EKF filters were initialized

[1]The data is freely available at www.ieso.ca/

Table 6.1: One Hour Ahead Forecast Results on the Ontario Data Set (Scores Reported in MAPE)

	Spring		Summer		Winter	
	week 1	week 2	week 1	week 2	week 1	week 2
MLP-EKF	16.83	16.74	12.64	15.25	16.77	16.96
MLP-EM	15.48	15.39	11.87	12.07	16.78	16.73
RNN-EKF	16.01	16.54	11.89	11.96	16.59	16.45
RNN-EM	15.09	15.16	10.32	10.21	15.78	15.71
RULE	21.70	17.80	22.92	37.77	24.60	24.55
IESO	23.78	25.26	10.41	16.22	22.06	23.51
MLR	16.26	19.23	17.69	20.55	16.73	18.54
MLP	16.56	19.34	17.45	20.27	17.03	19.69
WMLP	15.21	18.62	17.91	18.72	16.61	18.02

with the R hyper-parameter set to $1.0e^{-5}$, and the diagonal elements of $[\mathbf{Q}]_{ii}$ set to $1.0e^{-2}$. The EM trained models were initialized with R and the diagonal elements in $[\mathbf{Q}]_{ii}$ set to 200.

Table (6.1) contains the MAPEs for each model over the out-of-sample periods tested. The scores reported in the upper half of the table correspond to the models tested in this paper, and the scores in the lower half are taken from an experimental section in a similar study [4] which tested a heuristic based model (RULE), the forecast provided by the Ontario power exchange (IESO), a linear regression model (MLR), a feed-forward MLP (MLP), and a wavelet based MLP (WMLP). For the first week of the Spring data set, on average, the RNN-EM outperforms the RNN-EKF and the feed-forward network trained with both the EKF and the EM algorithm as shown in the second column of Table (6.1). The proposed model also outperforms the models in the lower section of the table. The corresponding numerical results of the forecasting for week 1 are presented in Figure (6.1). Figure (6.2) presents the performance of the models on the second week of the Summer data set. Slightly lower errors were found when forecasting in the summer months, as shown in Table (6.1). However, the RNN-EM algorithm still manages to outperform the other Kalman trained models for both weeks. As reported in the previous study, the IESO provided a low error forecast for the first week of the Summer test set. The proposed RNN-EM reports an error score that is slightly higher. However, in the Winter months, the proposed RNN-EM significantly outperforms the IESO provided forecast. The proposed model also provides lower error forecasts than the other remaining Kalman trained models included in this study. The convergence of the algorithm was monitored

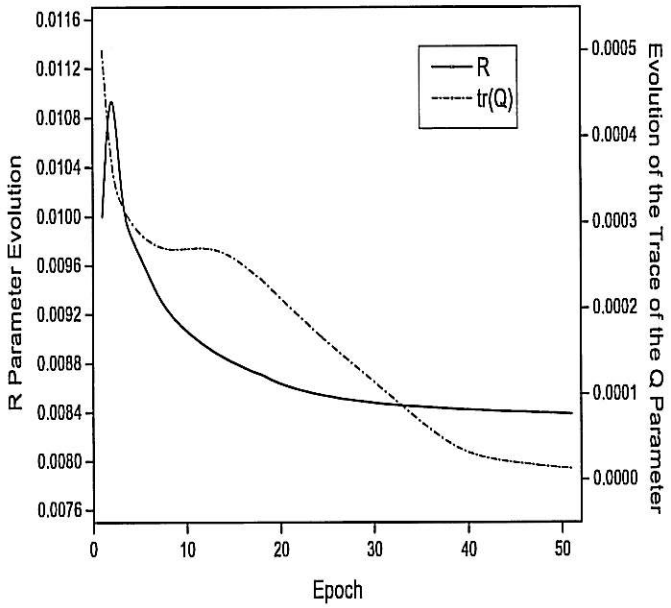

Figure 6.5: Evolution of the model hyper-parameters as training progresses on the spring Ontario spot prices. The bold solid line shows the convergence of the R parameter, and the dashed line shows the trace of the $\mathbf{Q}$ matrix.

over 50 iterations over the data. The solid line in Figure (6.4) shows the log-likelihood of the model over the training period. The likelihood monotonically increases as training progresses, which indicates good convergence of the model. The dashed curve shows the slope of the likelihood at each epoch. As training progresses, convergence is reached. The solid line in Figure (6.5) shows the EM estimates of the observation variance R quickly converge. The trace of the process noise covariance $\mathbf{Q}$ converges toward zero slower.

6.2.2 Forecasting of the Spanish Electricity Market

The second study investigates the performance of the proposed model on forecasting hourly prices from the Spanish electricity market. Competition in the power industry in Spain began in 1988, and is now the fifth largest electricity market in Europe. Here the data is taken from the hourly spot prices from the last week of February, May, August, and November, which are the same weeks tested in [22]. The training sets for building the forecast models included the previous 42 days of hourly data leading up to the test set. The dates of the Winter training set ranged from January 7 to February 17 2002, and the

test set ranged from February 18 to February 24, 2002. The Spring training set included

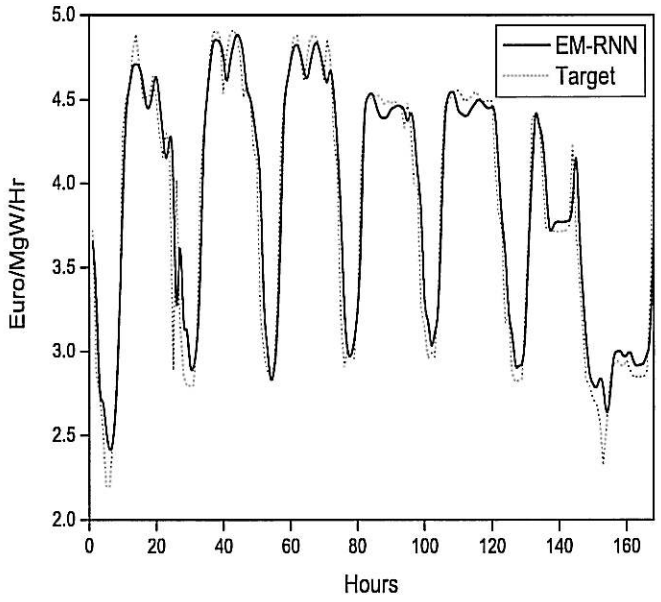

Figure 6.6: One hour ahead out-of-sample predictions of the Spain spot prices for the Summer week beginning on August 19, 2004 which corresponds to the first hour in the chart, and continues for 168 hrs. The bold solid line represents the forecast, and the dotted line represents the actual spot price, in euros per megawatts per hour.

data from April 8, to May 19 2002, and the test set ran from May 20 to May 26, 2002. The Summer training set used data from July 8, to August 18, 2002 and the following week August 19 to August 25, 2002 were used for the test set. Finally, the Fall data set included data taken from the dates of October 7, to November 17, 2002 and the test set was constructed from the dates of November 18 to November 24, 2002[2]. Each of the recurrent networks were initialized with 3 hidden neurons and one input and one output neuron. Similarly, the feed-forward networks were initialized with 3 hidden neurons, but used an input window of size 4. For the EKF trained networks, the initial diagonal elements of the covariance matrix for both $[\mathbf{Q}]_{ii}$ and R for all filters were set to $1.0e^{-3}$ and $1.0e^{-2}$ respectively, and for the EM trained models, both $[\mathbf{Q}]_{ii}$ and R were set to 100.

The upper portion of Table (6.2) provides the performance of each of the models on one hour ahead forecasting. The lower part of Table (6.2) shows results on the same data set reported in [22]. The model errors considered in [22] and reported in the lower

[2]The data was downloaded from www.omel.es

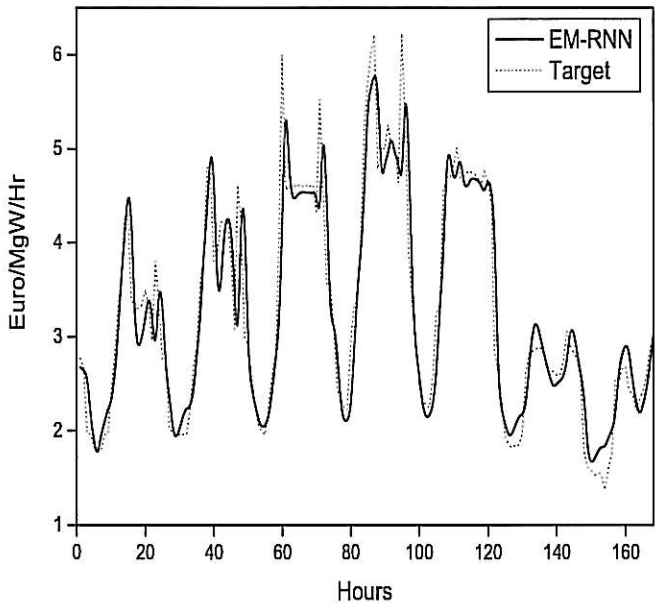

Figure 6.7: Out-of-sample one hour ahead forecast forecasts on the Fall week of the Spanish data set. The predictions begin on November 18, 2004, which corresponds to the first hour in the chart and continues for 168 hrs. The bold solid line represents the forecast, and the dotted line represents the actual spot price, in euros per megawatts per hour.

portion of Table (6.2) are the MLP trained with the second-order Levenberg-Marquardt algorithm (MLP-LM), the Autoregressive Integrated Moving Average linear time series model (ARIMA), and a Naive predictor (Naive) which is a random walk. The values in the second column shows the out-of-sample performance of the models on the Spring data set. The RNN-EM has the lowest error out of the models compared in the study. The MLP-EM algorithm and the RNN-EKF have similar performance, but the RNN-EM provides the lowest errors in predictions. The performance of the RNN-EM evaluated over the Spring out-of-sample data set are shown in Figure (6.3). For the one hour ahead forecasts on the Summer data set, the RNN-EM provides superior forecasts, as shown in column 3 of Table (6.2). The model predictions are shown in Figure (6.6). The fourth column of Table (6.2) reports the performance of the models on performance on the Fall data set. There was a greater difference in performance between the models on this data set, with the RNN based forecasts having lower errors than the MLP based models. Both RNN models performed better than the feed-forward models, with the RNN-EM perform-

