Development of finite memory neural fuzzy networks for lag-free improved time series forecasting

Subhrajit Samanta

2021

Subhrajit Samanta (2021). Development of finite memory neural fuzzy networks for lag-free improved time series forecasting. Doctoral thesis, Nanyang Technological University, Singapore. <https://hdl.handle.net/10356/146938>

<https://hdl.handle.net/10356/146938>

<https://doi.org/10.32657/10356/146938>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 16:09:06 SGT

Development of Finite Memory Neural Fuzzy Networks for Lag-Free Improved Time Series Forecasting

---

Subhrajit Samanta

Interdisciplinary Graduate School

Energy Research Institute @ NTU (ERI@N)

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

28th October 2020

.....

Date

Subhrajit Samanta

.....

Subhrajit Samanta

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

28th October 2020

.....

Date

Jun Zhao

.....

Assistant Professor Jun Zhao, SCSE

Co-Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

28th October 2020

.....

Date

.....

Dr. Narasimalu Srikanth, ERI@N

Authorship Attribution Statement

This thesis contains materials from 7 papers published in the following peer-reviewed journals/ conferences in which I am the first author.

Chapter 3 is published in three parts as,

1. Samanta, Subhrajit, Mahardhika Pratama, and Suresh Sundaram. "A novel spatio-temporal fuzzy inference system (spatfis) and its stability analysis." *Information Sciences* 505 (2019): 84-99
2. Subhrajit Samanta, Shubhangi Ghosh, and Suresh Sundaram. "A meta-cognitive recurrent fuzzy inference system with memory neurons (mcrfis-mn) and its fast learning algorithm for time series forecasting." In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp.2231-2237. IEEE, 2018.
3. Subhrajit Samanta, Andre Hartanto, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. "RIT2FIS: a recurrent interval type 2 Fuzzy Inference System and its rule base estimation." In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8. IEEE, 2019.

The contributions of the co-authors are as follows:

- I proposed the models, came up with the theorem to prove stability and implemented the algorithms. I also designed and conducted the experiments. The manuscripts are written by me as well.
- Asst. Prof. Maharadhika Pratma helped me with the analysis of the numerical results as well as in the revision of the manuscripts.
- Prof. Suresh Sundaram co-ordinated the projects, helped in the planning of the work-flow and reviewing the manuscripts before submission.
- Subhangi Ghosh worked alongside me as a student intern and helped me design and execute the experiments for paper 2, along with literature review.
- Andre Hartanto helped me in designing and performing the experiments for paper 3 as a part of his final year undergraduate project
- Dr. Narasimalu Srikanth helped in reviewing the manuscript (paper 3) and supported the project throughout its entirety.

Chapter 4 is published as,

4. Samanta, S., S. Suresh, J. Senthilnath, and N. Sundararajan. "A new neuro-fuzzy inference system with dynamic neurons (nfis-dn) for system identification and time series forecasting." *Applied Soft Computing* 82 (2019): 105567

The contributions of the co-authors are as follows:

- I formulated the problem, proposed the model and implemented the architecture. I also designed and conducted the experiments. Afterwards I wrote the manuscript too.
- Dr. Senthilnath helped me with the experiments and analysis of results.
- Prof. Sundaram supervised the project, helped in planning and designing experiments as well as reviewing the manuscripts before submission.
- Prof. Sundararajan helped in reviewing the manuscript and guided throughout the lifetime of this project.

Chapter 5 is published in two parts i.e.

5. Samanta, Subhrajit, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. "Learning elastic memory online for fast time series forecasting." *Neurocomputing* (2019).

6. Samanta, Subhrajit, Mahardhika Pratama, and Suresh Sundaram. "Bayesian Neuro-Fuzzy Inference System (BaNFIS) for Temporal Dependency Estimation", *IEEE Transaction on Fuzzy Systems* (2020)

The contributions of the co-authors are as follows:

- I proposed the probabilistic model, formalized the mathematical expressions and implemented the algorithms. I also planned and conducted the experiments. I wrote the manuscripts as well.
- Asst. Prof. Mahardhika Pratama helped me with the analysis of the results along with the revision/ modification of the manuscripts.
- Prof. Suresh Sundaram supervised the projects, helped in the work-flow design as well as in reviewing the manuscripts before submission.
- Dr. Narasimalu Srikanth gave valuable insights, helped in reviewing the manuscript (paper 5) and supported the project throughout its entirety.

Chapter 6 is published as

7. Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. "A Dual Network Solution (DNS) for lag-free time series forecasting", Accepted, 2020 International Joint Conference on Neural Networks (IJCNN)

The contributions of the co-authors are as follows:

- I formulated the problem, proposed the hybrid solution, and then implemented the algorithm. I also designed and performed the experiments. I wrote the paper as well.
- Asst. Prof. Mahardhika Pratama helped me with the numerical analysis and inference from the results and in the revision of the manuscripts.
- Prof. Suresh Sundaram supervised the project, helped in the planning as well as in reviewing the manuscript before submission.

- Dr. Narasimalu Srikanth supported the project from the very beginning, shared valuable insights and helped in reviewing the manuscript.

28th October 2020

.....

Date

Subhrajit Samanta

.....

Subhrajit Samanta

Acknowledgements

I would like to express my deepest gratitude to my supervisors. I started my Ph.D. journey with Prof. Suresh Sundaram and I am eternally thankful to him for bringing me to the world of research. His intelligent guidance and kind support over the past years have been truly exceptional. I have learned a great deal, not only about ways of critical thinking and analytical insights but also important life lessons of patience and perseverance from him. I am fortunate to have him as a mentor. Life at Nanyang Technological University would not have been the same without Prof. Suresh!

Prof. Mahradhika Pratama has been instrumental in my success with publications. His smart approaches for solving problems and fresh perspective towards research have inspired me a lot to think out of the box and ask new questions instead of just looking for answers to already existing problems. His supervision has been truly invaluable and I am very thankful for his guidance.

Dr. Narashimalu Srikanth has been a constant pillar of support since the beginning of my Ph.D. as my co-supervisor. He has been truly instrumental in perfecting my thesis. He has guided me with his wide expertise and knowledge of areas that are beyond my reach, thus enriching both my knowledge and quality of my research and I cannot be more thankful for everything he has done to help me and my work.

Prof. Jun Zhao has been extremely supportive in the little time he has supervised me. He has trusted me with my wits and given me complete independence and I am very thankful to him for that. He has taught me to look beyond my regular scope of research and learn more every day. Without his sincere help and support, this thesis would not have been possible.

I want to sincerely thank Prof. Dusit Niyato, Prof. Narasiman Sundararajan and all of my co-authors along with all my dear friends, colleagues and staff at Computational Intelligence Lab at NTU for their help, support and guidance.

I also acknowledge and thank the Interdisciplinary Graduate School and ERI@N, Nanyang Technological University for providing me with the financial support and this precious opportunity of a lifetime.

I have to thank Ms. Olivia Nadal Samanta most for putting up with me for all these years, being there for me through the good times and bad, through the successes and the many failures and most importantly being the anchor that always keeps me grounded, motivated and devoted to be a better person.

Finally, I would not be here where I am today if it was not for my family. Thank you does not do justice for the hard work and sacrifices my parents have made for me. I am and always will be eternally indebted to them for everything. Whatever I have achieved, it is all because of your support and blessing, ‘Ma’ and ‘Baba’!

“ I have not failed, I have successfully discovered 10,000 things, that won't work.”

— Thomas Alva Edison

To my 'Ma', my 'Baba', my 'Bhai' and my Fiance,
I am nothing without you...

Abstract

Time series modelling/ forecasting is one of the most popular areas of research in the machine learning and data science community. It has widespread applications across various domains from different sectors such as energy demand prediction, weather forecasting, wind speed forecasting, stock price prediction, Exchange rate prediction, pandemic growth forecasting etc. to name a few and the list goes on. This ever-growing field of study has seen some significant improvements over the past few decades since its inception starting from the classical statistical methods such as Auto-Regressive Moving Average (ARMA). The recent advents in artificial neural network based deep learning algorithms have propelled this research to a new high. However, with ever increasing amount of data every hour of each day, there are always more challenges and improvements to be made in this field. This thesis addresses a few of the major challenges associated with time series modelling which are well known to the community. I also look at some of the problems which are relatively under-explored.

One of the major known challenges of time series modelling is capturing the inherent temporal characteristics of the data. Most time series data are non-stationary in nature i.e. their statistical properties change over time. To model such data, it is of utmost importance to trace the underlying dynamics i.e. the temporal behavior. Another, big problem of time series modelling is to handle the uncertainty. Real-world time series data are often riddled with noise and volatility. Hence it is crucial to address the issue of uncertainty associated with such real-world data. Online learning from a time series data can also pose a novel issue as it requires the model to learn in a single epoch without any repetition. In the first two chapters, this thesis proposes two novel neuro-fuzzy systems to address these three challenges.

A Spatio-Temporal Fuzzy Inference System (or SPATFIS) with memory type neurons is proposed first which retains all past information to capture the system dynamics. It utilizes a new self-adaptive learning mechanism to add, eliminate and unify its fuzzy rules which helps it to attain a parsimonious structure. At the same

time, the linguistic nature of the fuzzy inference system makes it well capable of handling the uncertainty. SPATFIS also adopts a projection-based algorithm to update its parameters in a sequential manner thereby it is capable of online time series forecasting. However, when the system dynamics changes rapidly then memory neurons fail to adapt to the quickly changing dynamics (as they retain the effect of all past instances, thereby its output takes longer to get adjusted to a sudden shock). Therefore, it is very important to ensure that the temporal output is finite in nature and it is not depending on all pasts.

The Neuro-Fuzzy Inference System with Dynamic Neurons (NFIS-DN) addresses the aforementioned problem. The proposed Dynamic Neuron (DN) is designed in such way, that its temporal output considers only the effect of finite past instances, enabling the system with finite memory. The benefits of dynamic neurons (DNs) become more apparent in areas where there is a sudden change in the system dynamics as commonly seen in non-linear dynamic system identification problems. However, there is one drawback to the implementation of DN. All the samples need to be present to perform an empirical test to choose the number of past instances required for the model (i.e. temporal order or temporality: N), thereby making this process offline in nature. Hence in case of an online learning scenario, this method of choosing N will not hold.

Hence, a novel Bayesian method of online temporality analysis is proposed to estimate how many past instances a model requires for an improved time series forecasting; this results in finite memory. Temporality change or drift can be a common occurrence in online time series, hence a drift detection mechanism is also developed. The entire mechanism is termed as Learning Elastic Memory Online or LEMON. This method is agnostic of the underlying network or learning mechanism and can be utilized in any time series model. A neuro-fuzzy adaptation of LEMON is also developed by the name of Bayesian Neuro-Fuzzy Inference System (BaNFIS) to handle the problem of online temporality estimation and uncertainty handling simultaneously.

NFIS-DN, LEMON, and BaNFIS, all three of these models utilize only finite past information (hence finite memory) within their respective neural fuzzy frameworks to provide superior prediction performance. Apart from the aforementioned challenges, there is one more challenge i.e. ‘Lag in predicted sequence’ which is often overlooked in the time series literature as it does not contribute to a high mean

squared prediction error. Most neural network and neuro-fuzzy based time series models in literature (including proposed SPATFIS, NFIS-DN, LEMON and BaNFIS), trained with historical data alone can often lead to a lagged time series where the predicted sequence is always trailing the original sequence. This leads to poor forecast in terms of movement prediction.

In the final chapter of this thesis a novel trend driven Dual Network Solution (DNS) is proposed. Trend, defined as the inherent pattern of the data, is extracted here and utilized next to perform lag-free forecasting. DNS exhibits a substantially improved performance compared to more complex and resource-intensive state-of-the-art algorithms in large scale time series problems. Apart from the traditional Mean Squared Error (MSE), a new Movement Prediction Metric or MPM (for detection of lag in time series) is also developed as a new complementary performance metric to evaluate the efficacy of DNS better.

The performance of each of these proposed methods is evaluated meticulously on several benchmark data sets as well as real-world problems with a main focus on energy demand prediction and wind speed forecasting along with financial indicator prediction. The superior performance of the proposed methods further indicates their applicability across domains, especially in the energy sector. These methods could be instrumental for producing better renewable energy, reducing energy wastage, better planning and grid management etc. for a sustainable earth.

Overall, this thesis tackles several problems of time series forecasting, starting with the challenging tasks of capturing system dynamics, online learning and handling uncertainty of real-world data. It also addresses some under-explored issues of this domain including estimation of temporal order and the problem of lag. Future research in this domain, can benefit from this thesis and improve upon the presented methods to address other fundamental challenges of time series modelling.

Contents

Acknowledgements	xiii
Abstract	xvii
List of Figures	xxv
List of Tables	xxvii
Acronyms	xxix
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	5
1.3 Major contributions	6
1.4 Organization of the thesis	8
1.5 Publications from this thesis	10
2 A Brief Review of the Relevant Literature	13
2.1 Statistical methods of time series forecasting	14
2.2 Neural networks for time series forecasting	17
2.2.1 Feed-forward neural network	19
2.2.2 Recurrent neural network	21
2.3 Neuro-fuzzy inference systems for time series forecasting	25
2.3.1 Fuzzy Inference System (FIS)	26
2.3.2 Neuro-Fuzzy Inference System (NFIS)	28
2.3.2.1 Feed-forward neuro-fuzzy inference system	29
2.3.2.2 Recurrent neuro-fuzzy inference system	30
2.4 Learning algorithms for time series models	33
3 Spatio-Temporal Fuzzy Inference System (SPATFIS)	37
3.1 SPATFIS architecture	38
3.1.1 Memory Neuron (MN)	38
3.1.2 Layer wise operations	39
3.2 Learning in SPATFIS	42

3.2.1	Structure learning	42
3.2.2	Parameter learning	46
3.2.3	Rule reduction	48
3.3	Stability analysis in SPATFIS	49
3.4	Performance evaluation of SPATFIS	49
3.4.1	Standard benchmark problems	51
3.4.1.1	Mackey-Glass problem	51
3.4.1.2	Box-Jenkins problem	52
3.4.2	Real-world problems	52
3.4.2.1	Wind speed forecasting	52
3.4.2.2	Stock price forecasting	53
3.5	Limitations of SPATFIS	54
4	Neuro-Fuzzy Inference System with Dynamic Neuron (NFIS-DN)	57
4.1	NFIS-DN architecture	58
4.1.1	Dynamic Neuron (DN)	58
4.1.2	Layer wise operations	60
4.2	Learning in NFIS-DN	64
4.2.1	Self-regulated learning	64
4.2.2	Updating the consequent parameters	65
4.2.3	Updating the antecedent parameters	67
4.3	Performance evaluation of NFIS-DN	67
4.3.1	System identification problems	68
4.3.1.1	Nonlinear System Identification Problem I (SID 1)	68
4.3.1.2	Nonlinear System Identification Problem II (SID 2)	70
4.3.2	Standard benchmark problems	71
4.3.2.1	Mackey- Glass problem	71
4.3.2.2	Box-Jenkins problem	72
4.3.3	Real-world Problems	72
4.3.3.1	Wind speed forecasting	73
4.3.3.2	Stock price forecasting	74
4.4	Limitations of NFIS-DN	75
5	Learning Elastic Memory Online (LEMON)	77
5.1	Temporality Estimation (TE)	78
5.1.1	Memory window initialization	79
5.1.2	Probability derivations	80
5.1.2.1	Likelihood computation	80
5.1.2.2	Recursive prior probability	81
5.1.2.3	Posterior probability	81
5.1.2.4	Measuring temporality	82
5.2	Learning elastic memory	82
5.2.1	Temporal learning	83

5.2.2	Handling temporal drift	83
5.2.2.1	Drift indicator and drift detection	84
5.2.2.2	Memory window evolution	84
5.2.2.3	Temporality re-estimation	85
5.3	Proof of concept with synthtetic data	87
5.3.1	Nonlinear System Identification I	87
5.3.2	Nonlinear System Identification II	89
5.4	Performance evaluation of LEMON	91
5.4.1	Box-Jenkins problem	91
5.4.2	Wind speed forecasting	92
5.4.3	Stock price forecasting	93
5.4.4	Numerical analysis	94
5.4.4.1	Performance improvement	95
5.4.4.2	Performance comparison	96
5.5	Extension of LEMON with Neuro-Fuzzy Inference System	98
5.5.1	BaNFIS architecture	98
5.5.1.1	Global BaNFIS	99
5.5.1.2	Local BaNFIS	100
5.5.2	BaNFIS performance evaluation	101
5.5.2.1	Standard real-world problems	101
5.5.2.2	Large-scale real-world problems	103
5.6	Limitation of LEMON/ BaNFIS	105
6	Dual Network Solution (DNS)	107
6.1	Problem of ‘Lag’ in time series forecasting	108
6.2	Overview of the solution	109
6.3	Trend extraction	111
6.3.1	Moving Average Method (MAM)	111
6.3.2	Slope Difference Method (SDM)	112
6.4	Dual Network System	113
6.4.1	Trend prediction network (N_1)	113
6.4.2	Temporal network (N_2)	114
6.5	Performance evaluation of DNS	115
6.5.1	Discussion on hyperparameter setting	116
6.5.2	Performance indicators	116
6.5.3	Demonstration of lag-free forecast	117
6.5.4	Performance comparison	118
6.5.4.1	Financial indicator prediction	119
6.5.4.2	Energy consumption prediction	120
6.5.4.3	Baseline methods for comparison	123
6.5.5	Numerical analysis and discussion	124
6.5.6	Limitation of DNS	125

7	Conclusions and Future Work	127
7.1	Conclusions	127
7.2	Future work	129
A	Chapter 3 Additional Materials	131
A.1	Proof of Lemma 1	131
A.2	Proof of Lemma 2	132
A.3	Proof of Lemma 3	133
A.4	Proof of stability during learning	133
A.5	Rank based statistical tests	137
A.6	Extension of SPATFIS	138
B	Chapter 4 Additional Materials	139
B.1	NFIS-DN learning flow-chart	139
B.2	Additional experiment: Sunspot problem	140
B.3	Rank based statistical tests	141
C	Chapter 5 Additional Materials	143
	List of Author's Publications	145
	Bibliography	147

List of Figures

1.1	Time series example 1: Daily energy consumption data	2
1.2	Time series example 2: Hourly wind speed data	3
1.3	Time series example 3: Daily Google stock price data	4
2.1	Biological neuron diagram	18
2.2	Artificial neuron structure	19
2.3	Multi-layer perceptron network	20
2.4	Feed-forward artificial neural network	20
2.5	Recurrent neural network	22
2.6	Single LSTM unit	24
2.7	Fuzzy logic system	26
2.8	Locally recurrent neural-fuzzy system	31
2.9	Globally recurrent neural-fuzzy system	32
3.1	Realization of memory neuron	38
3.2	Network architecture of SPATFIS	40
3.3	Existing Fuzzy Rule Base	44
3.4	Rule Addition for the new Sample	45
4.1	Realization of a Dynamic Neuron	58
4.2	Network architecture of NFIS-DN	61
4.4	Wind speed prediction using NFIS-DN	73
4.5	Stock price prediction using NFIS-DN	74
5.1	Overview of Tempoality Estimation (TE) Process	79
5.2	Overview of LEMON	86
5.3	Temporal dependency estimation for SID	88
5.4	Temporal dependency Estimation for SID2	90
5.5	Temporal dependency estimation for BJ problem	91
5.6	Box Jenkins prediction using LEMON	92
5.7	Temporal dependency estimation for Wind problem	93
5.8	Wind speed prediction using LEMON	93
5.9	Temporal dependency Estimation for stock problem	94
5.10	Stock price prediction using LEMON	95
5.11	Global BaNFIS Architecture	99
5.12	Local BaNFIS Architecture	100

6.1	Demonstration of ‘Lag’ in time series forecasting	109
6.2	Overview of the Dual Network Solution (DNS)	110
6.3	Trend in wind dataset extracted with MAM	111
6.4	Trend in wind dataset extracted with SDM	112
6.5	Wind speed prediction with DNS	119
6.6	Amazon stock price prediction with DNS	121
6.7	Power consumption prediction with DNS	122

List of Tables

3.1	Performance of SPATFIS on MG problem	51
3.2	Performance of SPATFIS on BJ problem	52
3.3	Performance of SPATFIS on wind speed problem	53
3.4	Performance of SPATFIS on stock price problem	55
4.1	Performance of NFIS-DN on nonlinear SID I problem	70
4.2	Performance of NFIS-DN on nonlinear SID II problem	71
4.3	Performance of NFIS-DN on MG problem	72
4.4	Performance of NFIS-DN on BJ problem	72
4.5	Performance of NFIS-DN on wind speed problem	73
4.6	Performance of NFIS-DN on stock price problem	75
5.1	Performance improvement with LEMON	96
5.2	Performance of LEMON on BJ problem	97
5.3	Performance of LEMON on wind speed problem	97
5.4	Performance of LEMON on stock price problem	98
5.5	Performance of BaNFIS on BJ problem	102
5.6	Performance of BaNFIS on wind speed problem	102
5.7	Performance of BaNFIS on stock price problem	103
5.8	Performance of BaNFIS on large scale time-series problems	105
6.1	Performance of DNS on wind speed problem with MAM	118
6.2	Performance of DNS on wind speed problem with SDM	118
6.3	Performance of DNS on single step Amazon stock price prediction .	120
6.4	Performance of DNS on single step energy consumption prediction	122
6.5	Performance of DNS on large scale time-series problems	124
B.1	NFIS-DN Performance on sunspot problem (additional experiment)	140

Acronyms

ACF	Auto-Correlation Function
ANN	Artificial Neural Network
AR	Auto-Regression
ARIMA	Auto-Regressive Integrated Moving Average
ARIMAX	Auto-Regressive Integrated Moving Average with Exogenous Factor
ARMA	Auto-Regressive Moving Average
ATNN	Adaptive Temporal Neural Network
BP	Backpropagation
BPTT	Backpropagation Through Time
BRITS	Bidirectional Recurrent Imputation for Time Series
BaNFIS	Bayesian Neuro-Fuzzy Inference System
DFNN	Dynamic Fuzzy Neural Network
DL	Deep Learning
DN	Dynamic Neuron
DNS	Dual Network Solution
DRFNN	Diagonal Recurrent Fuzzy Neural Network
EKF	Extended Kalman Filter
ESAFIS	Extended Self-Adaptive Fuzzy Inference System
eTS	evolving Takagi Sugeno
i.i.d.	independent and identically distributed
GENEFIS	Generic Evolving Neuro-Fuzzy Inference System
IRSFNN	Interactively Recurrent Self-evolving Fuzzy Neural Network
LEMON	Learning Elastic Memory Online
LSTM	Long Short Term Memory
MA	Moving Average
MAE	Mean Absolute Error
ML	Machine Learning

MN	Memory Neuron
MPM	Movement Prediction Metric
MSAFIS	Modified Self-Adaptive Fuzzy Inference System
MSE	Mean Squared Error
McFIS	Meta-cognitive Fuzzy Inference System
McFIS-MN	Meta-cognitive Fuzzy Inference System with Memory Neuron
NARMA	Non-linear Auto-Regressive Moving Average
NDEI	Non-dimensional Error Index
NFIS	Neuro-Fuzzy Inference System
NFIS-DN	Neuro-Fuzzy Inference System with Dynamic Neuron
PACF	Partial Auto-Correlation Function
PALM	Parsimonious Learning Machine
PANFIS	Parsimonious Neuro-Fuzzy Inference System
PBL	Projection Based Learning
RFNN	Recurrent Fuzzy Neural Network
RIT2FIS	Recurrent Interval Type 2 Fuzzy Inference System
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RSOINFIN	Recurrent Self-Organizing Neural Fuzzy Inference Network
RTRL	Real Time Recurrent Learning
SAFIS	Self-Adaptive Fuzzy Inference System
SARMA	Seasonal Auto-Regressive Moving Average
SPATFIS	Spatio-Temporal
STMVL	Spatio-Temporal Multi-View based Learning
SUFIN	Sequential Update Fuzzy Inference Network
SimpleTS	Simple evolving Takagi Sugeno
TRFN	Tsk Recurrent Fuzzy Network
TSK	Takagi, Sugeno and Kang
VARMA	Vector Auto-Regressive Moving Average
WRFNN	Wavelet based Recurrent Fuzzy Neural Network

Chapter 1

Introduction

1.1 Motivation

A time series can be described as a chronological sequence of values at equally spread intervals. Time series analysis has become one of the major researched upon areas in recent times because of its wide applications for various purposes such as analyzing historical data, projecting future trend, making decisions, forming strategies, taking precautionary measures etc. To model a time series data, one needs to collect historical data over time and make sure that the observations are ordered.. The major difference between a time series and a standard regression problem is that the data is not necessarily independent and identically distributed (i.i.d) in time series. Generally, there is always a temporal (i.e. dynamical) relationship (hence not independently distributed) between the observations in a time series. Because of this dependency, ordering of the observations is very important and changing the order could change the meaning of the data which makes it very different from a regular regression type problem.

The usage of time series analysis is mainly two-fold, firstly to understand the underlying pattern/ distribution that generated the observed data and secondly fitting a model with the observed data to forecast into future. The second aspect of time series is its capability to explain how the past effects the future. Time series forecasting [1, 2] has been recognized as a popular research field for a long time because of its various implications across fields such as energy consummation

prediction, sales and budgetary prediction and analysis, economic prediction, stock forecasting, inventory management, workload prediction etc.

Let us take the example of the energy industry. It is obvious that there's a lot of wastage of power all over the world because of poor grid management, inefficient operational methods etc. In a time when the world needs to move onto renewable energy because of the dearth of fossil fuels and global warming, wastage of energy is simply unacceptable. Now if there was a forecasting model capable of tracking and forecasting the energy usage pattern ahead of time with good accuracy and efficiency, it could be saving megawatts of energy, making the electricity price go down to a reasonable extent. Factors such as weather, seasonal fluctuation, public holidays and weekends along with the operational strategies of the plants and unforeseen faults in the grid are among the most important factors leading to the uncertainty of load demand and generations.

FIGURE 1.1: Time series example 1: Daily energy consumption data

Figure 1.1 shows the power consumption pattern in a household in France. This data is downloaded from UCI-ML repository¹ and it contains the details of electric power consumption in one household for the year of 2010. The voltage sequence is the target time series. From the figure, the volatile behavior of data can be easily noticed.

¹<https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption>

Let us take another example of a wind speed prediction problem. In recent years, there has been a significant development in the field of wind power because of its potential as a sustainable source of clean energy. Considering the statistically significant relationship between wind power and wind speed, an effective forecasting of wind speed will highly benefit this industry.

FIGURE 1.2: Time series example 2: Hourly wind speed data

The wind dataset shown above is acquired from Iowa (USA) Department of transport at the location, Washington. The data is collected over a period of February 1, 2011, to February 28, 2011 ². The wind time series data contains a significant amount of noise and impreciseness i.e. uncertainty possibly because of the stochastic nature of the weather conditions as well as the noisy sensors.

Let us take one last example of financial time series prediction such as stock price forecasting. The stock price of any organization is a highly volatile quantity with no easily distinguishable pattern. The underlying dynamics changes fast and abruptly (as can be seen for Google stock price time series data in Fig. 1.3). Stock price fluctuation depends on various financial and technical indices with different time windows (number of past instances to be used) hence the exact functional relationship is unknown and too complex to solve in a deterministic way. In such scenarios necessity of a good forecasting algorithm becomes all the more prominent.

²<http://mesonet.agron.iastate.edu/request/awos/1min.php>

FIGURE 1.3: Time series example 3: Daily Google stock price data

Likewise, one can have better forecasting of weather and natural calamities, better exchange rate predictions, better inventory management, better budget planning, better model for disease spread etc. by developing an efficient and accurate time series forecasting algorithm. However, there are many fundamental challenges associated with time series data which are required to be addressed to build an effective forecasting algorithm. Some of these challenges are well known in the community such as capturing system dynamics, dealing with uncertainty and online learning ability. Some of the challenges however are under-explored i.e. estimation of temporal dependency and lag in the predicted sequence.

Numerous papers in the literature have been written in the broad field of time series forecasting to come up with better, more accurate and faster time series models. There have been many works with classical statistical methods (such as Auto Regressive Moving Average - ARMA [3–6]). Methods such as AR, MA, ARMA in spite of their popularity face some shortcomings. These models are mostly linear (except for some modified methods such as NARMA[7]), they cannot handle uncertainty in data well and they are offline. Machine learning methods became very popular after the advent of artificial neural networks because of their non-linear modeling ability. ANNs are also data driven and does not make any presumption about the statistical distribution of the data. Since then there have been many works in the field of time series analysis with different neural network based models [8]. Time

series can be often non-stationary in nature and to capture the system dynamics, feed-forward neural networks are not well suited. Recurrent neural networks however, are often used in such temporal settings to handle the system dynamics. On the other hand to address the issue of uncertainty and impreciseness in a real-world time series, fuzzy inference systems[9] can be very useful. Fuzzy inference systems are capable of mimicking human behavior by using a linguistic model and they also provide interpretability. Broadly there are two types of fuzzy inference engines i.e. Mamdani[10] and TSK[11] fuzzy. Empirically it has been shown in the literature that TSK-type fuzzy systems are more suitable for time series forecasting because of their lower complexity compared to Mamdani models where computational cost is higher because of the need for further defuzzification. They also outperform the Mamdani models in terms of accuracy in most cases (discussed further in the literature review section). Therefore, the main motivations of this thesis are to address the fundamental challenges of time series modeling with the help of recurrent neural networks and neuro-fuzzy systems in order to build improved forecasting algorithms.

1.2 Objectives

The main objective of this thesis is to build efficient and effective time series forecasting algorithms with good accuracy, low complexity and faster training. At the same time, this thesis also aims to address several fundamental challenges of time series modeling. More specifically, the objectives of the work in this thesis can be summarized as below,

- To develop an efficient algorithm that can trace the underlying temporal characteristics of the non-stationary time series data effectively. It is very important to handle the inherent system dynamics properly. A time series model can perform its forecasting task well when it is able to capture the temporal characteristics of the data properly. Hence this thesis tries to tackle the challenging issue of handling system dynamics.
- To propose a time series model that is well capable of handling the uncertainty/ impreciseness that steams from real-world noisy data. This thesis

addresses the problem of uncertainty in time series both in an offline and online manner. In an online learning framework a network has to self-evolve its structure from the scratch and updates its parameters, both in a sequential manner (hence online) to learn from the time series data.

- To propose an online methodology, which can estimate the temporal dependency (i.e. temporality) quantitatively in an explicit manner. ‘How many past observations are required to model a time series data?’, this has always been a highly important topic in time series domain. However, there has not been much work in this domain apart from auto-correlation (ACF) and partial autocorrelation (PACF) based methods. However both of these methods are linear and offline in nature (needs all samples to be present). Hence there is a requirement for an online temporality estimation method for time series data.
- To construct a solution for the under-explored issue of ‘Lag’ in the predicted sequence of a time series. ‘Lag in predicted sequence’ is often overlooked in the time series literature as it does not contribute to a high prediction error (such as MSE, RMSE). However, any network based time series model, trained with historical data alone often leads to a lagged time series where the predicted sequence is always behind. This leads to poor forecasts in terms of movement prediction. This thesis also aims to address this issue.

1.3 Major contributions

Major contributions of this thesis are many folds as listed below,

- A Spatio-Temporal Fuzzy Inference System (SPATFIS) is proposed in this thesis. One of the prime features of SPATFIS lies in its memory neurons which incorporates the temporal (past) information of the time series with a dual recurrent structure in its input and hidden layers. SPATFIS also proposes a new self-adaptive learning mechanism to add, eliminate and unify its fuzzy rule, which helps it to attain a parsimonious, low complex structure. Furthermore, stability is rigorously inspected and SPATFIS is proved to be stable using Lyapunov’s Input to State Stability theorem. Finally, it adopts a

projection-based learning algorithm to update its parameters in a sequential manner thereby making SPATFIS capable of online time series forecasting.

- One major drawback of the memory neurons in SPATFIS is that it retains the effect of all past instances i.e. it has effectively infinite memory. Thereby the temporal output takes longer to get adjusted to a sudden shock. When the system dynamics changes rapidly (i.e. a sharp drop in the stock price due to bad news etc.) then memory neurons fail to adapt to that quickly changing dynamics leading to poor performance of SPATFIS.

Hence, a Neuro-Fuzzy Inference System with Dynamic Neurons or NFIS-DN is proposed. The proposed Dynamic Neuron (DN) is designed such way, that its temporal output considers only the effect of finite past instances, enabling the system with finite memory. Thus it is able to remove the effect of infinite past associated with memory neurons in SPATFIS. The benefits of dynamic neurons (DNs) become more apparent in areas where there is a sudden change in the system dynamics (such as non-linear dynamic system identification problems). Both SPATFIS and NFIS-DN are well capable of handling the uncertainty associated with real-world time series data with their fuzzy inference engine.

- Traditionally, the temporal dependency (i.e. number of past samples required for a good prediction) is estimated with auto-correlation or partial auto-correlation function (or empirically), thus they are generally offline in nature (as in NFIS-DN), making them impractical to be adopted in the online or streaming context. Hence, a novel method of online temporality analysis is proposed in this thesis.

The proposed mechanism utilizes Bayesian probability to estimate the required number of past instances i.e. Temporality (or temporal dependency) thus providing the model with finite memory. In an online time series context, temporality change or drift can be a common occurrence, hence a drift handling mechanism is also proposed. The entire mechanism is termed as Learning Elastic Memory Online or LEMON. Further, LEMON is extended with a neuro-fuzzy inference system to build a Bayesian neuro-Fuzzy Inference System (BaNFIS) to handle both uncertainty and temporality estimation with its finite memory neuro-fuzzy structure.

- The unexplored problem of ‘Lag’ is very common in most of the network based time series models (even in the models proposed in this thesis i.e. SPATFIS, NFIS-DN or LEMON). ‘Lag in predicted sequence’ goes overlooked because it does not lead to a high MSE/ RMSE. Proposed Dual Network Solution defines this problem more elaborately and proposes a novel way with which this issue can be addressed. DNS also proposes a new performance metric that is more suited to detect the presence of such lag in the forecast.