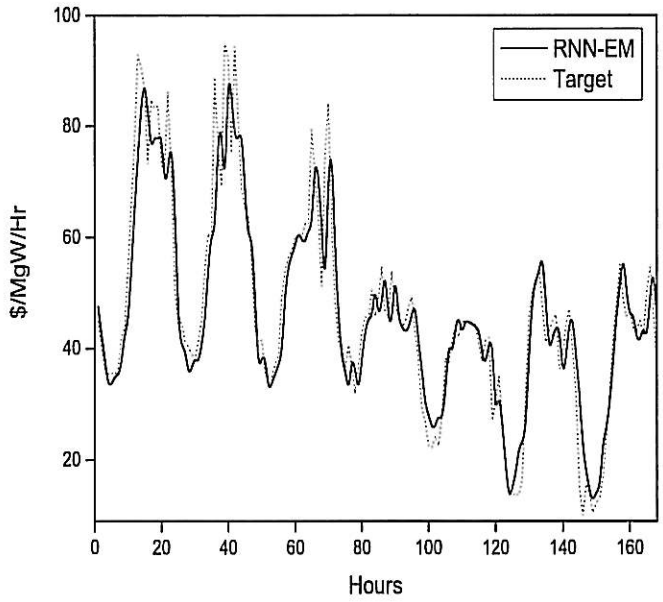

Figure 6.8: One hour ahead out-of-sample predictions of the Spain spot prices for the Winter week starting from January 7, 2002. The bold solid line represents the forecast, and the dotted line represents the actual spot price, in euros per megawatts per hour.

ing slightly better than the RNN-EKF. This suggests that the RNN may be more suitable for forecasting during the Fall than the MLP. The performance of the RNN-EM is shown in Figure (6.7). For the final (Winter) data set, there was not much variation in the performance between the models, as shown in the last column of Table (6.2). The prediction results of the RNN-EM are provided in Figure (6.7). In most cases the use of the EM algorithm has led to improvements in out-of-sample predictions over the EKF, and the RNN-EM more closely approximates the target series than the remaining algorithms.

After 50 epochs of training with the EM algorithm, the plots of the convergence properties of the model are provided in Figure (6.9). The solid curve shows the log-likelihood of the model increasing over the training period. The dashed curve shows the derivative of the likelihood at each epoch. In Figure (6.10), the solid line shows the EM estimates of the observation variance R which rapidly increase then approach a near constant value. The dashed curve shows the trace of $\mathbf{Q}$ approach zero.

Table 6.2: One hour ahead forecast performance on the Spain data sets (Scores Reported in MAPE)

Model	Spring	Summer	Fall	Winter
MLP-EKF	5.37	11.37	13.41	4.81
MLP-EM	5.30	11.32	13.08	4.36
RNN-EKF	5.27	11.38	9.04	4.39
RNN-EM	4.87	10.38	8.93	4.26
MLP-LM	5.36	11.40	13.65	5.23
ARIMA	6.36	13.39	13.78	6.32
Naive	7.27	27.30	19.98	7.68

6.3 Conclusion

Reliable forecasting of electricity spot prices is one of the most fundamental tasks for companies that trade in the electricity markets. Recently, RNNs have been proposed as a dynamical model suitable for capturing spot price dynamics. This paper extends the work in the area by proposing a probabilistic method for training recurrent neural models on electricity spot price time series. It is demonstrated that the EM training of RNNs has lead to improvements in prediction accuracy in out-of-sample forecasts of electricity spot prices. From numerical simulation on electricity spot data from two exchanges, it was found that the proposed model outperformed feed-forward and recurrent neural network algorithms on electricity spot price forecasting, as well as models proposed in previous studies. The presented results are encouraging and can serve as a stimulation for further investigation into modeling electricity prices from other markets.

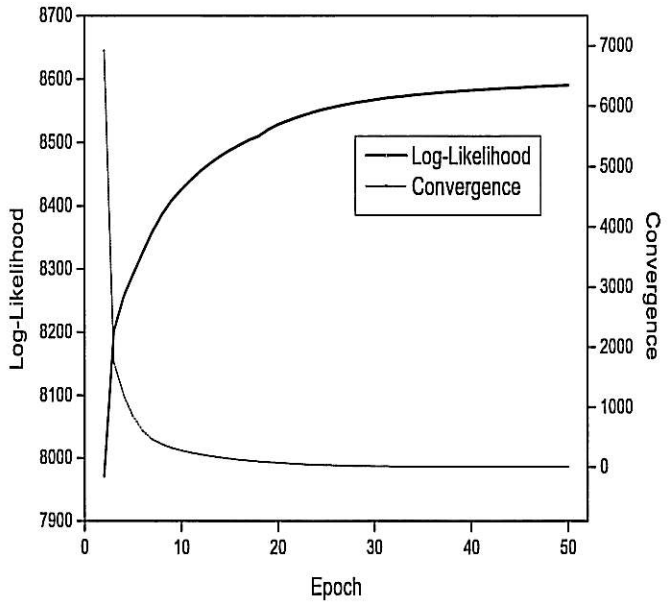

Figure 6.9: Per-Epoch convergence and log-likelihood of the model, evaluated on the Summer Spanish spot prices from July 8, to August 18, 2002. The bold solid curve shows an increasing likelihood of the model as training progresses. The dashed line shows the rate of change of the likelihood.

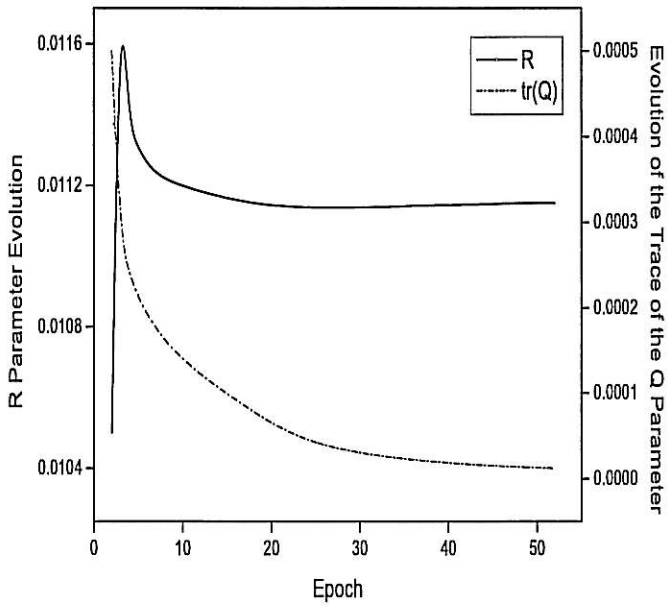

Figure 6.10: Evolution of the model hyper-parameters estimated with the EM algorithm on the Spanish summer spot prices training set. The bold solid line shows the convergence of the R parameter, and the dashed line shows the trace of the $\mathbf{Q}$ matrix.

Chapter 7

Summary and Conclusion

The previous chapters have dealt with the formulation and implementation of a computational framework for recursive Bayesian estimation of RNN parameters and hyperparameters. A theme common to all RNN training algorithms considered herein has been the inclusion of a recursively estimated regularization term which demands model smoothness and discourages over-fitting. It turns out that this method generally produces excellent results in terms of out of sample forecasts.

The remainder of this chapter further expands on these conclusions. It begins by recapitulating the main points of the thesis, including the novel features of the proposed algorithms and some of their shortcomings. Addressing these issues leads to a discussion on the possible paths for future work.

7.1 Summary

This dissertation can be summarized as follows

- Chapter 2 begins by distinguishing between the two main approaches to building a model of a dynamical system; the direct method and the indirect method. The direct approach to modeling assumes a deep understanding from first principles of the system so that equations of motion can be written down, and then solved for future states of the system. When less is known about the system, the inverse approach is usually taken. The inverse approach assumes the existence of a "teacher function" which has given rise to the observations. The teacher function takes the form of various time series model structures that are reviewed in the subsequent

sections of Chapter 2. The recurrent neural structures considered throughout this dissertation are then specified in the final section of the chapter.

- In Chapter 3, supervised learning algorithms for gradual refinement of the weights of the RNN architectures presented in Chapter 2 are reviewed. The chapter begins by defining RNN training as an optimization problem. Then the chapter reviews popular algorithms used for computing the derivatives of the RNN. Next, training algorithms are categorized into batch and sequential operation and further sub-categorized into first and second-order optimization classes. The strengths and weakness of the optimization algorithms with respect to RNN training are discussed. A gap in the literature is identified as no sequential Levenberg-Marquardt RNN training has been investigated.

- Chapter 4 investigates the sequential Levenberg-Marquardt RNN training from a Bayesian perspective. The chapter begins by reviewing ill posed problems, and then reviews solutions to the ill-posed problem via regularization. Then a recursively regularized learning algorithm based on second-order sequential Levenberg-Marquard optimization is developed. The forecast errors of the proposed model was compared to a group of first and second order trained neural models on a multitude of chaotic dynamical systems as well as a complex electricity spot price forecasting task. The proposed model showed improved generalization capacity over non-regularized RNNs (i.e. the proposed model had the lowest out of sample forecast errors over the other first and second order trained neural models).

  The novel features introduced in this chapter are restated here:

  1. incorporation of individual, non-uniform Bayesian regularization parameters for each weight to account for its uncertainties;
  2. handling of the noise through a specific noise hyperparameter;
  3. derivation of a regularized equation for recursive second-order estimation of the weights;
  4. formulation of an equation for the recursive computation of the inverted regularized dynamic Hessian matrix;
  5. estimation of Bayesian confidence intervals.

The main limitation to the proposed algorithm is the that large outliers are not handled well by the algorithm. There is no means to down-weight any large deviation from the mean of the data set.

- Chapter 5 presents the sequential Bayesian estimation framework for RNN training. The chapter first introduces a probabilistic view of RNN training via sequential Bayesian estimation, and then reviews nonlinear Kalman filtering algorithms as a means to compute posterior weight estimates. The EnKF is proposed as an efficient monte carlo filtering algorithm for RNN training. The filter is compared to various first and second-order RNN training algorithms, and it is found that the EnKF trained RNN provides a computationally efficient solution to RNN parameter estimation. Furthermore, the out of sample forecast errors of the EnKF trained RNN was found to be lower than other first and second-order RNN models which indicate that the proposed method is capable of producing well generalizing RNN time series models.