1.4 Organization of the thesis

The thesis is organized in the following way,

- Chapter 2 provides a literature review broadly on the domain of time series forecasting. Starting with a brief introduction to the classical statistical approaches, this chapter then moves on to the discussion on the neural networks based approaches for time series prediction. Next, the discussion revolves around fuzzy systems and neuro-fuzzy based methods with reference to time series forecasting. In the final part of this chapter, the focus is on different learning mechanisms that can be applied for time series forecasting.
- Chapter 3 proposes the Spatio-Temporal Fuzzy Inference System or SPATFIS. The structure and working principle of the memory neurons and how they are integrated into the neuro-fuzzy architecture of SPATFIS is discussed first. Then the discussion focuses on a new self-adaptive learning mechanism that adds, eliminates and unifies its fuzzy rule base in order to learn the structure in an online manner. Finally, the description of how it adopts a projection-based learning algorithm to update its parameters in a sequential manner thereby making SPATFIS capable of online time series forecasting. Furthermore, stability is rigorously inspected and SPATFIS is proved to be stable using Lyapunov’s Input to State Stability theorem. Experiments are performed on standard benchmark problems as well as real-world problems to evaluate the efficacy of SPATFIS at the end of this chapter.
- Chapter 4 proposes a Neuro-Fuzzy Inference System with Dynamic Neurons or NFIS-DN. This chapter starts with the discussion on the proposed Dynamic Neuron (DN) and how it retains the effect of finite past instances

compared to the infinite memory associated with memory neurons in SPAT-FIS. Then the architecture of NFIS-DN and the self-regulated backpropagation learning mechanism are discussed. The benefits of dynamic neurons (DNs) are demonstrated with non-linear dynamic system identification problems. Finally, experimental results are provided for performance evaluation at the end.

- Chapter 5 proposes a novel Bayesian method of online temporality analysis by the name of Learning Elastic Memory Online (LEMON). Starting with the formal definition of Temporality (i.e. temporal dependency), the discussion moves on to the proposed Bayesian estimation method. Next, the discussion is on how a temporal drift can be detected and handled with this mechanism. The performance of this method is evaluated in two steps, first with synthetic data for proof of concept and next with real-world time series data. This chapter also introduces the Bayesian Neuro-Fuzzy Inference System (BaNFIS) with its two variants, for the purpose of better handling of uncertainty while utilizing the estimated temporality (based on LEMON) for a superior prediction performance.
- Chapter 6 first explains the under-explored problem of ‘Lag’ in the predicted sequence of a time series. It talks about why it is often overlooked in the literature. Then it proposes a hybrid dual network solution to address this ever-present issue of time series forecasting. This chapter also proposes a novel performance metric to detect this issue (as standard measures fail to do so). Real-world large scale time series problems are utilized here to demonstrate the superior performance of DNS.
- Chapter 7 provides the conclusions of the previous chapters as well as discusses the future directions of possible research that can stem from this thesis.

1.5 Publications from this thesis

The publications that emerged from this thesis are listed below,

Journal Articles :

1. Subhrajit Samanta, Mahardhika Pratama, and Suresh Sundaram. "A novel spatio-temporal fuzzy inference system (spatfis) and its stability analysis." *Information Sciences* 505 (2019): 84-99.
2. Subhrajit Samanta, Sundaram Suresh, J. Senthilnath, and N. Sundararajan. "A new neuro-fuzzy inference system with dynamic neurons (nfis-dn) for system identification and time series forecasting." *Applied Soft Computing* 82 (2019): 105567.
3. Subhrajit Samantat, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. "Learning elastic memory online for fast time series forecasting." *Neurocomputing* (2019).
4. Subhrajit Samanta, Mahardhika Pratama, and Suresh Sundaram. "Bayesian Neuro-Fuzzy Inference System (BaNFIS) for Temporal Dependency Estimation." *IEEE Transactions on Fuzzy Systems* (2020).

Conference Proceedings :

5. Subhrajit Samanta, Shubhangi Ghosh, and Suresh Sundaram. "A meta-cognitive recurrent fuzzy inference system with memory neurons (mcrfis-mn) and its fast learning algorithm for time series forecasting." In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 2231-2237. IEEE, 2018.
6. Subhrajit Samanta, Andre Hartanto, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. "RIT2FIS: a recurrent interval type 2 Fuzzy Inference System and its rule base estimation." In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8. IEEE, 2019.

-
7. Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. “A Dual Network Solution (DNS) for lag-free time series forecasting”, 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020.

Chapter 2

A Brief Review of the Relevant Literature

A time series is a ordered sequence of values at equal intervals. When there is a single target sequence, it is referred to as an univariate time series whereas when we have multiple ordered sequences we can call it a multi-variate time series. Multivariate time series in general can be thought of as a collection of independent univariate time series (unless the variables are correlated, which is out of the scope of this discussion) hence the focus of this thesis will be on modeling univariate time series and what are its challenges. The main goal of time series modeling is to train a model with the available historical data so that it can learn the underlying pattern/ distribution of the observed data and the relationship between past and future. The second aspect is to forecast into the future, which is widely known as time series forecasting.

The application of time series modeling/forecasting is widespread. It is used for various purposes (such as analyzing historical data, projecting future trend, making decisions, forming strategies, taking precautionary measures etc.) across different applications (energy consumption prediction, sales and budgetary prediction and analysis, economic prediction, stock forecasting, inventory management, workload prediction etc.). Due to its wide cross-domain applications, the research interest in this field has seen steady growth over the past decades. Before diving deep into the time series literature let us discuss a little more on the components which are utilized for a classical way of modeling a time series.

Traditionally a time series is defined with three main components [1] as below,

- **Trend :** Overall tendency of a time series over a sufficiently larger period is basically coined as its trend. For example, a time series of the number of the population has an upward or increasing trend.
- **Seasonality :** Similar fluctuation in a time series during a particular time period in a year can be referred to as the seasonal component or seasonality. An increase in the sale of cold drinks in summer can be an example of seasonality.
- **Random component :** This is because of the unforeseen and unpredictable influences. In a time series model, this is attributed to the prediction error. These are irregular and not repetitive. These are caused by factors such as war, strike, natural calamities etc.

A time series model can be represented based on these 3 components in an additive fashion (addition of the 3 components) or a multiplicative fashion (multiplication of the components). However, most real-world time series data have nonlinear, chaotic and dynamic characteristics making the modeling of a time series even more a daunting task. There have been numerous researches in this broad field to develop methods of time series modeling for a long time. Box and Jenkins [1] have discussed in detail various aspects of time series forecasting (from definition, mathematical formulation, nature of time series data, different examples, a range of methods to deal with time series forecasting problem etc.). Hamilton also has reviewed the work and progress in this field very well in the book [2]. In this thesis, first we will briefly go over the classical statistical approaches for time series forecasting, followed by their limitations and how neural network and neuro-fuzzy based machine learning methods address those limitations. Finally, we will end the literature review with a brief discussion on the learning methods for the network-based models (to perform time series modeling).

2.1 Statistical methods of time series forecasting

Before building a statistical time series model there are some steps which should be taken as standard/ good practice whenever possible, these are as follows [1],

- Plotting the time series to visually inspect for trend, seasonality or random behavior (i.e. outlier). Time series decomposition methods can be used for this purpose.
- It is important to check if the time series is stationary (its property remains constant over time) or not. An Augmented Dickey-Fuller (ADF) test can be performed for this purpose.
- If the time series is non-stationary (most real world time series are) we need to make the series stationary, by the usage of de-trending or differentiating. Integrated ARMA (i.e. ARIMA) models can also handle non-stationary time series data.
- Plotting auto-correlation function (ACF) or partial auto-correlation function (PACF) to find the optimal orders of the time series model.
- Building the model.

As most real-world time series data are non-stationary in nature, they need to be converted to a stationary time series first. Afterwards, statistical methods such as AR, MA, ARMA etc. can be fitted to model that data in different ways. Let us start with the simpler linear Auto-Regressive (or AR) model [3]. It is one of the most popular and widely used statistical method for time series modeling.

AR model : With AR models, a univariate time series is defined using its past values alone in a linear summation manner. For example, an unit order (uses only 1 past value) auto-regressive model can be formulated using the following equation (as can be found in [4, 5]),

$$\hat{y}(k) = \alpha * y(k - 1) + e(k) \quad (2.1)$$

where α is the model parameter or weight and $e(k)$ is the prediction error (which the model cannot explain i.e. the random component). From the equation, it is apparent that a forecast value i.e. $\hat{y}(k)$ of the target variable will depend heavily on the present $y(k)$ value. The weight/s are determined with a least square solution in the presence of all data-points. Partial Auto-Correlation Function (PACF) is generally used to estimate the AR order of a model.

When tasked with multiple step ahead forecast: $\hat{y}(k + \tau)$ the AR model uses the predicted values to forecast in future. For example, in case of a 2 step ahead forecast (i.e. $\tau = 2$), the time series is modeled as,

$$\hat{y}(k + 2) = \alpha * \hat{y}(k + 1) + e(k + 2) = \alpha * (\alpha * y(k) + e(k + 1)) + e(k + 2)$$

because of the auto-regressive nature, AR model takes longer to adjust to an abrupt change to $y(k)$. Hence such models do not perform well in the presence of sudden shocks. This is why Moving Average (MA) models are introduced.

MA model : Main motivation behind using a Moving Average (MA) model is to adjust to a shock quickly. That is why with MA models, an univariate time series is defined using its past prediction errors (instead of past values as shown in AR). For example, An unit order moving average of MA model can be defined by the following equation as in [5, 6] ,

$$\hat{y}(k) = \beta * e(k - 1) + e(k) \quad (2.2)$$

where β is the model parameter and $e(k - 1)$ is the last prediction error. In MA models, the forecast $\hat{y}(k + 1)$ depends heavily on the previous error. Hence it is able to adapt to the abrupt change/shock quickly (as the high error caused by the shock is used in MA model). Auto-Correlation Function (ACF) is employed here to estimate the MA order of the model.

ARMA model : An Auto-Regressive Moving Average or ARMA model is the combination of both AR and MA model to reap the benefits of both models. As seen in [3–6], an unit order ARMA can be modeled with the equation below,

$$\hat{y}(k) = \alpha * y(k - 1) + \beta * e(k - 1) + e(k) \quad (2.3)$$

where α, β are model parameters. The model parameters are estimated similarly in an ordinary least square method. A generalized version of ARMA is given,

$$\begin{aligned} \hat{y}(k) = & \alpha_1 * y(k - 1) + \alpha_2 * y(k - 2) + \alpha_3 * y(k - 3) + \cdots \alpha_p * y(k - p) + \\ & \beta_1 * e(k - 1) + \beta_2 * e(k - 2) + \beta_3 * e(k - 3) + \cdots \beta_q * e(k - q) + e(k) \end{aligned} \quad (2.4)$$

where p is the AR order and q is the MA order (derived with ACF, PACF)). It is apparent that setting either of the orders to 0, we can get a strictly AR or MA model from the generalized ARMA model.

Limitations of ARMA : Since its first introduction, the ARMA and other ARMA based approaches have become widely used across many fields of time series modeling. However it has certain shortcomings as listed below.

- ARMA models are built for handling linear time series data
- ARMA models work well with stationary data. Hence a non-stationary data needs to be de-trended/ differentiated first to make it stationary before it can be fitted with an ARMA model.
- ARMA models are not equipped to handle uncertainty in data
- ARMA models require relatively long sequence of data to learn from
- No online learning is possible with ARMA models as it learns offline

There has been some works in the field of non-linear ARMA models (NARMA i.e. [7]) as well as other variants such as Integrated ARMA (ARIMA), ARIMA with exogenous factors (ARIMAX), Seasonal ARIMA (SARIMA), Vector ARMA (VARMA) etc. to improve upon existing ARMA models. However, in recent times Artificial Neural Networks (ANN) based models have become more popular in this field. In the following section, the discussion focuses on neural networks, their variations, why and how they are used in the context of time series forecasting.

2.2 Neural networks for time series forecasting

Machine Learning (ML) is arguably the most important branch of artificial intelligence. With machine learning a computer (i.e. machine) learns without explicit instructions like human brain does. Machine learning can be of few types which are supervised, semi-supervised, unsupervised and reinforcement. Here the discussion will be restricted to supervised machine learning since time series forecasting falls under this category. In time series forecasting, the goal is to learn from the historical data where the labelled target value is present, hence it is supervised. There are many ways to perform machine learning (such as Support Vector Machine, k Nearest Neighbor, Decision Tree, Random Forest etc.) however with huge advances in the computational capability of computers, the ANN has become state-of-the-art hence most popular in many fields of ML including time series modeling.

We have billions of neurons (nerve cells) in our body which are the basic building blocks of our central nervous system. A single neuron (as shown in the diagram of figure 2.1 ¹) has dendrites which act as receptors. These are electrically excitable so when a stimulation occurs they receive that signal and transmit that information through electrical and chemical signals via the axon to the next neuron or organ.

FIGURE 2.1: Biological neuron diagram

Artificial neural networks (ANN) are an efficient computing system whose central theme is borrowed from the analogy of the biological neural networks. ANNs can also be referred to as ‘artificial neural systems’, or ‘parallel distributed processing systems’, or ‘connectionist systems’. In general, ANNs have a large number of interconnected units (i.e. building blocks) in some pattern to allow a flow of information and communication between the units. These units are also referred to as nodes or neurons or perceptrons.

A basic model of a perceptron is provided in figure 2.2 ². Perceptrons resemble biological neurons in the way of accepting multiple inputs, processing them with nonlinear activation function and obtaining output. As can be seen in the Fig.2.2 the n inputs are weighted and summed with a bias term such that $x = \mathbf{W}^T \cdot \mathbf{X} + b$ (with $\mathbf{X} \in \mathbb{R}^n$ as the input vector, $\mathbf{W} \in \mathbb{R}^n$ as the weight vector and b as the bias term). Then x is passed through a non-linear activation function to achieve the desired non-linearity $\hat{y} = f(x)$. The human nervous system consists of billions of

¹Biological neuron diagram in figure 2.1 is adopted from [12]

²Artificial neuron diagram in figure 2.2 is adopted from [13]

FIGURE 2.2: Artificial neuron structure

neurons, similarly for an ANN to perform and execute a certain supervised task it also requires many such nodes with multiple layers. In the commonest form of ANN, there is one input layer where all the inputs (features) are fed followed by a single or multiple hidden layer/s. Note that the nodes in the hidden layers are connected to all of them in the input and the output layer when it's a fully connected approach. Finally, there's an output layer where the signals are aggregated and the output is obtained. This is also referred to as multi-layer perceptron (MLP) network as shown in Fig. 2.3.

where $x(k) \in \mathbb{R}^m$ denotes the inputs (features/ factors), $h(k) \in \mathbb{R}^N$ denotes the hidden node outputs (activation) and $y(k) \in \mathbb{R}^1$ is the network output. w_{ir}, w_r denotes the weights of the connections. The k denotes time instance which is important in the study of time series analysis (In other form of machine learning we do not need the k index). Artificial neural networks can be broadly categorized into two groups depending on the direction of information flow.

2.2.1 Feed-forward neural network

The flow of data is unidirectional from the input side to the output side and no feedback loops are present in feed-forward neural networks[14]. These are mostly used in pattern recognition problems, simple regression and classification tasks

FIGURE 2.3: Multi-layer perceptron network

where non-linearity is essential. Figure 2.3 can be considered as a feed-forward type network.

FIGURE 2.4: Feed-forward artificial neural network

There are many types of activation functions which can be utilized to form different types of neural networks. Some of the most popular activation functions are linear

($f(x) = mx$), sigmoid ($f(x) = \frac{1}{1+\exp(-x)}$), tanh ($f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$), relu ($f(x) = \max(0, x)$), leaky relu ($f(x) = \max(0.1x, x)$) etc. Radial Basis Function or RBF [15] is another type of feed-forward neural network which enjoys high popularity especially in the field of time series forecasting. RBF network generally utilizes Gaussian activation functions and looks at the distance between the input and the centers of the Gaussian function ($f(\mathbf{X}) = \exp(-r^2/\sigma^2)$ where $r = \mathbf{X} - \mu$ with μ, σ as the center and the spread of the activation function). RBF networks are well known for making accurate global mapping and refining local features. Please keep in mind, that feed forward networks have at least one hidden layer between its input and output layers. When the number of hidden layers are substantially high this is referred to as a deep neural network or deep MLP. Deep networks are particularly useful when the amount of data is high and there is significant non-linearity involved with the data. On the other hand for smaller dataset, a shallow neural network (with smaller number of hidden layers) may be more practical.

In spite their wide applications, feed-forward networks are not able to deal with data of dynamic nature. In non-stationary environment it is necessary to retain past information, hence recurrent neural networks are proposed.

2.2.2 Recurrent neural network

The information flow can be bi-directional and feedback loops are present [16] in recurrent neural networks (RNN). Time series forecasting, natural language processing are among the most important fields where recurrent neural networks are extensively used. The diagram of a basic recurrent neural network is presented in figure 2.6.

In this thesis, the discussions will concentrate mostly on the recurrent type neural networks as they are more suitable for time series forecasting than feed-forward networks. Different researchers have worked on different approaches to bring in recurrence into a neural network framework. In this section we will focus on some of the most prominent shallow recurrent neural network structures followed by a discussion on the state-of-the-art deep recurrent neural networks.

Shallow Recurrent Network : One of the initial design in this field was to introduce tap delay type external feedback. In this framework, past instances of

FIGURE 2.5: Recurrent neural network

the required variables (target and exogenous) are fed in as input to the model. However, one needs to keep in mind that to design a tap delay neural network the dynamical system has to be observable and order of the system has to be known (Billings [17] and Narendra [18]). A number of other works also use tap delay system with nonlinear function approximation such as [19–22]. Another major drawback of tap delay approach is, if the system characteristics change during training (can be a common phenomenon in an online/ streaming scenario), then the model may fail to approximate the system accurately. Some control and time series application ([23, 24]) use tap delay type recurrent networks as well.

Another approach for bringing recurrence is the incorporation of recurrent type memory neurons. Presence of these recurrent type filters throughout the layers makes these networks both locally and globally recurrent in nature (in contrast to tap delay network which was only globally recurrent). In the memory neuron network (MNN [25]) memory nodes have been associated with all the nodes in each layer (i.e input, hidden and output layers). The memory elements are comprised of a delayed output from the parent node and a delayed self-feedback. MNN performs well with problems of dynamic nature, as it is well capable of capturing

the temporal information with its memory neurons. In the memory neuron network [25] each of the neuron contains one spatial node and one memory neuron containing self-feedback leading to a both locally and globally recurrent system. However, there is one major issue with these type of memory neurons. They bring recurrence in the neural network by taking into account of all past instances with exponentially decreasing coefficients. If the system characteristics changes abruptly (or midway during training) then the recurrent mechanism in both global and local restrict the network to change faster and hence they are not able to approximate the dynamics quickly. Among other shallow recurrent neural networks, the most notable ones are real-time recurrent learning [26], Jordan- Elman architecture [27] and FARN[28]. All of these networks utilize locally interconnected feedback to bring recurrence into the system. At the same time, there have been tremendous advancements in the field of deep recurrent neural networks (in recent times which has become synonymous with recurrent networks) for their improved natural language modeling capability compared to shallow nets.

Deep Recurrent Network : One of the first deep recurrent neural networks was initially proposed by [29]. Since then it has been at the forefront of researches in the machine learning community for a long time due to its widespread applications in varied fields (especially time series forecasting, language modeling etc.). One of the major challenges associated with training a deep RNN with a regular backpropagation through time method (which will be discussed in next section) was the problem of vanishing or exploding gradient [30]. To train a recurrent neural network, the backpropagation happens over the unrolled network in time which means that the gradient of the loss gets accumulated over all time steps during weight update. In such scenarios it is very easy for the final gradient to get too big (exploding) or too small (vanishing). Both of this phenomenon results in failed training of the neural network. This problem hindered the popularity of RNN in the beginning until [31] proposed Long Short-Term Memory or LSTM. LSTM is one of the most well-received attempts in the area of recurrent neural network because of its ability to learn both long and short term dependencies. LSTM proposes gate mechanisms to deal with the problem of gradients as well as to capture the dynamics. It employs three type of gates i.e input gate, forget gate, output gate which work together to provide certain output as well as updates the memory. A schematic of LSTM block is presented in the following figure 2.6. As shown in the figure, the forget gate decides on how much of the past memory to

FIGURE 2.6: Single LSTM unit

retain, the input gate decides the amount of new information to be utilized and the output gate provides the output from the unit. Therefore it is able to retain both long and short term memories by learning what to remember and what to forget. There have been numerous researches in past to modify the structure of LSTM to serve different purposes. For example, in Bidirectional RNN [32] the network utilizes information from both past and future. Clockwork RNN [33] and Phased LSTM [34] can update their hidden states asynchronously. State Frequency Memory (SFM) Network [35] are able to separate dynamical patterns in the frequency domain to offer detailed analysis of temporal sequences. Recently [36] proposed a modification on LSTM referred to as the Gated Recurrent Unit (GRU) where input and forget gates of LSTM are combined into an update gate. There are other methods available in literature such as Echo State Network [37] where the learning task is simplified by selectively learning the non-recurrent weights (needs careful initialization), [38, 39] proposed a powerful Hessian-Free optimization (HF) technique robust to the gradient problems. These deep recurrent networks are built mainly focusing on language modeling hence they are not always well suited for short-term time series problems [40]. In PR-SSM [41] a probabilistic recurrent state space model is proposed for time series and system identification. Currently, there has been a growing interest in the domain of attention mechanism [42–44] as well,

which tries to capture local time-frequency features along with global long-term trend and fuse them to execute a better time series forecasting.

From the literature review above it is clear that there has been numerous works in the field of neural networks (as well as RNN) which revolve around improving supervised machine learning as well as time series forecasting. ANN are known to for their ability to approximate any non-linear functional relationship between input and output from data. The biggest advantage of ANN is its data driven learning capability, that it can learn from any data without any prior knowledge of the input-output relationship and perform a required task. However, ANNs fall short when it comes to handling uncertain, chaotic, imprecise or noisy data. Deep ANN and RNN also fail to provide much interpretability on the relationship between the inputs and outputs. Fuzzy logic (or fuzzy inference system, FIS) has seen huge advent in this direction because of its ability to mimic human like linguistic ability that makes it able to handle uncertainty/ noise and provide interpretability. That is why ANNs and FIS complement each other as ANN brings in its learning capability where FLS provides a comprehensible linguistic interpretation plus they are known to be universal approximator [45–47]. A brief introduction to fuzzy logic is presented in the next section, followed by a detailed discussion on neuro-fuzzy systems, its variances and how it can be useful for time series forecasting.

2.3 Neuro-fuzzy inference systems for time series forecasting

Fuzzy logic system has been in the practise for a long time for its proven advantages over traditional logic in the field of control and modeling. Fuzzy inference system (FIS) (also referred to as Fuzzy Logic System or FLS) is defined as a computer logic system based on fuzzy theories, fuzzy rule base, and fuzzy reasoning. FIS has varied usage in different areas such as classification, pattern recognition, control, robotics and time series modeling etc. The main reason behind the prominence of fuzzy logic is that it can mimic human-like behavior in terms of reasoning and comprehension. In the next section we briefly discuss how fuzzy system works.

2.3.1 Fuzzy Inference System (FIS)

Traditional logic system works with discrete results, for example binary logic works with two states true or false, yes or no. Contrary to traditional logic, fuzzy logic tries to introduce human like reasoning where results are not limited to binary rather they can have linguistic labels to express the uncertainty associated with it. For example, if a person does not know the exact temperature (meaning there is uncertainty associated with the measure), he/she can still use the following labels i.e. hot, warm, comfortable, cold, freezing etc. to express the measure. This is the main principle behind fuzzy logic. Naturally, when the data is noisy/ imprecise in nature, this logic is much more useful (because of the aforementioned linguistic nature) than a traditional model.

In a Fuzzy Inference System, first the crisp inputs (input feature values such as temperature) are fuzzified and turned into linguistic labels, using membership functions. Membership functions can be of different shapes such as triangular, trapezoidal, Pi shaped, Gaussian etc. Then there is a set of If-Then fuzzy rules (usually set from expert knowledge, discussed more in next subsection) which is called fuzzy rule base. Next step is to compute the inference for the fuzzified features with the help of this rule base. Finally, there is a defuzzification layer or a weighted average layer to get a crisp output from the FIS. Figure 2.7 depicts the design of a basic fuzzy inference system.

FIGURE 2.7: Fuzzy logic system

There are primarily two kinds of fuzzy logics as mentioned below,

1. **Mamdani fuzzy model :** Professor Mamdani in 1975 first proposed a model of fuzzy inference system which is known as Mamdani model [10]. In Mamdani model, both antecedent (input side) and consequent part (output side) of the architecture use fuzzy variables, hence to get a crisp output there needs to be some form of defuzzification technique. The most common defuzzification techniques are center of gravity method, bisector method, mean of maximum methods etc.

A Mamdani fuzzy rule has the following form:

If x_1 is A_{1r} AND x_2 is A_{2r} AND $\dots$ AND x_m is A_{mr} then $y = B_r$

where $\mathbf{x} = [x_1, \dots, x_m]^T \in \mathbb{R}^m$ is the input and y is the output. Here, $x_1, \dots, x_m, y$ are fuzzy variables and $A_{1r}, \dots, A_{mr}, B_r$ are linguistic labels s.t. $A_r = [A_{1r}, \dots, A_{mr}]^T \in \mathbb{R}^m$ are the rule antecedent for r^{th} fuzzy rule and B_r is the corresponding fuzzy consequent.

2. **TSK fuzzy model :** Takagi, Sugeno and Kang proposed the TSK fuzzy model [11] in 1985. The main difference between TSK and Mamdani type of fuzzy is in the consequent part. In TSK fuzzy system, crisp variables are used in the consequent, unlike Mamdani system, where it is fuzzy. This eliminates the need for a defuzzification method in TSK type system.

A TSK fuzzy rule has the following form :

If x_1 is A_{1r} AND x_2 is A_{2r} AND $\dots$ AND x_m is A_{mr} then $y = f_r(x_1, x_2, \dots, x_m)$

where $\mathbf{x} = [x_1, \dots, x_m]^T \in \mathbb{R}^m$ is the input and y is the output. Here, $x_1, \dots, x_m, y$ are fuzzy variables and $A_{1r}, \dots, A_{mr}$ is a linguistic label s.t. $A_r = [A_{1r}, \dots, A_{mr}]^T \in \mathbb{R}^m$ is the rule antecedent for r^{th} fuzzy rule and f_r can be any arbitrary function which determines the corresponding fuzzy consequent.

TSK type fuzzy systems have the advantages of better accuracy while handling chaotic and nonlinear time series. On the other hand, Mamdani models are better suited when it comes to interpretation of the fuzzy rule base. In this thesis, we are more interested in achieving better accuracy. Hence, only TSK-type FIS is considered in the rest of the literature review and the proposal of the new neuro-fuzzy systems (Chapter 3 and 4). Having discussed the literature of both neural

network and fuzzy inference systems, it is clear that they have their own strengths as well as shortcomings. The natural progression is to fuse them to reap the benefits of both world. Hence, the following discussion will focus on the incorporation of TSK-type FIS with ANN, which is termed as Neuro-Fuzzy Inference System or NFIS (also known as fuzzy neural network).

2.3.2 Neuro-Fuzzy Inference System (NFIS)

Most neuro-fuzzy inference system implements the fuzzy logic with a five layer architecture as listed below,

- **Input layer** takes in the crisp values from the inputs/ features and transferS them to subsequent fuzzification layer.
- **Fuzzification layer** transforms the crisp input values from different features into fuzzy linguistic variables. To perform this task, it can use membership function of different shapes such as triangular, trape-zoidal, Pi shaped, Gaussian etc.
- **Rule firing/ inference layer** computes the inference with the fuzzified values using an existing fuzzy rule base. For example, Increasing the speed of the fan when temperature is high is a good example of inference computation. This layer is also known as rule firing layer, as we compute the firing strength of different fuzzy rules here.
- **De-fuzzification/ Normalization layer** transforms the fuzzy values of the firing strengths (from last layer) into crisp values using defuzzification techniques. In case the firing strengths are already non-fuzzy (as in TSK fuzzy) then it normalizes them and pass it to the final layer.
- **Output layer** provides the final output after performing a weighted summation of the normalized or de-fuzzified firing strengths from previous layer.

However, the parameters of the neuro-fuzzy system are updated in a similar manner to a neural network using learning algorithms such as backpropagation, Extended Kalman Filter (EKF), Projection Based Learning (PBL) etc. (which will be discussed in the subsequent section). Some of the most prominent works in this field

focus on learning the rule base from scratch (starting from no rule) in a data-driven manner which eliminates the need for expert knowledge. These are termed as self-evolving neuro-fuzzy system and they are known for their data driven structure and parameter learning ability which makes them well suited for online or sequential problems. Next, we discuss some of the most recent and notable self-evolving neuro-fuzzy systems which work in a feed-forward manner (without any recurrence).

2.3.2.1 Feed-forward neuro-fuzzy inference system

In adaptive network based fuzzy inference system (ANFIS)[48] a feed-forward static neural network is presented which incorporates TSK-type fuzzy inference engine to map and track nonlinear functions, which can also be used for identifying nonlinear components in a control scenario. ANFIS is capable of predicting a chaotic time series. GS-EFS [49] proposes a smart evolving fuzzy system which utilizes generalized fuzzy rules in arbitrarily rotated position. The modified version of GS-EFS [50] is more focused towards application oriented time series forecasting problems. SOFNN [51] proposes a self-evolving fuzzy framework with ellipsoidal basis function (EBF) neurons as well as a recursive rule evolving and parameter learning technique. In eTS [52] a potential parameter is defined to quantify the novelty of data points. It utilizes both supervised and unsupervised learning frameworks to recursively learn the model structure and parameters. On the other hand SimpleTS [53] is an extension to eTS, that again uses a recursive method to calculate a scatter parameter to identify the structure. Both eTS and SimpleTS use fuzzily weighted recursive least square methods for parameter update. Recent works such as [54] uses Fuzzy C-Means clustering for structure identification, [55] adopts a cascaded neuro-fuzzy system is proposed that uses least square method for parameter tuning, CEFNS [56] uses a robust correlation-entropy based rule expansion and update criterion and [57] employs multidimensional adaptive wavelet activation membership functions. The major problem with the discussed methods above is they are prone to redundant rule addition with spurious sample posing as novel ones as no rule pruning methods are employed. SAFIS [58] introduces the concept of rule influence which is computed based on the rule's statistical contribution towards output. This influence parameter is utilized to expand its rule base and the least influential rule is pruned. McFIS [59] proposes a novel spherical potential

method for structure learning and it also prunes the non-performing rules from its rule base. PANFIS [60] employs a multi-dimensional membership functions and a statistical contribution approach to grow its rule base. It also proposes rule blending to prune redundant rule if two rules are very similar in nature. GENE-FIS [61] is an advancement over PANFIS with online feature selection ability that improves its performance significantly and also results in a compact and parsimonious rule base. ESAFIS [62] is developed as an extension of SAFIS primarily to handle classification problems (instead of time series and regression problems like SAFIS). MSAFIS [63] is a modified on top of the base framework of SAFIS which uses a stable gradient descent for parameter tuning purpose. SUFIN [64] on the other hand utilizes EKF for parameter update and it also provides a convergence analysis on the error. Its application was shown for prognostic health management plants. PALM [65] incorporates a hyperplane clustering approach to add, prune or merge its fuzzy rules both shown with type-1 and type-2 set up.

All the self-evolving neuro-fuzzy inference systems described above, are in principle feed forward in nature hence when it comes to handling rapidly changing system dynamics, they fall short. Hence, recurrence was introduced in the neuro-fuzzy framework. Next, the discussions will be on different types of recurrent neuro-fuzzy inference systems (recurrent NFIS) and how recurrence is attained in them.

2.3.2.2 Recurrent neuro-fuzzy inference system

The simplest way to bring recurrence is to introduce tap-delay type feedback as introduced in the section 2.2.2. Some of the prominent works in the field of neuro-fuzzy systems (such as ANFIS [48], SAFIS [58], McFIS [59]) while handling a dynamic system use a tap delay type external feedback to bring in past information. However, such mechanism only works when the system is observable and the temporal order is known. The other approach to capture dynamics is to use internal or self-feedback locally or globally in the neuro-fuzzy architecture instead of the external tap delay feedback (hence no need for the system to be observable). There has been numerous researches in the area of recurrent fuzzy neural network as can be found in [66–79]. All of these recurrent architectures can be further classified into two sub-categories which are locally recurrent and globally recurrent systems (depending on the design and type of the feedback).

Locally recurrent NFIS : In local recurrence, the nodes will have self-feedback in either membership layer or in rule layer or in the output layer. In DFNN[66] a time variant neural fuzzy system is introduced. Here no external feedback is used like tap delay networks, rather an internal feedback is used to bring the temporal output information internally and delay elements have been used in the input layer. Here, the output realizes a weighted sum of the fixed number of past values, which increases computational complexity of the network. Also, the number of past values in self-feedback depends on the order of system and may not be suitable for a system with changing characteristics. In RFNN[67] a local type of self-feedback is brought in the membership layer to overcome the aforementioned problem of DFNN. This recurrent network is used in many system identification problems, control and time series predictions. Figure 2.8 represents the locally recurrent neuro-fuzzy inference system as adopted in [67].

FIGURE 2.8: Locally recurrent neural-fuzzy system

In WRFNN[68] a wavelet-based recurrent neural net with an online training method is proposed. This also employs a similar self-feedback like RFNN. In DRFNN[69]

and IRSFNN[70] feedback is present locally in the rule layer in a fully interconnected manner. Due to superior system dynamics and uncertainty handling capability of the recurrent fuzzy neural network, they have been widely used in the literature for various industry and application oriented researches such as [71, 72]. Both utilize a self-feedback in the membership layer. In [73–75], self feedback was used in the rule layer instead. Authors of [76] incorporated the self-feedback both in the membership and the rule layer however with no connection between two layers.

Globally recurrent NFIS : Another way to bring recurrence is to use feedback in the network with a global approach where delayed information from one layer will be fed to another layer; thus making the feedback a global one in nature; unlike local feedback where a feedback is restricted to a single layer. In RSONFIN[78] a global type recurrence is presented where the delayed firing strength of a fuzzy rule is provided as feedback into the membership layer.