  The novel features of this chapter are as follows:

  1. an online ensemble approach to training RNNs which avoids derivative computation;
  2. reduction in computational complexity for online training to $O(H^2)$ which is two orders of magnitude lower than RTRL;

  The main limitation of the EnKF is covariance shrinkage. The EnKF systematically underestimates the error covariance during filtering, and in some cases an ad-hoc strategy for covariance inflation is necessary [97]. Although this has not been found to severely impact RNN training where the number of RNN outputs are less than the number of parameters, this can potentially become problematic when the number of RNN outputs exceeds the number of parameters [97].

- Chapter 6 investigates a maximum likelihood framework for RNN training. This is achieved through the use of the expectation maximization algorithm which allows for estimation of RNN parameters as well as hyperparameters. The framework builds on Kalman filtering for parameter estimation (discussed in Chapter 5), Kalman smoothing for estimation of the parameter covariances, and a probabilistic

iterative hyperparameter re-estimation algorithm. The proposed algorithm is evaluated on an electricity spot pricing task, and is compared to various models previously proposed in the literature. It was found that the EM trained RNN provided the lowest out of sample forecast errors over other first and second-order trained RNNs as well as other times series models proposed in the electricity spot pricing literature.

The novel feature of this chapter is:

1. a method for solving for the process and measurement noise hyperparameters when building models from time series data with RNNs.

The EM algorithm is known to be slow to converge. Even though the RNN parameters are estimated rapidly in the Expectation step, hyper parameter estimation takes more than a few epochs to reach convergence. This is a potential area for improvement.

7.2 Future Directions

This dissertation reviewed the literature and provided solutions for improving RNN time series modeling. Although improvements were realized, this study has opened the door to further research. Future research directions suggested by the presented material include:

- Further investigation of the sequential Levenberg Marquardt algorithm with heavy tailed noise distributions to account for outliers in the data.
- The Ensemble Kalman filter is known to occasionally suffer from "ensemble collapse." It is though that this phenomena can be avoided through an additional term in the cost function to promote negative correlation between ensemble members.
- The speed of convergence of the expectation maximization learning algorithm can be improved via conjugate gradient search. It is though that after computing the derivatives of the hyperparameters in the maximization stage, conjugate gradient search can then accelerate convergence of hyperparameter estimation.

7.3 Concluding Remarks

This dissertation has surveyed the literature in time series modeling with RNNs. State of the art algorithms for RNN training have been reviewed. Three novel approaches to building RNN models from time series data have been proposed. Comparisons have been made between the proposed algorithms and previously leading approaches in neural time series forecasting. It was found that the proposed models outperform other well known neural models in terms of out of sample forecast errors. The proposed algorithms are applied to the problems of climatological modeling and electricity spot price forecasting, which relate the developments within this dissertation to the wider field of modeling nonlinear dynamic systems.

Appendix A

Derivation of the BPTT Algorithm with Ordered Derivatives

This section provides a derivation of the BPTT algorithm from the perspective of ordered derivatives [223]. It is assumed that the gradient descent algorithm will be used. The rate of change of the cost function $C_T(\mathbf{w})$ is computed through the methodology of the ordered derivative which takes into account all influences on the weights with respect to the cost function:

$$\Delta w^{(i,j)} = -\eta \frac{\partial^+ C_T(\mathbf{w})}{\partial w^{(i,j)}} \tag{A.1}$$

Through the use of ordered derivatives, the weights $w^{(i,j)}$ are not written with time indexes, i.e. copies of the weights are not made at each time step. Throughout the derivation, the chain rules for ordered derivatives are invoked multiple times, so familiarization with the generic chain rules for ordered derivatives may be helpful [184]. The first chain rule is applied twice to the cost function as follows:

$$\begin{aligned}
\frac{\partial^+ C_T(\mathbf{w})}{\partial w^{(i,j)}} &= \frac{\partial C_T(\mathbf{w})}{\partial w^{(i,j)}} + \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \acute{\mathbf{s}}_t^{(i)}} \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(i,j)}} \\
&= 0 + \Big(\sum_{t=0}^{T} \frac{\partial C_T(\mathbf{w})}{\partial \acute{\mathbf{s}}_t^{(i)}} + \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} \Big) \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(i,j)}} \\
&= \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(i,j)}}
\end{aligned} \tag{A.2}$$

where the RNN weights $w^{(i,j)}$ of the ith neuron impacts the cost function indirectly via the summation block $\acute{\mathbf{s}}_t^{(i)}$ which propagates through the output of the ith neuron, resulting in the activation $\mathbf{s}_t^{(i)}$. As the RNN weights $w^{(i,j)}$ and the summation blocks $\acute{\mathbf{s}}_t^{(i)}$ are not directly impacting $C_T(\mathbf{w})$, the partial derivatives $\partial C_T(\mathbf{w})/\partial \mathbf{w}$ and $\partial C_T(\mathbf{w})/\partial \acute{\mathbf{s}}_t^{(i)}$ are zero.

Here, the chain rule is invoked to the first term twice in the third line of the above equation $\partial^+ C_T(\mathbf{w})/\partial \mathbf{s}_t^{(i)}$. As the outputs $\mathbf{s}_t^{(i)}$ impact the cost function $C_T(\mathbf{w})$ directly and indirectly through summation blocks $\acute{\mathbf{s}}_t^{(i)}$ and outputs $\mathbf{s}_t^{(i)}$ at the next time step, twice invoking the first chain rule results in:

$$\begin{aligned}
\frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} &= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} + \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \acute{\mathbf{s}}_{t+1}^{(i)}} \frac{\partial \acute{\mathbf{s}}_{t+1}^{(i)}}{\partial \mathbf{s}_t^{(i)}} \\
&= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} + \sum_{t=0}^{T} \Big(\frac{\partial C_T(\mathbf{w})}{\partial \acute{\mathbf{s}}_{t+1}^{(i)}} + \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(i)}} \frac{\partial \mathbf{s}_{t+1}^{(i)}}{\partial \acute{\mathbf{s}}_{t+1}^{(i)}} \Big) \frac{\partial \acute{\mathbf{s}}_{t+1}^{(i)}}{\partial \mathbf{s}_t^{(i)}} \\
&= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} + \sum_{t=0}^{T} \Big(0 + \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(i)}} \frac{\partial \mathbf{s}_{t+1}^{(i)}}{\partial \acute{\mathbf{s}}_{t+1}^{(i)}} \Big) \frac{\partial \acute{\mathbf{s}}_{t+1}^{(i)}}{\partial \mathbf{s}_t^{(i)}} \\
&= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} + \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(i)}} \frac{\partial \mathbf{s}_{t+1}^{(i)}}{\partial \acute{\mathbf{s}}_{t+1}^{(i)}} \frac{\partial \acute{\mathbf{s}}_{t+1}^{(i)}}{\partial \mathbf{s}_t^{(i)}}
\end{aligned} \tag{A.3}$$

Via the following conditions:

$$\frac{\partial C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} = \begin{cases} -e_t^{(i)} & \text{for } 0 \leq t \leq T \\ 0 & otherwise \end{cases} \tag{A.4}$$

$$\frac{\partial \mathbf{s}_t^{(l)}}{\partial \acute{\mathbf{s}}_t^{(l)}} = \sigma'(\acute{\mathbf{s}}_t^{(l)}) \quad \text{for all } t \tag{A.5}$$

$$\frac{\partial \acute{\mathbf{s}}_{t+1}^{(l)}}{\partial \mathbf{s}_t^{(l)}} = w^{(i,l)} \quad \text{for all } t \tag{A.6}$$

Equation A.3 is valid for $0 \leq t \leq T-1$

$$\frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(l)}} = -e_t^{(i)} + \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(l)}} \sigma(\mathbf{s}_t^{(l)}) w^{(l,i)} \tag{A.7}$$

Furthermore, as the neuronal activations at times $t \geq T+1$ have no impact on the cost

function $C_T(\mathbf{w})$

$$\frac{\partial^+ C_T(\mathbf{w})}{\mathbf{s}_{T+1}^{(l)}} = 0 \tag{A.8}$$

A recursive equation can be obtained

$$\frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(l)}} = \begin{cases} -e_t^{(i)} & \text{for } t = T \\ -e_t^{(i)} + \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_{t+1}^{(l)}} \sigma(\mathbf{s}_t^{(l)}) w^{(l,i)} & \text{otherwise} \end{cases} \tag{A.9}$$

which can be used in Equation A.3 to provide the required weight update.

To simplify notation, the following shorthand notations are introduced:

$$\epsilon_t^{(i)} = \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \tag{A.10}$$

$$\delta_t^{(i)} = \frac{\partial^+ C_T(\mathbf{w})}{\partial \acute{\mathbf{s}}_t^{(i)}} \tag{A.11}$$

which results in

$$\epsilon_t^{(i)} = \begin{cases} e_t^{(i)} & \text{for } t = T \\ e_t^{(i)} + \sum_{l=1}^{T} \delta_{t+1}^{(l)} \mathbf{w}^{(l,i)} & t < T \end{cases} \tag{A.12}$$

Given the definition of the gradient of the cost function, found in Equation A.1, the last line of Equation A.2, and the following partial derivative

$$\frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial \mathbf{w}^{(i,j)}} = z_t(j) \tag{A.13}$$

where the function $z_t(j)$ is defined in 2.25. The RNN weight update can be written as

$$\begin{aligned} \Delta w^{(i,j)} &= -\eta \sum_{t=0}^{T} \frac{\partial^+ C_T(\mathbf{w})}{\partial \mathbf{s}_t^{(i)}} \frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(i,j)}} \\ &= -\eta \sum_{t=0}^{T} -\delta_t^{(i)} z_t(j) = \eta \sum_{t=0}^{T} \delta_t^{(i)} z_t(j) \end{aligned} \tag{A.14}$$

Appendix B

Derivation of the RTRL Algorithm with Ordered Derivatives

The method of ordered derivatives can also be used to derive the RTRL algorithm [184]. As in the previous BPTT derivation, the derivation starts from the weight update rule

$$\Delta w^{(k,l)} = -\eta \frac{\partial^+ C_T(\mathbf{w})}{\partial w^{(k,l)}} \tag{B.1}$$

The gradient of the cost function represents the sensitivities of all weights $\mathbf{w}$ on the cost function. The weight $w^{(k,l)}$ of the kth neuron influences the cost function indirectly through the activations $\mathbf{s}_t^{(i)}$ of all neurons $(i = 1, \ldots, H)$. The second chain rule for ordered derivatives can be invoked as follows:

$$\begin{aligned} \frac{\partial^+ C_T(\mathbf{w})}{\partial w^{(k,l)}} &= \frac{\partial C_T(\mathbf{w})}{\partial \mathbf{w}^{(k,l)}} + \sum_{i=L+1}^{L+H} \frac{\partial}{\partial \mathbf{s}_t^{(i)}} E_t(\mathbf{w}) \frac{\partial^+ \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} \\ &= 0 + \sum_{i=L+1}^{L+H} \frac{\partial}{\partial \mathbf{s}_t^{(i)}} E_t(\mathbf{w}) \frac{\partial^+ \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} \\ &= \sum_{i=L+1}^{L+H} -e_t^{(i)} \frac{\partial^+ \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} \end{aligned} \tag{B.2}$$

For the partial derivative $\partial^+ \mathbf{s}_t^{(i)} / \partial w^{(k,l)}$, the second chain rule applies:

$$\begin{aligned} \frac{\partial^+ \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} &= \frac{\partial \mathbf{s}_t^{(i)}}{\partial w^{(k,l)}} + \frac{\partial \mathbf{s}_t^{(i)}}{\partial \acute{\mathbf{s}}_t^{(i)}} \frac{\partial^+ \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(i,j)}} = 0 + \frac{\partial \mathbf{s}_t^{(i)}}{\acute{\mathbf{s}}_t^{(i)}} \frac{\partial^+ \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} \\ &= \sigma'(\acute{\mathbf{s}}_t^{(i)}) \frac{\partial^+ \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} \end{aligned} \tag{B.3}$$

For the last partial derivative in the above equation $\partial^+ \acute{\mathbf{s}}_t^{(i)} / \partial w^{(k,l)}$ the summation block $\acute{\mathbf{s}}_t^{(i)}$ has direct dependence on the weight vector $w^{(k,l)}$ and indirect dependence on the inputs $z_t(j)$ to the network. The second chain rule applies here as well

$$\begin{aligned} \frac{\partial^+ \acute{\mathbf{s}}_t^{(i)}}{w^{(k,l)}} &= \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} + \sum_{j=1}^{L+H} \frac{\partial \mathbf{s}_t^{(i)}}{\partial z_t(j)} \frac{\partial^+ z_t(j)}{\partial w^{(k,l)}} \\ &= \delta_{i,k} z_t(l) + \sum_{j=1}^{L} w^{(i,j)} \frac{\partial^+ \mathbf{u}_t^{(j)}}{\partial w^{(k,l)}} + \sum_{j=L+1}^{L+H} w^{(i,j)} \frac{\partial^+ \mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}} \\ &= \delta_{i,k} z_t(l) + \sum_{j=L+1}^{L+H} w^{(i,j)} \frac{\partial^+ \mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}} \end{aligned} \tag{B.4}$$

where $\delta^{(i,k)}$ in the above equation is the Kroneker delta.

The change in the summation block with respect to the weights is equal to the following

$$\frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} = \delta^{(i,k)} z_t(l) \tag{B.5}$$

this is the main difference between the derivations of the RTRL algorithm (i.e. the ordered derivatives vs the traditional RTRL derivation). The traditional method for derivation of RTRL is rewritten below.

$$\begin{aligned} \frac{\partial \acute{\mathbf{s}}_t^{(i)}}{\partial w^{(k,l)}} &= \frac{\partial \left(\sum_{j=1}^{L+H} w^{(i,j)} z_t(j) \right)}{\partial w^{(k,l)}} \\ &= \sum_{j=1}^{L+H} \left[w^{(i,j)} \frac{\partial z_t(j)}{\partial w^{(k,l)}} + z_t(j) \frac{\partial w^{(i,j)}}{\partial w^{(k,l)}} \right] \\ &= \sum_{j=L+1}^{L+H} \left[w^{(i,j)} \frac{\partial \mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}} \right] + \delta_{i,k} z_t(l) \end{aligned} \tag{B.6}$$

Substituting Equation B.5 into Equation B.3 results in a recursive relation for computing the ordered partial derivatives $\partial^{+}\mathbf{s}_t^{(i)}/w^{(k,l)}$

$$\frac{\partial^{+}\mathbf{s}_t^{(i)}}{w^{(k,l)}} = \sigma'(\acute{\mathbf{s}}_t^{(i)})\left[\delta^{(i,k)} z_t(l) + \sum_{j=L+1}^{L+H} w^{(i,j)} \frac{\partial^{+}\mathbf{c}_t^{(j-L)}}{\partial w^{(k,l)}}\right] \tag{B.7}$$

Through substitution of the results obtained in Equation B.7 into the definition of the gradient of the cost function with respect to the weights Equation B.2 results in the training rule.

Appendix C

Derivation of Bayesian Levenberg-Marquardt Training Rule

The developed sequential Bayesian approach to RNN filtering considers the following objective error criterion (cost function) formulated according to the evidence framework:

$$C_t(\mathbf{w}) = \frac{1}{2}\Big(\sum_{\tau=1}^{t} \beta E_\tau(\mathbf{w}) + \mathbf{w}^{\mathrm{T}}(\alpha_i \mathbf{R}_\tau)\mathbf{w}\Big) \tag{C.1}$$

$$E_\tau(\mathbf{w}) = \Big(d_\tau - F(\mathbf{w}, \mathbf{u}_\tau)\Big)^2 \tag{C.2}$$

Treating the learning process as an optimization task suggests to consider the Taylor expansion of this cost function in the vicinity of the current weight vector $\hat{\mathbf{w}}_{t-1}$ estimated at time instant $t-1$. Using the notation $\mathbf{g}_t(\mathbf{w}_{t-1}) = \partial C_t(\mathbf{w}_{t-1})/\partial \mathbf{w}_{t-1}$ to denote the gradient, and $\mathbf{H}_t(\mathbf{w}_{t-1}) = \partial C_t^2(\mathbf{w}_{t-1})/(\partial \mathbf{w}_{t-1}\partial \mathbf{w}_{t-1})$ to denote the Hessian, the expansion can be written as follows:

$$C_t(\mathbf{w}) \approx C_t(\mathbf{w}_{t-1}) + \mathbf{g}_t(\mathbf{w}_{t-1})\delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\mathbf{H}_t(\mathbf{w}_{t-1})\delta\mathbf{w} \tag{C.3}$$

where $\delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{t-1}$.

This equation is solved for the weight vector, so as to reach the minimum of the cost function, by expanding the gradient. The minimization of the Taylor expansion leads to

the well known Newton's training rule [115, 152]:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{g}_t(\mathbf{w}_{t-1}) \tag{C.4}$$

which in this recursive form means that one complete iteration of Newton's method is performed at each next time step with the arrival of a new data point.

The intention here is to perform sequential Bayesian training via minimization of the cost function. The derivative of the Bayesian cost function is taken, and terms are rearranged as follows [137]:

$$\begin{aligned}\mathbf{g}_t(\mathbf{w}) = \frac{\partial C_t(\mathbf{w})}{\partial \mathbf{w}} &= -\sum_{\tau=1}^{t} \beta e_\tau(\mathbf{w})\mathbf{j}_\tau(\mathbf{w}) + \mathbf{w}\alpha_i\mathbf{R}_t \\ &= -\sum_{\tau=1}^{t-1} \beta e_\tau(\mathbf{w})\mathbf{j}_\tau(\mathbf{w}) - \beta e_t(\mathbf{w})\mathbf{j}_t(\mathbf{w}) + \mathbf{w}\alpha_i\mathbf{R}_t \\ &= \mathbf{g}_{t-1}(\mathbf{w}) - \beta e_t(\mathbf{w})\mathbf{j}_t(\mathbf{w}) + \mathbf{w}\alpha_i\mathbf{R}_t\end{aligned} \tag{C.5}$$

however, the gradient $\mathbf{g}_{t-1}(\mathbf{w}) = 0$ and thus

$$\mathbf{g}_t(\mathbf{w}) = -\beta e_t(\mathbf{w})\mathbf{j}_t(\mathbf{w}) + \alpha_i\mathbf{w}\mathbf{R}_t \tag{C.6}$$

After substituting into Equation (C.6) into Equation (C.4) and rearranging terms, the training equation becomes:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \beta\mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{j}_t(\mathbf{w}_{t-1})e_t(\mathbf{w}_{t-1}) - \alpha_i\mathbf{H}_t^{-1}(\mathbf{w}_{t-1})\mathbf{R}_t\mathbf{w}_{t-1} \tag{C.7}$$

where the second term is the incremental update, and the third term is the regularizer.

According to the specific definition of the matrix $\mathbf{R}_t$, the amount of regularization is partially distributed during training until reaching the boundary condition $m*(T \ div \ m)$. Regularization is applied one element at a time, that is partial regularization is carried out at every moment in time so as to distribute the total regularization over time. Bayesian regularization is applied when updating the Hessian matrix as follows [137]:

$$\mathbf{H}_t(\mathbf{w}_{t-1}) = \mathbf{H}_{t-1}(\mathbf{w}_{t-1}) + \beta\mathbf{j}_t(\mathbf{w}_{t-1})\mathbf{j}_t^{\mathrm{T}}(\mathbf{w}_{t-1}) + \alpha_i\mathbf{R}_t \tag{C.8}$$

by using $[\mathbf{R}_t]_{ii} = 1.0/z_i, i = (t+1) \ mod \ m$ if $(t) \leq (N \ div \ m)$ or $z_i = 1.0^{10}$ otherwise. Key to the derivation is to apply the matrix inversion lemma for sequential inversion of the regularized Hessian:

$$\mathbf{H}_t(\mathbf{w}_{t-1}) = \mathbf{H}_{t-1}(\mathbf{w}_{t-1}) + \beta \mathbf{j}_t^*(\mathbf{w}_{t-1}) \mathbf{\Lambda}_t^{-1} \mathbf{j}_t^{*\mathrm{T}}(\mathbf{w}_{t-1}) \tag{C.9}$$

where $\mathbf{\Lambda}_t^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & \alpha_i [\mathbf{R}_t]_{ii} \end{pmatrix}$ is the regularization matrix, and $\mathbf{j}_t^*(\mathbf{w}_{t-1})$ consists of a column of the derivative vector $\mathbf{j}_t^{\mathrm{T}}(\mathbf{w}_{t-1})$ augmented by a column with zero elements, except for the element at position $i = t \ mod \ m$ as follows [115]:

$$\mathbf{j}_t^*(\mathbf{w}_{t-1}) = \begin{pmatrix} \mathbf{j}_t^{\mathrm{T}}(\mathbf{w}_{t-1}) \\ 0 \ldots 1_i \ldots 0 \end{pmatrix} \tag{C.10}$$