FIGURE 2.9: Globally recurrent neural-fuzzy system

Figure 2.9 represents the globally recurrent neuro-fuzzy inference system architecture as proposed in [78]. In TRFN[79] a feedback of global nature (inter-layer, interconnected) is adhered from the rule layer to its membership layer as inputs. This network proposes a genetic algorithm for the training. Memory neuron network [25] although not a fuzzy inference system adapts a recurrence with its memory filters (as described in the previous section) which falls under both local and global category i.e. local because it has self feedback in all the nodes and global because it has inter-layer connection of the memory elements.

Having discussed the network based (neural and neuro-fuzzy models) literature in details, it is clear that there are multiple research gaps which require further investigation to improve the field of time series forecasting. Most of the recurrent neuro-fuzzy systems are offline (batch) in nature therefore they are not well suited for online time series tasks. Besides, presence of feedback in a recurrent system, increases computational complexity and introduces stability issues during the learning process. Secondly, memory neurons or the regular self-feedback takes into account of all past instances or a single past instances hence, in real-world time series problems with changing system dynamics they may fall short. Thirdly, there is also little to no literature which addresses the issue of estimating the temporal order in an online setting. Finally none of the network based models discussed so far address the issue of ‘lag’ which exists in the predicted sequence. These were the motivations behind this thesis as described more elaborately in section 1.1. In the final part of this literature review section, the discussion will concentrate on various learning algorithms for network based time series forecasting.

2.4 Learning algorithms for time series models

To learn from the available data and train a neural (or neuro-fuzzy) network in a supervised manner, there needs to be a learning algorithm. One of the most prominent learning algorithms which rejuvenated the research in the field of neural network is backpropagation. As shown in the literature[80, 81], backpropagation can be a very powerful tool for most feed-forward networks. However for time series data a regular backpropagation algorithm would not work (as time series requires recurrent network). For such recurrent scenarios a backpropagation through time (BPTT) algorithm is more suitable. Among other learning algorithms which are

also capable of sequential learning, some of the most important ones are Extended Kalman Filter (EKF) and Projection Based Learning (PBL) which will be briefly reviewed next, in the last part of our literature review.

Backpropagation Through Time (BPTT) : With backpropagation algorithm there is two-directional flow of information; first the signal is propagated from input to output side. The network calculates an output (from the inputs) according to the network parameters. There is a desired output considering this algorithm is for supervised learning scenarios. The difference between the predicted and the target output is termed as prediction error ($e(k)$). A cost function is then associated with this error, which can have the following form for S number of training samples,

$$J(k) = \sum_{k=1}^s e^2(k) \quad (2.5)$$

here global quadratic error is used as the cost function. The main goal of backpropagation (as pointed out in [81]) is to reduce the cost function with each training iteration. To achieve that first the gradient of the cost function is computed with respect to the network parameters. Then the optimal parameters are found by going in the negative direction of that gradient in the parameter space.

$$w(k+1) = w(k) - \eta \cdot \frac{dJ(k)}{dw} \quad (2.6)$$

here w is the weight to be updated and η is the learning rate. Depending on η one can modify the rate at which the training takes place (bare in mind, with higher learning rate the training is quick but there is a high chance of overshooting and missing the global optima, whereas with a lower learning rate the optima can be reached but the training becomes slower). The name backpropagation here symbolizes the reverse/ back traversal of error from output side to input side

In case of a recurrent neural network where feedback is present, a regular backpropagation is not suitable. In such cases, the network needs to unfold in time because the amount of update required to minimize the cost function (i.e. global squared error) is also dependent on past values of the parameters. This modified version of BP is referred to as Backpropagation Through Time. In principal, BPTT makes identical copies of the recurrent neural network, which is put together to provide a feed-forward type architecture; this is referred to as unfolding of a RNN. Please

note that during the unfolding the weights remain fixed, once the network has unfolded completely then only the network weights are updated. The backpropagation through time and how the RNN learns by unfolding in time, is explained in detail in the literature of [82, 83]. As mentioned, BPTT algorithm works on the assumption that the weights remain fixed throughout the unfolding. In order to allow online real-time learning (or RTRL [26, 83]) of RNN, it is necessary to relax the assumption that the weights are fixed and actually make them modifiable while the learning is on. RTRL has the added advantage that no epoch boundaries need to be defined for training the network, leading to better real time learning. However, the major problem with RTRL is that, it is much more computationally complex than a BPTT. For online compliant learning, shallow neural networks and neuro-fuzzy systems also employ Extended Kalman Filter and Projection Based Learning methods as briefly described below.

Extended Kalman Filter (EKF) : EKF is one of the other heavily used methods for training RNNs. This became popular because of its fast convergence to the optimal solution. With EKF approach first the RNN is formulated in terms of state-space equations. The goal of the Kalman filter (which is essentially a set of equations) is to provide with an accurate estimation of the aforementioned state vector. To execute that, in EKF two different sets of equations are used. The first set of equations deal with the time update whereas the second set of equations are for measurement purpose. This method was first proposed in the original work by Kalman [84] and later in other literature such as [85] etc. EKF although fast, tends to be computationally very expensive.

Projection Based learning (PBL) : When it comes to sequential (online) learning for a network model (for the purpose of time series forecasting), a regular back-propagation [82] or Stochastic Gradient Descent [86] method may not be suitable as they are not equipped to train RNN. As discussed above BPTT, RTRL or EKF can be used for this purpose, however all of them can be quite computationally expensive and slow in nature. Hence, we take a look at the recently proposed Projection Based Learning or PBL [87]. PBL employs an one-shot least square solution weight update method which is fast and easy to implement. PBL is also well known for its immunity against the problems of vanishing and exploding gradients (as there is no requirement for gradient calculation here). This method is

improved and then utilized throughout this thesis (described in detail in the third chapter).

Summary : In this chapter we have discussed and reviewed in detail different aspects of time series forecasting, its statistical modeling approaches, neural network based models, neuro-fuzzy models as well as different learning mechanisms along with their strengths, weakness and the motivations behind them. Throughout the discussion, I have pointed out several research gaps that motivated this thesis (as explained in detail in section 1.1). In the next chapter, a Spatio-temporal neuro-fuzzy inference system is proposed which is self-evolving and recurrent in nature plus it can handle online time series. The stability of a recurrent neuro-fuzzy network is also explored and proved in the following chapter.

Chapter 3

Spatio-Temporal Fuzzy Inference System (SPATFIS)

Time series datasets are often non-stationary and dynamic in nature. To capture these intrinsic dynamical characteristics of the underlying distribution, in this chapter a novel Spatio-Temporal Fuzzy Inference System (SPATFIS) is proposed. One of the prime features of SPATFIS lies in the inclusion of memory type neurons which incorporate both spatial and temporal information of the sequences with a dual recurrent structure. SPATFIS also proposes a new self-adaptive learning mechanism to add, eliminate and unify its fuzzy rules in order to attain a parsimonious rule base. Furthermore, stability is rigorously inspected and proved using Lyapunov's Input to State Stability theorem. SPATFIS also adopts a projection based learning algorithm to update its parameters in a sequential manner thereby it is capable of online time series forecasting. Besides, a Recurrent Interval Type 2 Fuzzy Inference System (or RIT2FIS) and a Meta-cognitive Recurrent Fuzzy Inference System with Memory Neurons (McRFIS-MN) are also developed as extensions to SPATFIS. However, this chapter will focus solely on SPATFIS ¹

---

¹The contents of this chapter are published in the following journals and conferences, [88] Subhrajit Samanata et.al "A novel spatio-temporal fuzzy inference system (spatfis) and its stability analysis." Information Sciences (2019), [89] Subhrajit Samanta et.al "A meta-cognitive recurrent fuzzy inference system with memory neurons (mcrfis-mn) and its fast learning algorithm for time series forecasting." Symposium Series on Computational Intelligence (SSCI 2018) [90] Subhrajit Samanta et.al "RIT2FIS: a recurrent interval type 2 Fuzzy Inference System and its rule base estimation." International Joint Conference on Neural Networks (IJCNN 2019)

3.1 SPATFIS architecture

In the proposed framework, memory type neurons[25] are incorporated in the fuzzy-inference system to bring in recurrence into the granular level of the architecture, thereby making SPATFIS capable of capturing system dynamics deftly.

3.1.1 Memory Neuron (MN)

With regards to time series forecasting, temporal mapping can be thought of as the relationship between input and output in the time domain. For example, how does past behaviour affects future can be regarded as a temporal relationship i.e. $y(k) = f_T(y(k-1))$ is a temporal mapping. On the other hand, spatial mapping refers to a cross-sectional relationship between an input and output of a system both at same time instance i.e. $y(k) = f_S(x(k))$ can be thought of as a spatial mapping. Memory neurons are built with these two components. First one is spatial in nature which maps the input vector to an output. Second one is the temporal one which uses self-feedback to retain past information and bring recurrence into the system. Figure 3.1 provides a schematic for realization of such neurons.

FIGURE 3.1: Realization of memory neuron

If input to the MN is $\mathbf{v}(k) \in R^P$, then the **spatial mapping** $x(k) \in R^1$ and **temporal mapping** $x^m(k) \in R^1$ (k represents the time instance) are defined as,

$$x(k) = f(\mathbf{v}(k)) \quad (3.1)$$

$$x^m(k) = \alpha x(k) + (1 - \alpha)x^m(k-1) \quad (3.2)$$

where $f(\cdot)$ is an arbitrary function executing the spatial functionality and z^{-1} represents the 1 unit time lag. α is the memory neuron parameter controlling the degree of temporality and is set between $[0,1]$ to ensure a bounded memory output which does not explode. From equation 3.2 it is apparent high value of α would mean more emphasis on recent information than long past whereas a lower value of α indicates that more past information is being considered.

Observing the recursive nature of equation 3.2 one can assimilate that the temporal mapping retains the effect of all past instances with exponentially decreasing coefficients. It can be easily shown expanding equation 3.2 as below,

$$x^m(k) = \alpha x(k-1) + \alpha(1-\alpha)x(k-2) + (1-\alpha)^2 x^m(k-2) \text{ and so on.}$$

As the memory neurons are able to retain all past information, there is no need to feed all the past instances of actual value $y(k)$ or exogenous factor $\mathbf{u}(k)$ to SPATFIS separately. Hence, the time series can be modeled using only the augmented input vector, $\mathbf{v}(k) = [\mathbf{u}(k), y(k-1)]^T \in R^{P+1}$ as given below,

$$\hat{y}(k) = h(\mathbf{v}(k)) = h(\mathbf{u}(k), y(k-1)) \quad (3.3)$$

where, $\hat{y}(k)$ is the predicted output with $h(\cdot)$ as the non-linear function representing the time series data (which is to be learned from the data in a supervised manner).

3.1.2 Layer wise operations

The architecture of the proposed Spatio-Temporal Fuzzy Inference System (SPATFIS) is shown in Figure 3.2. It is a five-layer neuro-fuzzy network, where the first layer is the input layer, the second layer is the fuzzification layer, the third layer is the rule firing layer, the fourth layer is the defuzzification layer and the last layer is the output layer. The input layer takes $\mathbf{u}(k) \in R^P$ and one lagged output $y(k-1)$ as inputs and the augmented input vector is represented as $\mathbf{v}(k) = [\mathbf{u}(k), y(k-1)]^T \in R^{\hat{P}}$ where $\hat{P} = P + 1$. SPATFIS employs the memory neurons (denoted by solid circles in figure 3.2) in input and defuzzification layers (i.e. crisp layers). Hence, the input layer has $\hat{P}$ memory neurons whereas the defuzzification layer has R memory neurons corresponding to the R number of rules in the fuzzy rule base.

FIGURE 3.2: Network architecture of SPATFIS

Input layer: Each of the nodes in the input layer is assigned a memory neuron to exercise the spatial and temporal mapping in order to capture the input dynamics locally. For the i^{th} memory neuron in the input layer (layer designated by 'I') the input is $\mathbf{v}_i(k)$ and the corresponding **spatial and temporal outputs** are respectively,

$$x_i(k) = f_i^I(\mathbf{v}_i(k)) = \mathbf{v}_i(k), \quad i = 1, 2, \dots, \hat{P} \quad (3.4)$$

$$x_i^m(k) = \alpha_i^I x_i(k-1) + (1 - \alpha_i^I) x_i^m(k-1) \quad (3.5)$$

where α_i^I is the memory neuron parameter ($0 \leq \alpha_i^I \leq 1$).

Fuzzification layer : The first sample (i.e. $k = 1$) of the time series sets the first rule in the SPATFIS rule base. Afterwards the model expands (and prunes when required) its fuzzy rule base with the help of a novel self-adaptive mechanism as discussed in the following section. Let's assume at k^{th} time instance the network has R number of fuzzy rules. Each of the crisp outputs from the input layer MNs, i.e. $x_i(k)$ and $x_i^m(k)$, are transformed into fuzzy variables using the aforementioned fuzzy rules. Here, Gaussian membership functions are used to represent the fuzzy rules. The r^{th} (where $r = 1, 2, \dots, R$) membership value for $x_i(k)$ and $x_i^m(k)$ are,

$$\phi_{ri}(k) = \exp \left[-\frac{1}{2} \left[\frac{(x_i(k) - \mu_{ri})}{\sigma_r} \right]^2 \right] \quad (3.6)$$

$$\phi_{ri}^m(k) = \exp \left[-\frac{1}{2} \left[\frac{(x_i^m(k) - \mu_{ri}^m)}{\sigma_r} \right]^2 \right] \quad (3.7)$$

where μ_{ri} and μ_{ri}^m are the centers of the r^{th} membership functions for $x_i(k)$ and $x_i^m(k)$ respectively and σ_r is the width.

Rule firing layer : The R regular neurons in the rule firing layer perform the rule aggregation (to compute inference) using a t-norm operation (which translates into multiplication) on all the fuzzy values from previous layer ($\phi_{ri}(k), \phi_{ri}^m(k)$ where $r = 1, 2, \dots, R$ and $i = 1, 2, \dots, \hat{P}$). So the firing strength corresponding to the r^{th} rule is,

$$F_r(k) = \Pi_{i=1}^{\hat{P}} (\phi_{ri}(k) \cdot \phi_{ri}^m(k)) \quad (3.8)$$

Normalization layer : In this layer, each of the R rules is assigned a Memory Neuron to trace the inner dynamics deftly utilizing the spatial and temporal mapping. For the r^{th} MN in this layer (layer designated by ' R' ') the input is

$[F_1(k), F_2(k), \dots, F_R(k)]^T$. Hence the corresponding **spatial and temporal outputs** are respectively defined as below,

$$\bar{F}_r(k) = f_r^R(F_1(k), \dots, F_R(k)) = \frac{F_r(k)}{\sum_{r=1}^R F_r(k)} \quad (3.9)$$

$$\bar{F}_r^m(k) = \alpha_r^R \bar{F}_r(k-1) + (1 - \alpha_r^R) \bar{F}_r^m(k-1) \quad (3.10)$$

where α_r^R is the memory neuron parameter ($0 \leq \alpha_r^R \leq 1$).

Output layer : The single node present in the final layer executes a weighted summation of the outputs from the previous layer. The output from this layer is the final prediction $\hat{y}(k)$ and it is given as,

$$\begin{aligned} \hat{y}(k + \tau) &= \sum_{r=1}^R w_r \cdot \bar{F}_r(k) + \sum_{r=1}^R w_r^m \cdot \bar{F}_r^m(k) \\ &= \mathbf{w}(k)^T \cdot \mathbf{F}(k) \end{aligned} \quad (3.11)$$

Dropping the k notation of time for the clarity of expression, weight vector $\mathbf{w} \in R^{2Rx1}$ is defined as, $\mathbf{w} = [w_1 \dots w_R, w_1^m \dots w_R^m]^T$ and normalized firing strength vector $\mathbf{F} \in R^{2Rx1}$ is,

$$\mathbf{F} = [\bar{F}_1 \dots \bar{F}_R, \bar{F}_1^m \dots \bar{F}_R^m]^T$$

. In the following sections the structure identification and parameter learning process of SPATFIS are explained in details.

3.2 Learning in SPATFIS

SPATFIS learns in two ways. They are structure learning and parameter learning. The structure learning process involves addition of rules (and deletion too when required) whereas parameter learning is where SPATFIS updates its parameters. Both of the proposed learning methods are sequential (online) in nature.

3.2.1 Structure learning

SPATFIS starts with no rules. Then it adds rules in a data-driven manner. It can prune rules too if they are deemed redundant (more about rule pruning in later sections). The main idea behind the rule growing mechanism is that when a

sample (data-point) is significantly knowledge rich then there might be a need for a new fuzzy rule to accommodate the new knowledge of this sample into the model. This is to ensure that the uncharted distribution of data point can be explored with the newly added rule. Most adaptive algorithms in the literature use error based criteria [52, 58], spherical potential method [59] or statistical contribution [60] or clustering method [91] to determine if a new rule should be added to the existing fuzzy rule base. SPATFIS proposes a more intuitive and computationally inexpensive ϵ -completeness method, to determine the novelty (knowledge richness) of a sample. If the novelty criterion is met then a new fuzzy rule is added to the existing rule base model.