When we apply the matrix inversion lemma [96]: $(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}[\mathbf{DA}^{-1}\mathbf{B}+\mathbf{C}^{-1}]^{-1}\mathbf{DA}^{-1}$ to the above expression we arrive at:

$$\left(\mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) + \beta \mathbf{j}_t^*(\mathbf{w}_{t-1}) \mathbf{\Lambda}_t^{-1} \mathbf{j}_t^{*\mathrm{T}}(\mathbf{w}_{t-1})\right)^{-1} = \mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) - \beta \mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) \mathbf{j}_t^*(\mathbf{w}_{t-1}) \mathbf{S}_{t-1}^{-1} \mathbf{j}_t^{*\mathrm{T}} \mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) \tag{C.11}$$

where

$$\mathbf{S}_{t-1} = \left(\beta \mathbf{j}_t^{*\mathrm{T}}(\mathbf{w}_{t-1}) \mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) \mathbf{j}_t^{*\mathrm{T}} + \mathbf{\Lambda}_t\right) \tag{C.12}$$

which is the recursive update formula for computing the inverted Hessian

$$\mathbf{H}_t^{-1}(\mathbf{w}_{t-1}) = \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) - \beta \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) \mathbf{j}_t^*(\mathbf{w}_{t-1}) \mathbf{S}_{t-1}^{-1} \mathbf{j}_t^*(\mathbf{w}_{t-1}) + \mathbf{\Lambda}_t)^{-1} \mathbf{j}_t^{*\mathrm{T}}(\mathbf{w}_{t-1}) \mathbf{H}_{t-1}^{-1}(\mathbf{w}_{t-1}) \tag{C.13}$$

It should be noted that the denominator of the second term $\mathbf{S}_{t-1}$ is only a two dimensional square matrix which is inverted with analytical formula [96].

Bibliography

[1] H. D. I. ABARBANEL. *Analysis of Observed Chaotic Data*. Springer, New York, 1996.

[2] H. D. I. ABARBANEL, R. BROWN, AND J. B. KADTKE. Prediction in chaotic nonlinear systems: Methods for time series with broadband fourier spectra. *Phys. Rev. A*, **41(4)**:1782–1807, 1990.

[3] H. D. I. ABARBANEL AND U. LALL. Nonlinear dynamics of the great salt lake: system identification and prediction. *Clim. Dyn.*, **12**:287–297, 1996.

[4] S. K. AGGARWAL, L. M. SANI, AND A. KUMAR. Electricity price forecasting in ontario electricity market using wavelet transform in artificial neural network based model. *International Journal of Control, Automation, and Systems*, **6**(5):639–650, 2008.

[5] A. ALBANO, A. PASSAMANTE, T. HEDIGER, AND M. E. FARRELL. Using neural networks to look for chaos. *Physica D*, **58**:1–9, 1992.

[6] E. AMATA, G. PALLOCCHIA, G. CONSOLINI, M. F. MARCUCCI, AND I. BERTELLO. Comparison between three algorithms for d_{st} predictions over the 2003-2005 period. *Journal of Atmospheric and Solar-Terrestrial Physics*, **70**:496–502, 2008.

[7] N. AMJADY. Day-ahead price forecasting of electricity markets by a fuzzy neural network. *IEEE Trans. Power. Syst.*, **21**(2), 2006.

[8] J. A. ANDERSON AND E. ROSENFELD, editors. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, 1988.

[9] I. ARASARATNAM AND S. HAYKIN. Nonlinear bayesian filters for training recurrent neural networks. In *Proceedings of the 7th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence*, pages 12–33. Springer-Verlag, 2008.

[10] M. A. ARBIB. *Brains, Machines, and Mathematics*. Springer–Verlag, New York, NY, 2nd edition, 1987.

[11] S. ARULAMPALAM, S. MASKELL, N. GORDON, AND T. CLAPP. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, **50**(2):174–188, 2002.

[12] Y. BAR-SHALOM AND X. R. LI. *Estimation and Tracking: Principles, Techniques and Software*. Artech House, Boston, 1993.

[13] M. BASK, T. LIU, AND A. WIDERBERG. The stability of electricity prices: Estimation and inference of the lyapunov exponents. *Physica A*, **376**:565–572, 2007.

[14] Y. BENGIO, P. SIMARD, AND P. FRASCONI. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, **5**(2):157–166, March 1994.

[15] C. M. BISHOP. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[16] C. M. BISHOP. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, **7**(1):108–116, 1995.

[17] W. L. BUNTINE AND A. S. WEIGEND. Bayesian back-propagation. *Complex Systems*, **5**:603–643, 1991.

[18] P. CAMPOLUCCI, M. UNCIN, AND F. PIAZZA. New second-order algorithms for recurrent neural networks based on conjugate gradient. In *Proc. Int. Joint Conf. Neural Networks*, **1**, pages 384–389. IEEE, 1998.

[19] M. CASDAGLI. Nonlinear prediction of chaotic time series. *Physica D*, **35**:335–356, 1989.

[20] M. C. CASDAGLI, S. E. EUBANK, J. D. FARMER, AND J. GIBSON. State space reconstruction in the presence of noise. *Physica D*, **51**:52–98, 1991.

[21] M. CASEY. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Comput.*, **8**(6):1135–1178, 1996.

[22] J. P. S. CATALAO, S. J. P. S. MARIANO, V. M. F. MENDES, AND L. A. F. M. FERREIRA. Short-term electricity prices forecasting in a competitive market: A neural network approach. *Electric Power Systems Research*, **77**(10):1297–1304, 2007.

[23] M. CERNANSKY AND L. BENUSKOVA. Simple recurrent network trained by rtrl and extended kalman filter algorithms. *Neural Network World*, **13**(3):223–234, 2003.

[24] L. W. CHAN AND C. C. SZETO. Training recurrent network with block diagonal approximated levenberg marquardt algorithm. In *Proc. Int. Joint Conf. Neural Networks*, **2**, pages 1521–1526, 1999.

[25] B. CHENG AND D. M. TITTERINGTON. Neural networks: A review from a statistical perspective. *Statistical Science*, **9**(1):2–30, 1994.

[26] J. CHOI, A .C. LIMA, AND S. HAYKIN. Kalman filter-trained recurrent neural equalizers for time-varying channels. *IEEE Trans on Communications*, **53**(3):472–480, 2005.

[27] J. CHOI, T. H. YEAP, AND M. BOUCHARD. Online statespace modeling using recurrent multilayer perceptrons with unscented kalman filter. *Neural Processing Letters*, **22**(1):69–84, 2005.

[28] P. H. G. COELHO. An extended rtrl training algorithm using hessian matrix. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*, **4**, pages 563–568. IEEE.

[29] A. J. CONEJO, M. A. PLAZAS, R. ESPINOLA, AND A. B. MOLINA. Day-ahead electricity price forecasting using the wavelet transform and arima models. *IEEE Trans. Power Syst*, **20**(2):1035–1042, 2005.

[30] J. CONNOR AND L. ATLAS. Recurrent neural networks and time series prediction. In *Proceedings of the International Joint Conference on Neural Networks 1991*, pages 301–306, 1991.

[31] J. P. CRUTCHFIELD AND B. S. MCNAMARA. Equations of motion from a data series. *Complex Systems*, **1**:417–452, 1987.

[32] G. CYBENKO. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, **2**:303–314, 1989.

[33] J. F. G. DE FREITAS, M. NIRANJAN, AND A. H. GEE. Dynamic learning with the *em* algorithm for neural networks. *J. Vlsi. Signal. Proc.*, **26**:119–131, 2000.

[34] L. M. DE MENEZES, D. BUNN, AND J. TAYLOR. Review of practical guidelines for combining forecasts. *European Journal of Operational Research*, **120**:190–204, 2000.

[35] L. M. DE MENEZES AND N. NIKOLAEV. On univariate time series forecasting of electricity prices and similar data. In *ISF2008 - International Symposium on Forecasting*, 2008.

[36] L. M. DE MENEZES, N. NIKOLAEV, AND H. IBA. Overfitting avoidance in genetic programming of polynomials. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)*. IEEE Press, 2002.

[37] G. DECO AND B. SCHURMANN. Neural learning of chaotic dynamics. *Neural Processing Lett.*, **2**(2):23–26, 1995.

[38] A. P. DEMPSTER, N. M. LAIRD, AND D. B. RUBIN. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc. B*, **39**:1–38, 1977.

[39] A. M. DENISOV. *Elements of the Theory of Inverse Problems*. VSP, Utrecht, The Netherlands, 1999.

[40] J. E. DENNIS AND R. B. SCHNABEL. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[41] K. DOYA. Universality of fully-connected recurrent neural networks. Technical report, University of California, 1993.

[42] K. DOYA AND S. YOSHIZAWA. Adaptive neural oscillator using continuous time back-propagation learning.

[43] R. DUDA AND P. HART. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York.

[44] J. L. ELMAN. Finding structure in time. *Cognitive Science*, **14**:179–211, 1990.

[45] N. EULIANO AND J. PRINCIPE. Dynamic subgrouping in rtrl provides a faster $o(n^2)$ algorithm. In *Proc. IEEE Int Conf. Acoust. Speech Signal Process.*, **6**, pages 3418–3421, Istanbul, 2000.

[46] G. EVENSEN. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *J. Geophysics. Res.*, **99**:10143–10162.

[47] G. EVENSEN. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean Dynamics*, **53**:343–367, 2003.

[48] G. EVENSEN. *Data assimilation: The ensemble Kalman filter*. Berlin, Springer, 2007.

[49] J. D. FARMER. A rosetta stone for connectionism. *Physica D*, **42**(1–3):153–187, 1990.

[50] L. V. FAUSETT. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, New Jersey, 1994.

[51] L. A. FELDKAMP, T. M. FELDKAMP, AND D. V. PROKHOROV. Recurrent neural network training by nprkf joint estimation. In *in Proc. Int. Joint Conf. Neural Networks.*, pages 2086–2091.

[52] R. D. FIERRO, G. H. GOLUB, P. C. HANSEN, AND D. P. OLEARY. Regularization by truncated total least squares. *SIAM Journal on Scientific Computing*, **18**(1):1223–1241, 1997.

[53] W. FINNOF, F. HERGERT, AND H. G. ZIMMERMANN. Improving model selection by nonconvergent methods. *Neural Networks*, **6**:771–783, 1993.

[54] A. FRASCONI, M. GORI, AND G. SODA. Local feedback multi-layered networks. *Neural Computation*, **4**(1):120–130, 1992.

[55] K. FUNAHASHI. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, **2**:183–192, 1989.

[56] R. GELB. *Applied optimal estimation*. MIT Press, 1974.

[57] S. GEMAN, E. BIENENSTOCK, AND R. DOURSAT. Neural networks and the bias / variance dilemma. *Neural Computation*, **4**:1–58, 1992.

[58] R. GENCAY AND T. LIU. Nonlinear modeling and prediction with feed-forward and recurrent networks. *Physica D*, **108**(1):119–134, 1997.

[59] N. A. GERSHENFELD. An experimentalist's introduction to the observation of dynamical systems. In B. L. HAO, editor, *Directions in Chaos*, **2**, pages 310–384. World Scientific, Singapore, 1989.

[60] N. A. GERSHENFELD AND A. S. WEIGEND. The future of time series: Learning and understanding. In A. S. WEIGEND AND N. A. GERSHENFELD, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pages 1–70, Reading, MA, 1994. Addison-Wesley.

[61] Z. GHAHRAMANI AND G.E. HINTON. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Toronto, 1996.

[62] C. L. GILES AND C. W. PMLIN. Pruning recurrent neural networks for improved generalization performance. *IEEE Trans. Neural Networks*, **5**(108), 1994.

[63] P. E. GILL, T. LIN, AND B. G. HORNE. *Practicle Optimization*. Academic Press, London, 1981.

[64] FEDERICO GIROSI, MICHAEL JONES, AND TOMASO POGGIO. Regularization theory and neural networks architectures. *Neural Computation*, **7**(2):219–269, 1995.

[65] D. E. GOLDBERG. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[66] G. GOLUB AND C. VAN LOAN. *Matrix Computations*. John Hopkins University Press, third edition edition, 1996.

[67] A. M. GONZALEZ, A. M. SAN ROQUE, AND J. G. CONZALES. Modeling and forecasting electricity prices with input/output hidden markov models. **20**(1):13–24, 2005.

[68] Y. GRANDVALET AND S. CANU. Outcomes of the equivalence of adaptive ridge with least absolute shrinkage. In *Advances in Neural Information Processing Systems 11*, pages 445–451. MIT Press, 1999.

[69] P. GRASSBERGER AND I. PROCACCIA. Characterization of strange attractors. *Phys. Rev. Lett.*, **50**:346–349, 1983.

[70] N. K. GUPTA AND R. K. MEHRA. Computational aspects of maximum likelihood estimation and reduction in sensitivity funciton calculations. *IEEE Transactions on Automatic Control*, **6**:774–783, 1974.

[71] B. G. HORNE H. T. SIEGELMANN AND C. L. GILES. Computational capabilities of recurrent narx neural networks. *IEEE Trans on Systems, Man, and Cybernetics*, **27**(2):208–215.

[72] J. HADAMARD. Sur les problems aux derivees partielies et leur signification physique. *Bull. Univ. Princeton*, **13**:49–52, 1902.

[73] J. D. HAMILTON. *Time Series Analysis*. Princeton University Press, Princeton NJ, 1994.

[74] H. HARTLEY. Maximum likelihood estimation from incomplete data. *Biometrics*, **14**:174–194, 1958.

[75] A. C. HARVEY. *Forecasting Structural Time Series Models and the Kalman Filter*. Cambridge University Press, UK, 1989.

[76] A. C. HARVEY AND G. D. A. PHILLIPS. Maximum likelihood estimation of regression models with autoregressive-moving average disturbances. *Biometrika*, **66**:49–58, 1979.

[77] S. HAYKIN. *Neural Networks*. Macmillan, Canada, 1st edition, 1994.

[78] S. HAYKIN. *Kalman Filtering and Neural Networks*. John Wiley & son, New York, 2001.

[79] S. HAYKIN AND J. PRINCIPE. Making sense of a complex world. *IEEE Signal Processing Magazine*, **15**(3):66–81, 1998.

[80] D. O. HEBB. *The Organization of Behavior: A Neurophysiological Theory*. Wiley, New York, 1949.

[81] M. HENON. A two-dimensional mapping with a strange attractor. *Comm. Math. Phys.*, **50**:69–77, 1976.

[82] Y. C. HO AND R. C. K. LEE. A bayesian approach to problems in stochastic estimation and control. *IEEE Transactions on Automatic Control*, pages 333– 339, 1964.

[83] A. E. HOERL AND R. W. KENNARD. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, **12**(3):55–67, 1970.

[84] J. H. HOLLAND. *Hidden order: how adaptation builds complexity.* Addison Wesley Longman Publishing Co., Redwood City, CA, USA, 1996.

[85] J. J. HOPFIELD AND D. W. TANK. Neural computation of decisions in optimization problems. *Biological Cybernetics*, **52**:144–152, 1985.

[86] K. HORNIK, M. STINCHCOMBE, AND H. WHITE. Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**:359–366, 1989.

[87] X. HU, D. V. PROKHOROV, AND D C. WUNSCH. Time series prediction with a weighted bidirectional multi-stream extended kalman filter. *Neurocomputing*, **70**(13-15):2392 – 2399, 2007.

[88] U. HÜEBNER, N. ABRAHAM, AND C. WEISS. Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared nh3 laser. *Phys. Rev. A.*, **40**:6354–6360, 1989.

[89] H. HYOTYNIEMI. Turing machines are recurrent neural networks. In *Proceeding of STEP96, Finish Artificial Intelligence Society*, pages 13–24, Vaasa, Finland, 1996.

[90] K. IKEDA. Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system. *Opt. Commun.*, **30**:257–263, 1979.

[91] R. A. JACOBS. Methods for combining experts probability assessments. *Neural Computation*, **7**(5):867–888, 1995.

[92] A. H. JAZWINSKI. Adaptive filtering. *Automatica*, **5**:475–485, 1969.

[93] A. H. JAZWINSKI. *Stochastic Process and Filtering Theory.* Academic Press, New York, 1970.

[94] M. I. JORDAN. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, Englewood Cliffs, NJ: Erlbaum, 1986.

[95] C. L. GILES K. C. JIM AND G. G. HORNE. An analysis of noise in recurrent neural networks: Convergence and generalization. *IEEE Trans. Neural Networks*, **7**(6):1424–1438, 1996.

[96] J. T. KENT K. V. MARDIA AND J. M. BIBBY. *Multivariate Analysis, Probability and Mathematical Staticstics*. Academic Press, London, 1979.

[97] E. KALNAY. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, Cambridge, UK, 2003.

[98] H. KANTZ AND T. SCHREIBER. *Nonlinear Time Series Analysis*. Cambridge University Press, Cambridge, UK, 1997.

[99] F. KIANIFARD AND W. SWALLOW. A review of the development and application of recursive residuals in linear models. *Journal of the American Statistical Association*, **91**(433):391–400, 1996.

[100] L. KINDERMANN AND T. P. TRAPPENBERG. Modeling time-varying processes by unfolding the time domain. In *International Joint Conference on Neural Networks (IJCNN'99)*, Washington DC, 1999.

[101] S. KIRKPATRICK, C. D. GELATT, AND M. P. VECCHI. Optimization by simulated annealing. *Science*, **New Series 220**(4598):671–680, 1983.

[102] T. KOSKELA, M. LEHTOKANGAS, J. SAARINEN, AND K. KASKI. Time series prediction with multilayer perceptron, fir, and elman neural networks. In *Proc. WCNN.*, pages 491–496. INNS Press, 1996.

[103] A. KROK AND Z. WASZCZCYZYN. Kalman filtering for neural prediction of response spectra form mining tremors. *Comput. Struct.*, **85**:1257–1263, 2007.

[104] C. M. KUAN, K. HORNIK, AND T. LIU. Recurrent back-propagation and newton algorithms for training recurrent networks. In J.L.FLANAGAN ET AL., editor, *Substance Identification Analytics*, **2093**, pages 220–229. SPIE Press, Bellingham, WA, 1994.

[105] J. M. KUO. *Nonlinear Dynamic Modeling with Artificial Neural Networks.* PhD thesis, University of Florida, 1993.

[106] A. LAPEDES AND R. FARBER. Nonlinear signal processing using neural networks. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[107] S. L. LARUITZEN. Time series analysis in 1880: A discussion of the contributions made by *t. n. thiele. International Statistical Review*, **49**:319–331, 1981.

[108] J. LATIMER. *Cumulant Filters: Nonlinear Filters for Recursive State Estimation of Systems with Non-Gaussian Noise.* PhD thesis, Department of Electrical Engineering and Computer Science, The Catholic University of America, 2004.

[109] D. J. LEE. *Nonlinear Bayesian Filtering with Applicatoins to Estimation and Navigation.* PhD thesis, Department of Aerospace Engineering, Texas A&M University, 2005.

[110] T. LEEN. From data distributions to regularization in invariant learning. *Neural Computation*, **7**(5):974–981, 1995.

[111] C. S. LEUNG AND P. M. LAM. A local training-pruning approach for recurrent neural networks. *International Journal of Neural Systems*, **13**(1):25–38, 2003.

[112] K. LEVENBERG. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, **2**:164–168, 1944.

[113] W. LIU, L. L. YANG, AND L. HANZO. Recurrent neural network based narrowband channel prediction. In *Proc. Vehicular Technology Conference*, **5**, pages 2173–2177.

[114] L. LJUNG. *System Identification. Theory for the User.* Prentice Hall, New Jersey, 1987.

[115] L. LJUNG AND T. SÖDESTROM. *Theory and practice of recursive identification.* MIT Press, Cambridge MA, 1983.

[116] L. LJUNG AND B. WAHLBERG. Asymptotic properties of the least-squares method for estimating transfer functions and disturbance spectra. *Advances in Applied Probability*, **24**:412–440, 1992.

[117] E. N. LORENZ. Deterministic non-periodic flow. *J. Atmos. Sci.* , **20**:130–141, 1963.

[118] H. LUNDSTEDT. Neural networks and prediction of solar-terrestrial effects. *Planet. Space Sci.*, **40**:457–464.