To determine the novelty of an incoming sample SPATFIS utilizes the spatial membership values of that sample. Membership value of a sample corresponding to a certain fuzzy rule represents the degree of similarity between the sample and that rule. Hence a higher membership value indicates a higher similarity and vice versa. A sample far away from a fuzzy rule center results in a low membership value for that rule. If the membership value is low for all rules it will clearly indicate that the sample is not (similar to) encompassed by any of the rules in the rule base, hence it should be regarded as a novel sample. The novelty of a sample is judged from the $3\text{-}\sigma$ rule to prevent the use of a user-defined threshold.

If the membership value (spatial) of the k^{th} sample for r^{th} rule and i^{th} feature (i.e. $\phi_{ri}(k)$) is less than 0.0110, then with 99.7% confidence it can be stated that the sample is at least $3\sigma_r$ (three times of its width) away from the center along i^{th} dimension. Note that this statement holds under the unimodal normal distribution assumption. Now if the sample is not encompassed properly by any of the R existing rules then we conclude the sample to be novel. Hence we can formulate the novelty criterion for $i = 1 \cdots \hat{P}$ and $r = 1 \cdots R$ as given,

If $\max_r(\min_i(\phi_{ri}(k))) < 0.0110$ then add a new Rule

The use of min operator here is to overcome the curse of dimensionality suffered from the dot product operation. Let us take a working example to understand the intuition behind this better.

Example case : In figure 3.3 we have a single feature, two rule fuzzy inference system with centers at -2 and 3 respectively. For a new sample with feature value 0.5, it can be observed that the maximum of the membership values between

$\phi_1(k), \phi_2(k)$ is less than 0.0110. Hence it can be concluded that the sample has proven its novelty and met the rule growing criterion. Hence we add a rule as shown in figure 3.4.

FIGURE 3.3: Existing Fuzzy Rule Base

When a new rule is added, then corresponding center, width of the rule and the consequent weights (to the output node) need to be initialized as well. Center and spread of the new rule are set as follows,

$$\begin{aligned} \mu_{(R+1)i} &= x_i(k) \\ \mu_{(R+1)i}^m &= x_i^m(k), \quad i = 1, 2, \dots, \hat{P} \\ \sigma_{R+1} &= \sigma_{nr} \end{aligned} \quad (3.12)$$

where nr represents the nearest existing fuzzy rule from the new data point.

In figure 3.4 it can be seen that the newly added rule center is initialized with 0.5 in accordance with the above equations.

With the newly added rule center (i.e. $\mu_{(R+1)i}$) and spread (i.e. σ_{R+1}), the new rule firing strength $\bar{F}_{R+1}(k)$ can be computed for the current sample. We assign the memory and memory neuron parameter of the nearest existing rule to the new rule such that $\bar{F}_{R+1}^m(k) = \bar{F}_{nr}^m$ and $\alpha_{R+1} = \alpha_{nr}$. This allows fast convergence because the closest rule is supposed to capture the same data trend. Rule addition leads to two more output weights w_{R+1} and w_{R+1}^m (corresponding to $\bar{F}_{R+1}(k)$ and $\bar{F}_{R+1}^m(k)$) which needs to be initialized for completeness of system equations. The initialization of the new consequent weights should minimize the prediction error.

FIGURE 3.4: Rule Addition for the new Sample

Prediction error at k^{th} time instance before the rule addition (with R rules) was,

$$e(k) = y(k + \tau) - (\sum_{r=1}^R w_r \cdot \bar{F}_r(k) + \sum_{r=1}^R w_r^m \cdot \bar{F}_r^m(k)) \quad (3.13)$$

To compensate for this error and drive the total error close to zero, the new weights have to satisfy the following criteria,

$$w_{R+1} \cdot \bar{F}_{R+1}(k) + w_{R+1}^m \cdot \bar{F}_{R+1}^m(k) = e(k) \quad (3.14)$$

The least square solution of the above equation can be found using singular value decomposition method. Please note the new weights are introduced only when a new rule is added hence the weight initialization process is allowed to make use of only the current sample (adhering to the principal of online/sequential learning). Hence the Moore-Penrose Pseudo inverse [92] of $[\bar{F}_{R+1}(k); \bar{F}_{R+1}^m(k)]$ designated by the $\dagger$ sign provides the best solution as,

$$[w_{R+1}; w_{R+1}^m] = [\bar{F}_{R+1}(k); \bar{F}_{R+1}^m(k)]^\dagger \cdot e(k) \quad (3.15)$$

Remark 1: It can be concluded that the new weights (corresponding to the newly added rules) are initialized in such a way that the prediction error for that sample is minimized, to fully utilize the localization property of the rules.

3.2.2 Parameter learning

For a sample which is not very novel, adding an extraneous rule will lead to obvious over-fitting so instead the knowledge in offer (from the sample) is utilized to update all the consequent weights. Hence, when the rule addition criterion fails i.e. $\max_r(\min_i(\phi_{ri}(k))) \geq 0.0110$, SPATFIS adopts a projection based learning (PBL) method to update the consequent weights either in one shot or incrementally. On the other hand, the membership parameters (center and spreads of the fuzzy rules) are updated only for winning rules with each sample.

One shot PBL for weights update: SPATFIS prediction is given by $\hat{y} = \mathbf{w}^T \cdot \mathbf{F}$ from equation 3.11 with the normalized firing strength vector $\mathbf{F} \in R^{2RX1}$ and consequent weight vector $\mathbf{w} \in R^{2RX1}$. A cost function is defined as the sum of squared prediction errors for all the the training samples the network has encountered yet (i.e. till k^{th} time instance) and the goal is to find the optimal weights $\mathbf{w}^*$ that minimizes the cost function.

$$J(\mathbf{w}) = \frac{1}{2} \sum_k e^2 = \frac{1}{2} \sum_k (y - \hat{y})^2 = \frac{1}{2} \sum_k (y - \mathbf{w}^T \cdot \mathbf{F})^2 \quad (3.16)$$

$$\mathbf{w}^* = \text{argmin} J(\mathbf{w})$$

where y is the actual target output. Fermat's Theorem suggests, for $J(\mathbf{w})$ to have a minima at $\mathbf{w}^*$, its derivative needs to be zero. Hence,

$$\frac{dJ(\mathbf{w})}{d\mathbf{w}}_{\mathbf{w}=\mathbf{w}^*} = 0 \quad (3.17)$$

Solving equation 3.17 (followed by rearranging in next step) will lead to following,

$$\sum_k (-y \cdot \mathbf{F}^T + \mathbf{w}^T \cdot \mathbf{F} \cdot \mathbf{F}^T) = 0 \quad (3.18)$$

$$\sum_k \mathbf{w}^T \cdot \mathbf{F} \cdot \mathbf{F}^T = \sum_k y \cdot \mathbf{F}^T$$

Transposing both side we get,

$$\sum_k \mathbf{F}^T \cdot \mathbf{F} \cdot \mathbf{w} = \sum_k \mathbf{F} \cdot y \quad (3.19)$$

$$A\mathbf{w} = B$$

Here, matrix $A \in R^{2RX2R}$ consists of elements a_{rr^*} and $B \in R^{2RX1}$ of b_r where,

$$\begin{aligned} a_{rr^*} &= \sum_k F_r \cdot F_{r^*} \\ b_r &= \sum_k F_r \cdot y, \quad r, r^* = 1, 2, \dots, 2R \end{aligned} \quad (3.20)$$

Solution to the equation 3.19 is the optimal weight for the consequent paths and is obtained by,

$$\mathbf{w}^* = A^{-1} \cdot B \quad (3.21)$$

The A matrix is computed such a way that if it is singular then it will be transformed to a matrix of small values (i.e. $a_{rr^*} = \min(0.001, \sum_k F_r \cdot F_{r^*})$) so that it is not singular any more and the matrix inverse operation can be carried out.

Lemma 1 : The cost function utilized here is convex and the least square optimal solution can be attained utilizing the one shot PBL method. This lemma is presented in appendix A.

Center and spread update: In a fuzzy inference system, the fuzzy rules should ideally be a true representation of the data samples i.e. if there are R number of rules then they should represent R cluster of data samples. To ensure this, membership parameters (center and spread of rules) are also updated with each of the samples during parameter update. This strategy refines the coverage of fuzzy rule base to warrant the ϵ -completeness of fuzzy rules. These parameters are updated only for the winning rule where a winning rule is defined as the nearest rule from the current sample. This winning rule strategy is engaged to ensure that catastrophic forgetting does not occur. If all rules were allowed to update their membership parameters, a spurious sample could have changed all the parameters to an extent where previous knowledge learned (i.e. shape and position of the clusters) would be lost, causing catastrophic forgetting.

To update the center of the winning rule, a recursive k-means clustering inspired center of gravity method is incorporated. When a certain rule is identified as the winning rule then that rule center is updated with the location of the new sample as given by,

$$\mu_{r,i}(k+1) = (k \cdot \mu_{r,i}(k) + x(k)) / (k+1) \quad (3.22)$$

where r is the winning rule. The update method for the spread of the winning rule is little different as the spread should be updated in such a fashion that the winning rule is able to encompass the new sample by either increasing or decreasing its variance. The same is provided here,

$$\sigma_r(k+1) = \left(\sigma_r^2(k) + \beta_k \cdot \|x(k) - \mu_{ri}(k)\|^2 \right)^{\frac{1}{2}} \quad (3.23)$$

where β_k acts as a weight for the new sample. β_k sets itself in such a manner that the spread is increased if the sample is far from the rule and decreased if it's closer to the rule. Hence the rate is kept positive ($+\beta_k$) when $\|x(k) - \mu_{ri}(k)\| \geq 2\sigma_r$ (i.e. far away from the winning rule center) or else (i.e. closer to the winning rule center) it is kept as negative ($-\beta_k$). Over time, the fuzzy widths should attain stable values and to do so the weight β_k needs to decrease smoothly. We have adopted an exponential decay for the weight with $\beta_k = e^{-0.1 \cdot k}$. A regulating factor is also set at 0.1 empirically to slow the decay.

3.2.3 Rule reduction

A certain fuzzy rule will be considered redundant in the following two cases,

- If a rule fails to encompass a certain number of consecutive samples properly leading to negligible contribution towards the output, then that rule is considered redundant and pruned using Rule Elimination.
- If two fuzzy rules are found to be very similar in nature i.e. their centers are very close with almost identical spreads than any one of the rule is regarded as redundant. These two rules are unified to form one rule.

Rule elimination: A spurious sample can pose as a novel knowledgeable sample and increase the number of fuzzy rules using the rule addition criteria. To make sure that such rules are pruned from the final optimal rule base, rule elimination method is employed. The average membership value (spatial) is computed for each of the fuzzy rules for every N_{el} samples. If the same is found to be less than 0.0110 (inspired from the 3- σ rule of normal distribution) for a certain rule then that rule is considered as an insignificant one and discarded.

If $\frac{1}{N_{el}} \sum_{k=1}^{N_{el}} (\min_i(\phi_{ri}(k))) \leq 0.0110$ then discard r^{th} rule

Lemma 2: Redundant rule elimination does not effect the firing strength of the other rules. This lemma is presented in appendix A.

Rule unification: If SPATFIS finds two almost identical fuzzy rules in its rule base then to reduce the number of total rules, these two rules are merged together to form one unified rule. Similarity index (similar to [93]) for p^{th} and q^{th} rules $S(p, q)$ is defined as, $S(p, q) = \frac{1}{\hat{P}} \sum_{i=1}^{\hat{P}} \exp(-|\mu_{pi} - \mu_{qi}| - |\sigma_{pi} - \sigma_{qi}|)$

If $S(p, q) \geq E_{sim}$ then merge p^{th} and q^{th} rules

The similarity threshold E_{sim} is set to 0.9 (empirically) throughout all the experiments. Any smaller value will lead to frequent merging of rules and eventually to under-training (i.e. poorer prediction) hence to ensure only highly similar rules are unified we keep a high threshold of 0.9. This condition is checked for every N_{el} samples and whenever satisfied the, p^{th} and q^{th} rules are merged to form rule M with following parameters, $\mu_{Mi} = (\mu_{pi} + \mu_{qi})/2$ and $\sigma_{Mi} = (\sigma_{pi} + \sigma_{qi})/2$.

Lemma 3: Similar Rule unification does not effect the overall predictive performance of SPATFIS. In the appendix A this lemma is provided.

3.3 Stability analysis in SPATFIS

With any recurrent system stability can be a critical issue. Hence it is of high importance to explore and analyze the stability of SPATFIS (as it also induces recurrence with its memory neurons). We have utilized Lyapunov Input to State Stability theorem to show that the average normalized prediction error over infinite number of time steps (samples) is upper bounded. Hence SPATFIS is proved to be Lyapunov Input to State stable (or ISS). Please refer to Appendix A for a detailed analysis and the proof of stability for SPATFIS (during the learning phase).

3.4 Performance evaluation of SPATFIS

In this section, SPATFIS is employed on different kinds of problems to evaluate its performance in terms of speed, accuracy and size of the rule base. In the

first category, performance comparison is carried out on benchmark time-series forecasting problems such as Mackey Glass [94] and Box Jenkins problem[1]. In the next category, SPATFIS is employed on two real-world problems (i.e. wind speed and stock price prediction). Six state-of-the-art neural fuzzy methods are built in MATLAB 2016 environment on an Intel Six-core 3.2 GHz, 16-GB RAM Windows system. The six neuro-fuzzy methods picked for the experiments are eTS [52], SimpleTS [53], SAFIS [58], McFIS [59], PANFIS [60] and GENEFIS [61] (these are already reviewed in the literature review). They are picked here as they are popular for their self-evolving and online learning capability. The experimental results ensure numerical consistency (followed by a statistical analysis to demonstrate SPATFIS's significant superiority). The performance metrics used to evaluate the efficacy of SPATFIS are,

- Size of fuzzy rule base (#Rules)
- Achieved accuracy (Test NDEI / Test RMSE)
- Training time (CPU time in seconds)

Size of the rule base determines the computational complexity of the model. Root Mean Square Error (RMSE) or Non-Dimensional Error Index (NDEI) are performance indices for accuracy. Root Mean squared Error (RMSE) is defined as the square root of the average of the squared prediction errors,

$$RMSE = \sqrt{\frac{1}{S} \sum_S (y(k) - \hat{y}(k))^2} \quad (3.24)$$

with S as the number of samples. NDEI is nothing but the ratio of RMSE and the standard deviation of the dataset.

$$NDEI = \frac{RMSE}{S.D} \quad (3.25)$$

It is used for performance evaluation on the MG problem as per the standard practise.

3.4.1 Standard benchmark problems

The first set of experiments are performed with standard benchmark problems. Please note that for further statistical study ranks can be assigned based on training time to compare the fastness of the models (refer Appendix A).

3.4.1.1 Mackey-Glass problem

The chaotic long term time series data is obtained from a differential equation as proposed in [94].

$$\frac{dy}{dk} = \frac{0.2y(k - \bar{\tau})}{1 + y^{10}(k - \bar{\tau})} - 0.1y(k) \quad (3.26)$$

With MG problem the goal is to estimate the 85 instances ahead future value $y(k + 85)$ based on past and current values, i.e. $y(k - 18), y(k - 12), y(k - 6), y(k)$. Among the 3500 samples first 3000 were used for training and remaining 500 were used for testing the performance of SPATFIS. During training, the network is only allowed to learn from one sample at a time to adhere to the principal of true sequential learning (with the one shot PBL). Presence of memory neuron eliminates the need for feeding all the 4 delayed values, rather $y(k - 1)$ is enough for good prediction as the memory neurons retain the required information. The accuracy is measured using the non-dimensional error index (ratio of test RMSE and the standard deviation of the test data or NDEI). Table 3.1 provides the measures of performance i.e. rule numbers, training time (in s) and test-NDEI. From this table it is observed that SPATFIS outperforms the other approaches both in terms of training time and rule base size while obtaining good accuracy.

TABLE 3.1: Performance of SPATFIS on MG problem

Problem	Network Model	#Rules	Test NDEI	Time(s)
MG	eTS [52]	09	0.260	3.1
	SimpleTS [53]	11	0.320	4.4
	SAFIS [58]	08	0.360	5.5
	McFIS [59]	10	0.250	8.2
	PANFIS [60]	15	0.160	7.5
	GENEFIS [61]	12	0.105	8.7
	SPATFIS	06	0.011	0.5

3.4.1.2 Box-Jenkins problem

This classic problem deals with the prediction of one time step ahead CO_2 emission rate based on input oxygen flow rate in the Box-Jenkins furnace. The details of this time series prediction problem can be found in [1]. Note, $y(k)$ is the CO_2 concentration and $u(k)$ is the gas flow rate. Here, the input to SPATFIS is both $y(k-1)$ and $u(k)$ for prediction of $y(k)$. Memory neurons take care of the required lags. Based on the problem guidelines, the first 200 samples are used for training and the remaining 90 samples are used for testing. It can be observed from the numeric in table 3.2 that SPATFIS trains much quicker and still achieves better test RMSE with a smaller rule base than its peers.

TABLE 3.2: Performance of SPATFIS on BJ problem

Problem	Network Model	#Rules	Test RMSE	Time(s)
BJ	eTS [52]	08	0.049	0.7
	SimpleTS [53]	07	0.055	0.8
	SAFIS [58]	09	0.071	1.5
	McFIS [59]	12	0.045	0.4
	PANFIS [60]	11	0.070	0.5
	GENEFIS [61]	05	0.050	0.6
	SPATFIS	04	0.045	0.3

3.4.2 Real-world problems

In the second category, the superior performance of SPATFIS is demonstrated with two real-world problems, to show its suitability in practical scenarios where data contains high uncertainty and impreciseness.