[119] H. LUNDSTEDT AND P. WINTOFT. Prediction of geomagnetic storms from solar wind data with the use of a neural network. *Ann Geophys*, **12**:19–24, 1994.

[120] D. J. C. MACKAY. Bayesian interpolation. *Neural Computation*, **4**(3):415–447, 1992.

[121] D. J. C. MACKAY. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

[122] D. J. C. MACKAY. A practical bayesian framework for backpropagation networks. *Neural Computation*, **4**:448–472, 1992.

[123] S. MANDAL AND M. PRABAHARAN. Ocean wave forecasting using recurrent neural networks. *Ocean Engineering*, **33**:1401–1410, 2006.

[124] J. MANDEL. Personal communication.

[125] J. MANDEL, J. BEEZLEY, J. L. COEN, AND M. KIM. Data assimilation for wildland fires: Ensemble kalman filters in coupled atmosphere-surface models. *IEEE Control Systems Magazine*, **29**(3):47–65, 2009.

[126] W. L. MAO. Novel srekf-based recurrent neural predictor for narrowband/fm inference rejection in gps. *International Journal of Electronics and Communications*, **62**:216–222, 2008.

[127] D. MARQUARDT. An algorithm for least-squares estimation of non-linear parameters. *SIAM J. Appl. Math*, **11**:431–441, 1963.

[128] J. L. MCCLELLAND AND D. E. RUMELHART. *Parallel distributed processing: Explorations in the microstructure of cognition*. MIT Press, Cambridge, 1986.

[129] W. S. MCCULLOCH AND W. PITTS. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 1943.

[130] B. S. MCNAMARA. *Stochastic Resonance and Equations of Motion from a Time Series*. PhD thesis, Department of Physics. University of California, Santa Cruz, 1989.

[131] M. R. G. MEIRELES, P. E. M. ALMEIDA, AND M. G. SIMOES. A comprehensive review for industrial applicability of artificial neural netowrks. *IEEE Transactions on Industrial Electronics*, **50**(3):585–601, 2003.

[132] M. MINSKY AND S. PAPERT. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[133] D. T. MIRIKITANI AND N. NIKOLAEV. Recursive bayesian levenberg-marquardt training of recurrent neural networks. In *Proc. Int. Joint Conf. Neural Networks*, pages 1089–1098, 2007.

[134] D. T. MIRIKITANI AND N. NIKOLAEV. Dynamic modeling with ensemble kalman filter trained recurrent neural networks. In *ICMLA '08: Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications*, pages 843–848. IEEE Computer Society, 2008.

[135] D. T. MIRIKITANI AND N. NIKOLAEV. Recurrent expectation maximization neural modeling. In *Proceedings of the International Conferences on Computational Intelligence for Modelling, Control and Automation (CIMCA'08)*, pages 674–679. IEEE Computer Society, 2008.

[136] D. T. MIRIKITANI AND N. NIKOLAEV. Efficient online recurrent connectionist learning with the ensemble kalman filter. *Neurocomputing*, **73**(4–6):1024–1030, 2010.

[137] D. T. MIRIKITANI AND N. NIKOLAEV. Recursive bayesian recurrent neural networks for time-series modeling. *IEEE Transactions on Neural Networks*, **21**(2):262–274, 2010.

[138] D. T. MIRIKITANI AND N. NIKOLAEV. Nonlinear maximum likelihood estimation of electricity spot prices using recurrent neural networks. *Neural Computing and Applications*, DOI: 10.1007/s00521-010-0344-1, 2010.

[139] D. T. MIRIKITANI AND L. OUARBYA. Modeling dst with recurrent *em* neural networks. In *In the* 19^{th} *International Conference on Artificial Neural Networks (ICANN2009)*, pages 975–984. Springer-Verlag, 2009.

[140] D. T. MIRIKITANI, I. PARK, AND M. DAOUDI. Dynamic reconstruction from noise contaminated data with sparse bayesian recurrent neural networks. In *Proceedings of the First Asia International Conference on Modelling & Simulation (AMS'07)*, pages 409–414. IEEE Computer Society, 2007.

[141] H. L. MITCHELL. Personal communication.

[142] H. L. MITCHELL, P. L. HOUTEKAMER, AND G. PELLERIN. Ensemble size, balance, and model-error representation in an ensemble kalman filter. *Monthly Weather Review*, **130**(11):2791–2808.

[143] M. MITCHELL. *Complexity A Guided Tour*. Oxford University Press, New York, NY, 2009.

[144] M. MOLLER. *Efficient Training of Feed-Forward Neural Networks*. PhD thesis, Department of Computer Science, University of Aarhus, 2003.

[145] M. C. MOZER. Neural net architectures for temporal sequence processing. In A. S. WEIGEND AND N. A. GERSHENFELD, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pages 243–264, Reading, MA, 1994. Addison-Wesley.

[146] I.T. NABNEY. *Netlab: Algorithms for Pattern Recognition*. Springer Verlag, London, 2002.

[147] K. S. NARENDRA AND K. PARTHASARATHY. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, **1**:4–27, 1990.

[148] R. M. NEAL. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., USA, 1996.

[149] R. M. NEAL AND G. E. HINTON. A view of the *em* algorithm that justies incremental, sparse, and other variants. In M. I. JORDAN, editor, *Learning in Graphical Models*, pages 355–368. MIT Press, 1998.

[150] L. NERGER, W. HILLER, AND J. SCHRTER. A comparison of error subspace kalman filters. *Tellus A*, **57**(5):715–735, 2005.

[151] R. NEUNEIER AND H. G. ZIMMERMANN. How to train neural networks. In *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, pages 373–423. Springer, 1996.

[152] L. S. H. NGIA AND J. SJÖBERG. Efficient training of neural nets for nonlinear adaptive filtering using a recursive levenberg-marquardt algorithm. *IEEE Trans. on Signal Processing*, **48**(7):1915–1927, 2000.

[153] N.Y. NIKOLAEV AND H. IBA. *Adaptive Learning of Polynomial Networks.* Springer, New York, 2006.

[154] F. J. NOGALES, J. CONTRERAS, A. J. CONEJO, AND R. ESPINOLA. Forecasting next-day electricity prices by time-series models. *IEEE Trans. Power Syst.*, **17**(2):342–348, 2002.

[155] K. NOURI, R. DHAOUADI, AND N. B. BRAIEK. Adaptive control of a nonlinear *dc* motor drive using recurrent neural networks. *Applied Soft Computing*, **8**(1):371–382, 2008.

[156] L. OUARBYA AND D. T. MIRIKITANI. Modeling geomagnetospheric disturbances with sequential bayesian recurrent neural networks. In *In the* 16^{th} *International Conference on Neural Information Processing (ICONIP2009)*, pages 91–99. Springer-Verlag, 2009.

[157] L. OUARBYA AND D. T. MIRIKITANI. Sequential modeling of d_{st} dynamics with seek trained recurrent neural networks. In *Proceedings of the International Conference on Intelligent Systems, Modelling and Simulation (ISMS '10)*, pages 32–37. IEEE Computer Society, 2010.

[158] N. H. PACKARD, J. P CRUTCHFIELD, J. D. FARMER, AND R. S. SHAW. Geometry from a time series. *Physical Review Letters*, **45**:712–716, 1980.

[159] G. PALLOCCHIA, E. AMATA, G. CONSOLINI, M. F. MARCUCCI, AND I. BERTELLO. Geomagnetic dst index forecast based on imf data only. *Ann Geophys*, **24**:989–999, 2006.

[160] J. PARK AND I. W. SANDBERG. Universal approximation using radial-basis-function networks. *Neural Comput.*, **3**(2):246–257, 1991.

[161] M. W. PEDERSEN. *Optimization of Recurrent Neural Networks for Time Series Modeling*. PhD thesis, Department of Mathematical Modelling. Technical University of Denmark, 1997.

[162] M. W. PEDERSEN AND L. K. HANSEN. Recurrent netwroks: Second order properties and pruning. In D. TOURETZKY G. TESAURO AND T. LEEN, editors, *Advances in Neural Information Processing Systems*, pages 673–680. MIT Press, 1995.

[163] M. P. PERRONE. Averaging/modular techniques for neural networks. In R. J. MAMMONE, editor, *The Handbook of Brain Theory and Neural Netowrks*, pages 126–129. MIT Press, 1995.

[164] M. P. PERRONE AND L. N. COOPER. When networks disagree: Ensamble methods for hybird neural networks. In R. J. MAMMONE, editor, *Artificial Neural Networks for Speech and Vision*, pages 126–142. London, 1993.

[165] A. A. PETROSIAN, D. V. PROKHOROV, W. LAJARA-NANSON, AND R. B. SCHIFFER. Recurrent neural network-based approach for early recognition of alzheimer's disease in eeg. *Clinical Neurophysiology*, **112**(8):1378 – 1387, 2001.

[166] D. T. PHAM, J. VERNON, AND M. C. ROUBAUD. A singular evolutive extended kalman filter. *Journal of Marine Systems*, **16**(3, 4):323–340.

[167] F. PINEDA. Recurrent back-propagation and the dynamical approach to adaptive neural computation. *Neural Computation*, **1**:161–172, 1989.

[168] T. POGGIO AND F. GIROSI. Networks for approximation and learning. *Proceedings of the IEEE*, **78**(9):1481–1497, 1990.

[169] J. PRINCIPE, N. EULIANO, AND C. LEFEBVRE. *Neural and Adaptive Systems: Fundamentals through Simulations*. John Wiley, 2000.

[170] D. PROKHOROV. Training neurocontrollers for robustness with derivative-free kalman filter. *IEEE Trans. on Neural Networks*, **17**(6):1606–1616, 2006.

[171] D. V. PROKHOROV. Toyota prius hev neurocontrol and diagnostics. *Neural Networks*, **21**:458–465, 2008.

[172] G. V. PUSKORIUS AND L. A. FELDKAMP. Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Trans. Neural. Networks.* , **5**:279–297, 1994.

[173] G. V. PUSKORIUS, L. A. FELDKAMP, AND L. I. DAVIS. Dynamic neural network methods applied to on-vehicle idle speed control. *Proceedings of the IEEE*, **84**(10):1407–1420.

[174] H. E. RAUCH, F. TUNG, AND C.T. STRIEBEL. Maximum likelihood estimates of linear dynamic models. *AIAA Journal*, **3**(8):1445–1450, 1965.

[175] B. RISTIC, S. ARUMPALAMPAM, AND N. GORDON. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House Publishers, Artech House Radar Library, 2004.