3.4.2.1 Wind speed forecasting

As of late there has been a noteworthy improvement in the field of wind power due to its potential as a sustainable source of green energy. The energy generated from the wind machines has uncertain and volatile characteristics due to the stochastic

nature of climate conditions ([95]). Hence, there is an obvious need for good forecasting of the wind speed for efficient wind power generation. Taking into account the statistical relationship between wind speed and wind course, both of these variables are considered in SPATFIS for the one step ahead speed prediction problem. The wind dataset is obtained from Iowa (USA) Department of transport at the location, Washington and contains noise and impreciseness from sensory measurements. The data, gathered over a period of February 1, 2011, to February 28, 2011, is downloaded from the website.² The sampling period of the data was 10 min. From the averaged hourly data 2 features (speed and direction of the past hour) are extracted. The training dataset is composed of 500 samples and the test set has 100 instances. In Table 3.3 the performance of SPATFIS is compared with the aforementioned neuro-fuzzy systems. The result shows that SPATFIS attains better accuracy with smaller rule base in a shorter time.

TABLE 3.3: Performance of SPATFIS on wind speed problem

Problem	Network Model	#Rules	Test RMSE	Time(s)
Wind	eTS [52]	09	0.380	0.5
	SimpleTS [53]	11	0.394	0.4
	SAFIS [58]	16	0.376	0.7
	McFIS [59]	10	0.230	0.9
	PANFIS [60]	16	0.190	1.0
	GENEFIS [61]	07	0.153	0.3
	SPATFIS	06	0.140	0.2

3.4.2.2 Stock price forecasting

The forecasting ability of SPATFIS in a real-world nonlinear, uncertain and volatile situation is going to be exhibited in this subsection with stock price forecasting problem. The stock related information of Google is obtained from Yahoo finance³ for a time span of 6 years from 19-August-2004 to 21-September-2010 (as in [96]). The stock price can be highly unpredictable and uncertain in a manner which does not follow any clear pattern or trend. To examine if SPATFIS can perform well

²<http://mesonet.agron.iastate.edu/request/awos/1min.php>

³<https://finance.yahoo.com/>

with this problem, the dataset is partitioned into training and testing sets of 1529 and 900 sample sizes respectively as per standard practice. One day-ahead price of the stock is estimated based on the previous day's closing price. Learning takes place in a sequential manner where the samples are fed one by one and the test results are produced in table 3.4. SPATFIS outperforms the other self-evolving fuzzy models here as well.

Rank based statistical analysis : To demonstrate that the improvement achieved with SPATFIS is statistically significant (and not random), a thorough rank based statistical analysis is performed and the numerical results are provided in appendix A.

Assessing the performance of the proposed SPATFIS method with the state-of-the-art neural fuzzy systems it is clear that SPATFIS attains better accuracy with less training time as shown in table 3.1 - 3.4. Looking at the tables, the following conclusions can be drawn,

- The inclusion of memory neurons in the neuro-fuzzy system helps SPATFIS to capture systems dynamics better and in turn it is able to attain better prediction accuracy (lower RMSE/ NDEI).
- Projection based learning (PBL) makes SPATFIS not only faster (hence lower training time) but it also enables the online capability of SPATFIS.
- Proposed self-evolving mechanism ensures that SPATFIS adopts the least number of fuzzy rules so that we get a parsimonious fuzzy rule base (smaller rule base).

An extension of SPATFIS is developed with an interval type-2 fuzzy system to handle uncertainty even better as briefly introduced in Appendix A.

3.5 Limitations of SPATFIS

In this chapter, we discussed how SPATFIS utilizes the memory type neurons to bring in recurrence and thus handles the intrinsic system dynamics of the time

TABLE 3.4: Performance of SPATFIS on stock price problem

Problem	Network Model	#Rules	Test RMSE	Time(s)
Stock	eTS [52]	09	0.070	1.4
	SimpleTS [53]	10	0.059	2.3
	SAFIS [58]	12	0.071	1.6
	McFIS [59]	12	0.036	1.5
	PANFIS [60]	15	0.049	2.0
	GENEFIS [61]	04	0.036	0.6
	SPATFIS	03	0.020	0.5

series. However, if the system dynamics is changing rapidly (i.e. a sharp drop in the stock price due to bad news etc.) then memory neurons fail to adapt to that quickly changing dynamics. As explained in section 3.1.1, memory neurons retain the effect of all past instances (refer to 3.2). Thereby the temporal output takes longer to get adjusted to a sudden shock and this phenomenon leads to poorer handling of system dynamics. Hence in such cases, the effect of infinite past in the memory neurons can become a critical issue. Therefore, it is very important to ensure that the temporal output is finite in nature and not depending on all pasts. The next chapter proposes a novel dynamic type neuron to address this problem.

Chapter 4

Neuro-Fuzzy Inference System with Dynamic Neuron (NFIS-DN)

A new Neuro-Fuzzy Inference System with Dynamic Neurons or NFIS-DN is presented in this chapter. The proposed dynamic system based neuron, referred to here as Dynamic Neuron (DN) is realized with a discrete-time nonlinear state-space model. The DN is designed in such a way, that its temporal output considers only the effect of finite past instances, enabling the system with finite memory. Thus it is able to remove the effect of infinite past associated with memory neurons in SPATFIS. The benefits of dynamic neurons (DNs) become more prominent in areas where there is a sudden change in the system dynamics such as non-linear dynamic system identification problems. NFIS-DN is also comprised of five layers like SPATFIS, and the Dynamic Neurons are employed only in the layers handling crisp non-fuzzy values. The antecedent and the consequent part of NFIS-DN are updated using a self-regulated backpropagation through time or BPTT learning algorithm to achieve finer tuning of the parameters.

[1](#)

---

¹The contents of this chapter is published in the following journal,
[97] Subhrajit Samanta et. al "A new neuro-fuzzy inference system with dynamic neurons (nfis-dn) for system identification and time series forecasting." Applied Soft Computing (2019)

4.1 NFIS-DN architecture

In this section, the network architecture of NFIS-DN is discussed starting with the design aspects of the proposed novel Dynamic Neurons (DN). Then the discussion will focus on layer wise operations of NFIS-DN and how the Dynamic Neurons are able to incorporate finite recurrence in the crisp layers of NFIS-DN.

4.1.1 Dynamic Neuron (DN)

Most artificial neural networks use a neuron model which is static in nature, i.e a nonlinear mapping between the static input and output. In this chapter, we propose a neuron model which is dynamic as it can non-linearly map input and output of time varying data. This neuron can be addressed as a neuron based on dynamic system and for convenience is hereafter referred to as a dynamic neuron or DN. These dynamic neurons are then incorporated in the different layers of the proposed neuro-fuzzy inference system to introduce local and global recurrence. The structure of a dynamic neuron (DN) is given in Fig. 4.1. Please note that the notation of Z^{-1} is used to denote one step delay and so on.

FIGURE 4.1: Realization of a Dynamic Neuron

The input to the dynamic neuron is $\mathbf{d}^I(k) \in R^p$, any p dimensional vector and the output vector is a 2 dimensional vector $\mathbf{d}^O(k) \in R^2$, where $\mathbf{d}^O(k) = [d_1^O(k), d_2^O(k)]^T$.

$d_1^O(k)$ provides the static mapping of inputs ($\mathbf{d}^I(k)$) whereas $d_2^O(k)$ represents the temporal output with the effect of finite time instances. The underlying dynamic system for the neuron is described by the following discrete-time nonlinear difference equations with its two dimensional states $\mathbf{s}(k) = [s_1(k), s_2(k)]^T \in R^2$ where $s_1(k), s_2(k)$ are the states representing the static nonlinear mapping and dynamic nonlinear mappings respectively, as given below.

$$s_1(k) = f(\mathbf{d}^I(k)) \quad (4.1)$$

$$s_2(k) = \alpha s_1(k-1) + (1-\alpha)s_2(k-1) \quad (4.2)$$

where $f(\cdot)$ can be any nonlinear function depending on the layer the dynamic neuron is in and α is the dynamic neuron learning parameter ($0 \leq \alpha \leq 1$). From equation 4.2 it is clear that the effect of infinite past is always present. Hence, a forgetting mechanism is employed to get the finite past. The output variables from the dynamic neuron are defined as below,

$$d_1^O(k) = s_1(k) \quad (4.3)$$

$$d_2^O(k) = s_2(k) - (1-\alpha)^N s_2(k-N) \quad (4.4)$$

where N is the number of finite past instances (past lags) which is required to be retained. Please note that the second term of the equation 4.4 helps in cutting off distant past values. Therefore it makes the dynamic neurons flexible and capable enough of handling quickly changing dynamics. Hence, the first output of DN (d_1^O) is a nonlinear static mapping from its inputs whereas the second output (d_2^O) works as a finite temporal mapping of the inputs.

Numerical example : In a scenario where the dynamic system requires to retain only past two instances (i.e. $N = 2$) the working of a dynamic neuron can be given by expanding equation 4.2,

$$s_2(k) = \alpha s_1(k-1) + \alpha(1-\alpha)s_1(k-2) + (1-\alpha)^2 s_2(k-2) \quad (4.5)$$

Plugging equation 4.5 into equation 4.4 we will get the final output as below,

$$d_2^O(k) = s_2(k) - (1-\alpha)^2 s_2(k-2) = \alpha s_1(k-1) + \alpha(1-\alpha)s_1(k-2) \quad (4.6)$$

This (equation 4.6) shows that the second (i.e. dynamic) output from the DN depends only on past two instances as intended.

During experiments, when N is unknown it is set using a constructive destructive method [98] i.e. the number of lags is first set to one, then number of lags is increased one at a time and corresponding cross-validation RMSE is recorded till the cross-validation RMSE does not go down rather goes up; that number of lags producing lowest cross-validation RMSE is set as the final number lags. It was observed from the conducted experiments that NFIS-DN produces the best result when N is in the range of $[2, 6]$. Similar strategy is utilized to set the number of fuzzy rules in this work as well (described in next section).

The proposed dynamic neuron addresses the biggest drawback of memory neurons (as introduced in SPATFIS) by addressing the issue of infinite memory. With memory neurons, the effect of infinite or all past was always present which hindered SPATFIS in adapting to quickly changing system dynamics. However, with proposed DN, NFIS-DN is able to discard the long term dependency on all past and hence it can handle quickly changing system dynamics more effectively. Unlike other static networks, NFIS-DN does not need the required past instances of $\mathbf{v}(k) = [y(k-1), \mathbf{u}(k)]$ to be fed as separate inputs, because the DNs are able to retain the necessary information of finite past. Here, $y(k)$, $\mathbf{u}(k)$ being the actual output and the exogenous input to the system.

Remark : It should also be noted that DN structure is different from an Infinite Impulse Response (IIR) filter [99] as DN uses a feed-forward connection to discard the effect of long past (refer to the bottom right area of the figure 4.1). Also, realizing an IIR with higher order lags is more complex than the proposed dynamic neuron which has a single order self-feedback (refer to the boxed left side of the figure 4.1). Besides, with an IIR filter, finite memory cannot be ensured as the implicit effect of all pasts will always be present.

4.1.2 Layer wise operations

The architecture of the proposed fixed structure neuro-fuzzy inference system with the dynamic neurons is shown in figure 4.2. It consists of a five-layer neuro-fuzzy network, which realizes a zeroth order TSK-type fuzzy inference system.

FIGURE 4.2: Network architecture of NFIS-DN

Here, the first layer is the input layer, the second layer is the fuzzification layer, the third layer the rule firing layer, the fourth layer is the normalization layer and the last layer is the output layer. The network while dealing with crisp data (i.e. input, normalization and output layer) employs the dynamic neurons (represented by solid squares in the figure 4.2). The input layer has $\hat{m}$ dynamic type neurons. The fuzzification layer and Gaussian rule layer has regular fuzzy neural nodes, whereas the normalization (or the defuzzification) layer has R dynamic neurons corresponding to the R number of rules followed by a single dynamic neuron in the final output layer. The number of fuzzy rules R is determined using a constructive-destructive procedure [98] in a similar manner as N was set. The center and the spread of the fuzzy rules are set randomly between $[0, 1]$. Description of each layer and their operations are presented next.

Input layer: The input layer has $\hat{m}$ input dynamic neurons which receive the m exogenous inputs $\mathbf{u}(k)$ and one lagged output $y(k-1)$, hence number of nodes is

$\hat{m} = m + 1$. We represent the augmented input vector as $\mathbf{v}(k) = [y(k-1), \mathbf{u}(k)]^T$. All of the neurons in this layer are dynamic neurons.

For the i^{th} dynamic neuron in the input layer (layer designated by 'I') the input ($\mathbf{d}^I$) is $\mathbf{v}_i(k)$ and the corresponding state variables (of the DN) are,

$$s_{1i}^I(k) = f_i^I(\mathbf{v}_i(k)) = \mathbf{v}_i(k), \quad i = 1, 2, \dots, \hat{m} \quad (4.7)$$

$$s_{2i}^I(k) = \alpha_i^I s_{1i}^I(k-1) + (1 - \alpha_i^I) s_{2i}^I(k-1) \quad (4.8)$$

where α_i^I is the dynamic neuron learning parameter, with $0 \leq \alpha_i^I \leq 1$. The outputs ($\mathbf{d}^O$) from the i^{th} dynamic neuron in the input layer are $x_{1i}(k)$ and $x_{2i}(k)$ and they are defined as below,

$$x_{1i}(k) = s_{1i}^I(k) \quad (4.9)$$

$$x_{2i}(k) = s_{2i}^I(k) - (1 - \alpha_i^I)^N s_{2i}^I(k-N) \quad (4.10)$$

here only N lags are considered. All of the states and the outputs definitions shown above are in accordance with subsection 2.1 (i.e. equations 4.1-4.4).

Fuzzification layer: Each of the crisp outputs from the input layer DNs, i.e. $x_{1i}(k)$ and $x_{2i}(k)$, are converted into fuzzy membership values using fuzzification process. Here, Gaussian membership functions are used to turn the crisp inputs into linguistic variables. The r^{th} (where $r = 1, 2, \dots, R$) membership value corresponding to r^{th} rule, for $x_{1i}(k)$ and $x_{2i}(k)$ are,

$$\phi_{1ri}(k) = \exp \left[-\frac{1}{2} \left[\frac{(x_{1i}(k) - \mu_{1ri})}{\sigma_r} \right]^2 \right] \quad (4.11)$$

$$\phi_{2ri}(k) = \exp \left[-\frac{1}{2} \left[\frac{(x_{2i}(k) - \mu_{2ri})}{\sigma_r} \right]^2 \right] \quad (4.12)$$

where μ_{1ri} and μ_{2ri} are the centers of the r^{th} membership functions for $x_{1i}(k)$ and $x_{2i}(k)$ respectively, and σ_r is the width of r^{th} membership function.

Rule firing layer: The R fuzzy neural nodes in this layer form the antecedents and do the rule aggregation of the fuzzy inference system using a t-norm (i.e. AND) operation which is realized by multiplying all the fuzzified input values ($\phi_{1ri}(k), \phi_{2ri}(k)$) where $r = 1, 2, \dots, R$ and $i = 1, 2, \dots, \hat{m}$ for each rules. So the

firing strength corresponding to the r^{th} rule is,

$$F_r(k) = \Pi_{i=1}^{\hat{P}} (\phi_{1ri}(k) \cdot \phi_{2ri}(k)) \quad (4.13)$$

Normalization layer: This is the consequent part of the architecture. The normalization layer (layer designated by 'R', also referred to as defuzzification layer) has R number of dynamic neurons corresponding to each rule. For r^{th} DN the input (d^I) is $[F_1(k), F_2(k), \dots, F_R(k)]^T$ and the corresponding state variables are,

$$s_{1r}^R(k) = f_r^R(F_1(k), \dots, F_R(k)) = \frac{F_r(k)}{\sum_{r=1}^R F_r(k)} \quad (4.14)$$

$$s_{2r}^R(k) = \alpha_r^R s_{1r}^R(k-1) + (1 - \alpha_r^R) s_{2r}^R(k-1) \quad (4.15)$$

where α_r^R is the dynamic neuron learning parameter ($0 \leq \alpha_r^R \leq 1$). Next, we compute the outputs ($\mathbf{d}^O$) for the DN (i.e. $\bar{F}_{1r}(k)$ and $\bar{F}_{2r}(k)$),

$$\bar{F}_{1r}(k) = s_{1r}^R(k) \quad (4.16)$$

$$\bar{F}_{2r}(k) = s_{2r}^R(k) - (1 - \alpha_r^R)^N s_{2r}^R(k-N) \quad (4.17)$$

The first output from each of the dynamic neurons in this layer represents the current normalized firing strength whereas second output is the finite temporal effect (N past instances) of the same. The states and outputs are defined as per subsection 4.1.1 (using equations 4.1 to 4.4)

Output layer: The output layer (layer designated by 'F') contains a single dynamic neuron. Its inputs (d^I) is $[\bar{F}_{11}(k), \dots, \bar{F}_{1R}(k), \bar{F}_{21}(k), \dots, \bar{F}_{2R}(k)]^T$ and the corresponding state variables are,

$$s_1^F(k) = f^F(\bar{F}_{11}(k), \dots, \bar{F}_{1R}(k), \bar{F}_{21}(k), \dots, \bar{F}_{2R}(k)) \quad (4.18)$$

$$= \sum_{r=1}^R w_{1r} \cdot \bar{F}_{1r}(k) + \sum_{r=1}^R w_{2r} \cdot \bar{F}_{2r}(k)$$

$$s_2^F(k) = \alpha^F s_1^F(k-1) + (1 - \alpha^F) s_2^F(k-1) \quad (4.19)$$

where α^F is the dynamic neuron learning parameter ($0 \leq \alpha^F \leq 1$). w_{1r}, w_{2r} are the weights connecting the first and second output of the r^{th} dynamic neuron (from

the rule layer) to the output DN, respectively. Next, we compute the outputs ($\mathbf{d}^O$) for the output dynamic neuron (i.e. $\hat{y}_1(k)$ and $\hat{y}_2(k)$),

$$\hat{y}_1(k) = s_1^F(k) \quad (4.20)$$

$$\hat{y}_2(k) = s_2^F(k) - (1 - \alpha^F)^N s_2^F(k - N) \quad (4.21)$$

Here also the state and output definitions are according to subsection 2.1. Finally, τ time step ahead prediction from NFIS-DN, i.e. $\hat{y}(k + \tau)$ is computed using the weighted summation of $\hat{y}_1(k)$ and $\hat{y}_2(k)$ as given below,

$$\begin{aligned} \hat{y}(k + \tau) &= \hat{y}_1(k) + \beta \cdot \hat{y}_2(k) \\ &= \sum_{r=1}^R w_{1r} \cdot \bar{F}_{1r}(k) + \sum_{r=1}^R w_{2r} \cdot \bar{F}_{2r}(k) + \beta \cdot \hat{y}_2(k) \end{aligned} \quad (4.22)$$

where β is the weight of the link connecting the second output from the dynamic neuron in the output layer to the final node. Here $0 \leq \beta \leq 1$ and weight of the link between the first output of the dynamic neuron and the final node is 1.