[176] H. ROBINSON AND S. MUNRO. A stochastic approximation method. *Annals of Mathematical Statistics*, **22**(5):400–407, 1991.

[177] T. ROBINSON AND F. FALLSIDE. A recurrent error propagation network speech recognition system. *Computer Speech and Language*, **5–3**:259–274, 1991.

[178] C. P. RODRIGUEZ AND G. J. ANDERS. Energy price forecasting in the ontario competitive power system market. *IEEE Trans. Power Syst.*, **21**(2):887–896, 2006.

[179] F. ROSNBLATT. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, New York, 1962.

[180] O. E. RÖSSLER. An equation for hyperchaos. *Physics Letters*, **71**(A):155–157, 1979.

[181] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS. Learning internal representations by error propagation. In D. E. RUMELHART AND J. L. MCCLELLAND, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, **I: Foundations**, pages 318–362. MIT Press/Bradford Books, Cambridge, MA, 1986.

[182] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS. Learning representations by back-propagating errors. *Nature*, **323**:533–536, 1986.

[183] D. E. RUMELHART, J. L. MCCLELLAND, AND PDP RESEARCH GROUP. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA, 1986.

[184] S. SANTINI, A. DEL BIMBO, AND R. JAIN. Block-structured recurrent neural networks. *Neural Networks*, **8**(1):135–147, 1995.

[185] A. M. SCHAFER AND H. G. ZIMMERMANN. Recurrent neural networks are universal approximators. *Int. J. Neural. Syst.*, **7**(4):253–263, 2007.

[186] C. SCHITTENKOPF, P. TINO, AND G. DORFNER. The beneft of information reduction for trading strategies. *Applied Financial Economics*, **34**(7):917–930, 2002.

[187] F. H. SCHLEE, C. J. STANDISH, AND N. F. TODA. Divergence in the kalman filter. *AIAA Journal*, **5**:ll14–1120, 1967.

[188] J. SCHMIDHUBER. A fixed size storage $\mathcal{O}(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, **4**:243–340, 1992.

[189] B. SCHOTTKY AND D. SAAD. Statistical mechanics of ekf learning in neural networks. *J. Phys. A: Math. Gen.*, **32**:1605–1621, 1999.

[190] COMPUTER SCIENCE AND TELECOMMUNICATIONS BOARD. *Academic careers for experimental computer scientists and engineers*. National Academic Press, New York, NY, USA, 1994.

[191] D. R. SEIDL AND R. D. LORENZ. A structure by which a recurrent neural network can approximate a nonlinear dynamic system. In *Proceedings of the International Joint Conference on Neural Networks 1991*, pages 709–714, 1991.

[192] R. H. SHUMWAY AND D. S. STOFFER. An approach to time series smoothing and forecasting using the em algorithm. *J Time Ser Anal*, **3**(4):253–264.

[193] R. H. SHUMWAY AND D. S. STOFFER. *Time Series Analysis and Its Applications: With R Examples*. Springer, 2006.

[194] H. T. SIEGELMANN. Computation beyond the turing limit. *Science*, **238**(28):632–637, 1995.

[195] H. T. SIEGELMANN AND E. D. SONTAG. Turing computability with neural networks. *Applied Mathematics Letters*, **4**(6):77–80, 1991.

[196] D. M. SIMA. *Regularization Techniques in Model Fitting and Parameter Estimation*. PhD thesis, Faculteit Ingenieurswetenschappen, Department Elektrotechniek, Katholieke Universiteit Leuven, 2006.

[197] M. SMALL. *Applied Nonlinear Time Series Analysis: Applications in Physics, Physiology and Finance*. Nonlinear Science Series A, World Scientific, 2005.

[198] H. W. SORENSON. Recursive estimation for nonlienar dynamic systems. In J. C. SPALL, editor, *Bayesian Analysis of Time Series and Dynamic Models*, pages 127–165. Marcel Dekker, New York, 1988.

[199] P. STAGGE AND B. SENDHOFF. An extended elman net for modeling time series. In *International Conference on Artificial Neural Networks ICANN'97*, pages 427–432. Lecture Notes in Computer Science, Springer, 1997.

[200] J. SUM AND L. W. CHAN. On the approximation property of recurrent neural network. In *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics, Caracas, Venezuela*, 1997.

[201] J. SUM, L. W. CHAN, A. C. S. LEUNG, AND G. H. YOUNG. Extended kalman filter-based pruning method for recurrent neural networks. *Neural Computation*, **10**(6):1481–1505, 1998.

[202] G. Z. SUN, H. H. CHEN, AND Y. C. LEE. Green's function method for fast online learning algorithm of recurrent neural networks. In *in Proc. Neural Inf. Process. Syst*, pages 333–340, 1991.

[203] F. TAKENS. Detecting strange attractors in turbulence (lecture notes). In D. A. RAND AND L. S. YOUNG, editors, *Mathematics*, **898**, pages 336–381. Springer-Verlag, Berlin, 1980.

[204] J. W. TAYLOR, L. M. DE MENEZES, AND P. E. MCSHARRY. A comparison of univariate methods for forecasting electricity demand up to a day ahead. *International Journal of Forecasting*, **22**(1):1–16.

[205] T. N. THIELE. *Sur la compensation de quelques erreurs quasi-systematiques par la methode des mondres carrees*. Reitzel, Kopenhagen.

[206] L. TIAN AND A. NOORE. Software reliability prediction using recurrent neural network with bayesian regularization. *Int. J. Neural systems*, **14**(3):165–174, 2004.

[207] A. N. TIKHONOV. Solution of incorrectly formulated problems and regularization method. *Solviet Math. Dokl 4*, pages 1035–1038, 1963.

[208] A. N. TIKHONOV AND V. Y. ARSENIN. *Solutions of Ill-Posed Problems*. Wiley, New York, 1977.

[209] P. TINO AND B. HAMMER. Architectural bias in recurrent neural networks: Fractal analysis. *Neural Computation*, **15**(8):1931–1957, 2002.

[210] P. TINO, B. G. HORNE, C. L. GILES, AND P. C. COLLINGWOOD. Finite state machines and recurrent neural networks – automata and dynamical systems approaches. In *Neural Networks and Pattern Recognition*, pages 171–220. Academic Press, 1998.

[211] P. TINO, N. NIKOLAEV, AND X. YAO. Volatility forecasting with sparse bayesian kernel models. In *Proceedings of 8th Joint Conference on Information Sciences 2005 (4th International Conference on Computational Intelligence in Economics and Finance)*, pages 1150–1153, Salt Lake City, UT, 2005.

[212] M. E. TIPPING. Bayesian inference: An introduction to principles and practice in machine learning. In *Lectures in Machine Learning*, pages 41–62. Springer.

[213] A.C. TSOI AND A.D. BACK. Locally recurrent globally feedforward networks: A critical review of architectures. *IEEE Transactions on Neural Networks*, **5**(2):229–239, 1994.

[214] A.C. TSOI AND A.D. BACK. Discrete-time recurrent neural network architectures: a unifying review. *Neurocomputing*, **15**(3 and 4):183–223, 1997.

[215] J. M. VARAH. Pitfalls in the numerical solution of linear ill-posed problems. *Journal on Scientific and Statistical Computing*, **4**.

[216] A. A. VARTAK, M. GEORGIOPOULOS, AND G. C. ANAGNOSTOPOULOS. Online gauss newton based learning for fully recurrent neural networks. *Nonlinear Analysis TMA*, **63**(5–7):867–876, 2005.

[217] E. WAN. *Finite impulse response neural networks with applications in time series prediction.* PhD thesis, Stanford, CA, USA, 1994.

[218] A. S. WEIGEND. *Connectionist Architectures for Time Series Prediction of Dynamical Systems.* PhD thesis, Department of Physics, Stanford University, June 1991.

[219] A. S. WEIGEND AND N. A. GERSHENFELD. Results of the santa fe time series competition. In *Proceedings of World Congress on Neural Networks Portland, OR (WCNN'93)*, pages IV–633–670, Hillsdale, NJ, 1993. Erlbaum.

[220] A. S. WEIGEND AND N. A. GERSHENFELD, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley, Reading, MA, 1994.

[221] A. S. WEIGEND, B. A. HUBERMAN, AND D. E. RUMELHART. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, **1**:193–209, 1990.

[222] A. S. WEIGEND, D. E. RUMELHART, AND B. A. HUBERMAN. Generalization by weight-elimination with application to forecasting. In R. P. LIPPMANN, J. E. MOODY, AND D. S. TOURETZKY, editors, *Advances in Neural Information Processing Systems 3 (NIPS*90)*, pages 875–882. Morgan Kaufmann, 1991.

[223] P. J. WERBOS. *Beyond Regression: New Tools for Regression and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Division of Engineering and Applied Physics, 1974.

[224] P. J. WERBOS. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, **78**, pages 1550–1560. IEEE Press, 1990.

[225] R. WERON. *Modeling and Forecasting Electricity Loads and Prices: A Statistical Approach.* John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, England, 2006.

[226] B. WIDROW AND JR. M. HOFF. Adaptive switching circuits. *IRE WESCON Convention Record, Pt. 4*, pages 96–104, 1960.

[227] R. J. WILLIAMS AND J. PENG. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, **4**:491–501, 1990.

[228] R. J. WILLIAMS AND D. ZIPSER. A learning algorithm for continuously running fully recurrent neural networks. *Neural Computation*, **1**:270–280, 1989.

[229] L. WU AND J. MOODY. A smoothing regularizer for feedforward and recurrent neural networks. *Neural Computation*, **8**(3):461–489, 1996.

[230] T. YAMADA AND T. YABUTA. Dynamic system identification using neural networks. *IEEE Transactions on Systems, Man, And Cybernetics*, **23**:204–211, 1993.

[231] G. YULE. On a method of investigating periodicity in disturbed series with special reference to wolfer's sunspot numbers. *Phil. Trans. Roy. Soc. London*, **A 226**:267–298, 1927.

[232] H. ZAREIPOUR, C. A. CANIZARES, AND K. BHATTACHARYA. Application of public-domain market information to forecast ontario's wholesale electricity prices. *IEEE Trans. Power Syst.*, **21**(4):1707–1717, 2006.

[233] D. ZIPSER. A subgrouping strategy that reduces complexity and speeds up learning in recurrent neural networks. *Neural Computation*, (1):552–558, 1989.