4.2 Learning in NFIS-DN

NFIS-DN adopts the backpropagation through time (BPTT) learning algorithm to learn from time series data, however it learns in a self-regulated manner.

4.2.1 Self-regulated learning

Recently, it has been shown that the self-regulated learning introduced in SRAN [100], IT2FIS [101] and McIT2FIS [96] can be helpful in selecting an appropriate learning strategy for a given sample. NFIS-DN introduces a new variant of self-regulated learning which captures the coarse knowledge first, followed by learning the finer knowledge. To express in a mathematical format, the self-regulated learning decides to learn from a sample (i.e. to update the parameters of the network for that sample) only when the prediction error $e(k) = \hat{y}(k) - y(k)$ follows the criteria below,

$$e(k) \geq T_{ini} (1 - q/Q) \quad (4.23)$$

where T_{ini} is the initial learning threshold, q is the current epoch with Q as the total number of epochs for the learning process. The learning threshold is empirically set between $[0.08, 0.12]$. Naturally, in the initial epochs (when q is small) the effective threshold is quite high hence the above criterion allows the network to learn from fewer samples (with higher errors). At the later stage however (when q is high), the effective threshold gets smaller and NFIS-DN learns from more and more samples to gain the finer knowledge.

As mentioned earlier, we employ back-propagation to train the network by unfolding it through time as introduced in [25]. The goal is to adjust the parameters of NFIS-DN such that it minimizes the squared prediction error loss E ,

$$E = \frac{1}{S} \sum_{k=1}^S e(k)^2 \quad (4.24)$$

where S is the total number of sampling instants (data points/ time instances), $\hat{y}(k)$ and $y(k)$ being the predicted and actual output respectively.

4.2.2 Updating the consequent parameters

The gradient descent method is used to perform backpropagation in order to update the consequent weights. In a gradient descent process, minimization of the loss function E (global quadratic error energy) is performed by going in the negative direction of the gradient of E in the parameter spaces.

Let's consider only the weight w_{1r} (weight of the link between r^{th} rule dynamic neuron's first output; where $r = 1, \dots, R$). To minimize E , the required change in this weight is proportional to the negative gradient of E with respect to w_{1r} ,

$$\Delta w_{1r} = -\eta \cdot \frac{dE}{dw_{1r}} \quad (4.25)$$

where η is the learning rate and is kept close to 0. Using chain rule of partial differentiation we can further simplify the right hand side as,

$$\frac{dE}{dw_{1r}} = \frac{dE}{d\hat{y}} \cdot \frac{d\hat{y}}{dw_{1r}} = -e(k) \cdot \bar{F}_{1r}(k) \quad (4.26)$$

Hence we can have the final update rule for the weight w_r^1 as below,

$$w_{1r}(k+1) = w_{1r}(k) + \eta \cdot e(k) \cdot \bar{F}_{1r}(k) \quad (4.27)$$

Depending on the parameter (which is being updated) backpropagation ensures that the traversal of error back through the architecture is following the correct path. While computing gradient for β (weight of the link connecting second output from the DN in the output layer and the final node) the error traversal will be along the second link of the output dynamic neuron, i.e.

$$\frac{dE}{d\beta} = \frac{dE}{d\hat{y}} \cdot \frac{d\hat{y}}{d\beta} = -e(k) \cdot \hat{y}_2(k) \quad (4.28)$$

Hence update rule for β is given by,

$$\beta(k+1) = \beta(k) + \eta \cdot e(k) \cdot \hat{y}_2(k) \quad (4.29)$$

The update rules for the other consequent weight w_{2r} is derived similarly as shown,

$$w_{2r}(k+1) = w_{2r}(k) + \eta \cdot e(k) \cdot \bar{F}_{2r}(k) \quad (4.30)$$

where $r = 1, 2, \dots, R$ represents the different fuzzy rules. On the other hand, the update rule for the learning parameter α^F in the output dynamic neuron is derived using a backpropagation through time approach. Only difference here with a regular back-propagation is that, the network unfolds in time so at any instance the required change in the particular parameter, will be dependent on past instances as well. The update rule is given by,

$$\alpha^F(k+1) = \alpha^F(k) + \eta \cdot e(k) \cdot \beta \cdot [\hat{y}_1(k-1) - \hat{y}_2(k-1)] \quad (4.31)$$

Similarly, the update rules for the learning parameters α_r^R in rule dynamic neurons (in defuzzification layer) is given as,

$$\alpha_r^R(k+1) = \alpha_r^R(k) + \eta \cdot e(k) \cdot w_{2r} \cdot [\bar{F}_{1r}(k-1) - \bar{F}_{2r}(k-1)] \quad (4.32)$$

4.2.3 Updating the antecedent parameters

The update rule for the centers of the r^{th} Gaussian membership functions (i.e. μ_{1ri} or μ_{2ri}) for $x_{1i}(k)$ and $x_{2i}(k)$ is,

$$\mu_{jri}(k+1) = \mu_{jri}(k) + \eta \cdot e(k) \cdot w_{1r} \cdot [\gamma(k) \cdot (x_{ji}(k) - \mu_{jri})] \quad (4.33)$$

where $j = 1, 2$ depending on the center which is being updated between μ_{1ri} or μ_{2ri} and $\gamma(k)$ is defined as $\frac{F_r(1-\bar{F}_{1r}(k))}{(\sum_{r=1}^R F_r(k))\sigma_{ri}^2}$. The width σ_r corresponding to each of the fuzzy rule is updated with,

$$\sigma_r(k+1) = \sigma_r(k) + \eta \cdot e(k) \cdot w_{1r} \cdot \left[\frac{\gamma(k)}{\sigma_{ri}} \cdot \sum_{i=1}^P [(x_{1i}(k) - \mu_{2ri})^2 + (x_{2i}(k) - \mu_{2ri})^2] \right] \quad (4.34)$$

where $r = 1, 2, \dots, R$. Finally the update rules for the learning parameters α_i^I in the input dynamic neurons is derived,

$$\begin{aligned} \alpha_i^I(k+1) = \alpha_i^I(k) + \eta \cdot e(k) \sum_{r=1}^R & \left[w_{1r} \cdot \gamma(k) (x_{2i}(k) - \mu_{2ri}) \right. \\ & \left. \times [x_{2i}(k-1) - x_{1i}(k-1)] \right] \end{aligned} \quad (4.35)$$

A pseudo-code for the proposed self-regulated gradient descent based backpropagation through time learning method for NFIS-DN is presented with a flow chart in the appendix B

4.3 Performance evaluation of NFIS-DN

In this section, the performance evaluation of the proposed NFIS-DN is performed under three categories. In the first category, the performance evaluation is carried out on nonlinear system identification problems ([58],[102], [79]) primarily to demonstrate the advantages of the dynamic neurons in capturing rapidly changing system dynamics. Under the second category, the performance of NFIS-DN is presented using standard benchmark problems such as Mackey Glass [94] and Box Jenkins. Finally, two real world applications of the proposed NFIS-DN, one

on wind speed forecasting [103],[104] and the other on financial stock price prediction [105, 106] are presented. For performance comparison, root mean square error or RMSE (normalized) or non-dimensional error index (NDEI) are used as performance metrics according to the standard practice. The number of fuzzy rules and required lags are set using constructive-destructive method, as mentioned in the previous section. More experiment results are provided in the appendix B.

4.3.1 System identification problems

Performance of NFIS-DN are evaluated with two nonlinear dynamical system identification problems (popular in the control research) along with further analysis.

4.3.1.1 Nonlinear System Identification Problem I (SID 1)

The first system identification problem is taken from [79, 102]. This discrete time nonlinear dynamical dataset is generated with a geometric non-linearity such that the future output $y(k+1)$ depends on three past outputs and two past inputs as given below,

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)(y(k-2)-1) + u(k)}{1 + y^2(k-2) + y^2(k-2)} \quad (4.36)$$

where $u(k)$ is the exogenous input to the system. As described in [79] the data generated from 900 time steps are used for training the network. Among the 900 time steps, the first 350 steps use a *iid* sequence uniform over $[-2, 2]$ as input and the remaining time steps use $\sin\left(\frac{\pi k}{45}\right)$ as an input to the system. If one uses the input-output realization [102], [59], [107] the number of inputs to the network are five, whereas in NFIS-DN only two inputs are fed, i.e. $u(k)$ and $y(k-1)$ while output is $\hat{y}(k)$ as the dynamic neurons take care of the required past instances. The network is trained in batch mode for 100 epochs. The number of fuzzy rules (R) in NFIS-DN was set at 8 and N was set to 2 using constructive-destructive method. The learning rate is kept at 0.0001 and the initial learning threshold at 0.1 whereas the dynamic neuron parameters, center and spreads of fuzzy rules are initialized randomly. The other experiments also employ the same initialization of the parameters except for R and N .

After the training, the network is tested with the following input signal,

$$u(k) = \begin{cases} \sin(\frac{\pi k}{25}) & \text{where } 0 \leq k \leq 250 \\ + 1.0 & \text{where } 251 \leq k \leq 500 \\ - 1.0 & \text{where } 501 \leq k \leq 750 \\ 0.3 \sin(\frac{\pi k}{25}) + 0.01 \sin(\frac{\pi k}{32}) + 0.6 \sin(\frac{\pi k}{10}) & \text{where } 751 \leq k \leq 1000 \end{cases} \quad (4.37)$$

(A) NFIS with dynamic neuron (NFIS-DN)

(B) With regular memory neuron (like in SPATFIS)

FIGURE 4.3: Study on effect of dynamic neuron in NFIS

Figure 4.3a shows the actual output of the discrete time dynamical system vs NFIS-DN's predicted output. From the figure, we can easily observe that the NFIS-DN follows the actual output very closely. The RMSE for training and testing are 0.015 and 0.020 respectively. The Dynamic Neurons in the input, normalization and output layer helps NFIS-DN to approximate the dynamics closely.

In order to study the effect of dynamic neurons, we have conducted a study with the identical structure as NFIS-DN but instead of DN we use memory neurons (MN) which retains memory of all past instances as described in SPATFIS. The predicted

output from SPATFIS and the actual output is shown in Figure 4.3b. From the figure 4.3b, we can observe that a memory type neuron with all past memory leads to overshoot in the predicted output, whereas the proposed NFIS-DN with finite memory DN tracks the quickly changing dynamics precisely as shown in Fig. 4.3a. The same reflects in the performance measure of SPATFIS (with same number of rules and self-regulated BPTT learning) where RMSE is 0.035, which is at least 40% more than that of NFIS-DN.

Next, the performance of the NFIS-DN is compared with 4 popular neuro-fuzzy systems and SPATFIS. The number of rules, CPU time(s) and testing RMSE are reported in table 4.1. From the table, we can observe that NFIS-DN achieves better performance than the other state-of-the-art neuro-fuzzy networks both in terms of accuracy and run-time. For further statistical tests, ranks are assigned in this chapter with regards to the accuracy (more details can be found in B). Please note that NFIS-DN is slower than SPATFIS as it learns in a traditional offline manner (i.e. self regulated BPTT) whereas SPATFIS learns in an online manner.

TABLE 4.1: Performance of NFIS-DN on nonlinear SID I problem

Problem	Network Model	#Rules	Time(s)	Testing RMSE
SID 1	eTS [52]	49	30	0.0212
	SimpleTS [53]	22	25	0.0297
	SAFIS [58]	17	45	0.0221
	McFIS [59]	10	22	0.003
	SPATFIS [88]	10	5	0.035
	NFIS-DN	8	12	0.020

4.3.1.2 Nonlinear System Identification Problem II (SID 2)

The second problem used here is a single-input-single-output discrete time nonlinear dynamics system and it is adopted from [58, 102]. The input $u(k) = \sin\left(\frac{2\pi k}{100}\right)$ is used to generate the one time step ahead output $y(k+1)$. The first 50,000 samples are used for training and the remaining 200 samples are used for testing the network (as described in [58]). Number of fuzzy rules is set at 10 and N is kept at 2. Performance of the system identification 2 problem is also compared

with the well-known neuro-fuzzy systems. Table 4.2 shows the number of rules, CPU time(s) and testing RMSE clearly indicating NFIS-DN's better performance in comparison with other state-of-the-art neuro-fuzzy methods.

TABLE 4.2: Performance of NFIS-DN on nonlinear SID II problem

Problem	Network Model	#Rules	Time(s)	Testing RMSE
SID 2	eTS [52]	49	09	0.021
	SimpleTS [53]	22	14	0.030
	SAFIS [58]	17	11	0.022
	McFIS [59]	10	17	0.003
	SPATFIS [88]	10	3	0.025
	NFIS-DN	10	07	0.002

4.3.2 Standard benchmark problems

In this section, performance comparison is executed on two benchmark time series forecasting problems, viz., the Mackey-Glass time series prediction [94] and the Box-Jenkins gas furnace problem [1].

4.3.2.1 Mackey- Glass problem

The MG problem is already described in the chapter 3 (refer to section 3.4.1.1). Using the constructive-destructive method N was set to 3. The performance of the prediction is measured using the non-dimensional error index (NDEI is the ratio of the Test RMSE and the standard deviation of the test data; in this case it is 0.51). Table 4.4 provides the rule numbers, CPU time(s) and test-NDEI of NFIS-DN. From this table, it can be observed that NFIS-DN outperforms the other state-of-the-art approaches in terms of both accuracy and run-time, while using a similar number of rules (however slower than SPATFIS).

TABLE 4.3: Performance of NFIS-DN on MG problem

Problem	Network Model	#Rules	Time(s)	Test NDEI
Mackey-Glass	eTS [52]	09	3.1	0.260
	SAFIS [58]	08	5.5	0.360
	SimpleTS [53]	11	4.4	0.320
	McFIS [59]	10	8.2	0.250
	SPATFIS [88]	06	0.5	0.011
	NFIS-DN	08	2.6	0.020

4.3.2.2 Box-Jenkins problem

Detailed descriptions of the BJ data-set is already provided in the chapter 3 (refer to section 3.4.1.2). The performance of NFIS-DN (with N set to 3) is compared with state-of-the-art neuro-fuzzy methods available in the literature and given in table 4.4. From the table, it can be observed that the test RMSE achieved with NFIS-DN was better than the other methods. Hence, it is apparent that NFIS-DN is able to predict better in comparison with other approaches used in the literature.

TABLE 4.4: Performance of NFIS-DN on BJ problem

Problem	Network Model	#Rules	Time(s)	Test RMSE
Box-Jenkins	eTS [52]	08	0.7	0.049
	SAFIS [58]	09	1.5	0.071
	SimpleTS [53]	07	0.8	0.055
	McFIS [59]	12	0.4	0.045
	SPATFIS [88]	04	0.3	0.046
	NFIS-DN	10	1.8	0.033

4.3.3 Real-world Problems

Next, the performance of NFIS-DN is presented on two standard real-world problems, viz. , that of wind speed prediction and stock price prediction problems.

4.3.3.1 Wind speed forecasting

The wind speed prediction problem is already explained in the chapter 3 (refer to section 3.4.2.1). Number of past lags to consider for DN i.e. N , is set at 2 using the constructive-destructive method. Figure 4.4 shows the actual speed vs forecast speed on testing dataset. It is apparent from figure 4.4, that NFIS-DN is able to generalize and predict well in most cases, except for the areas highlighted by circles which are possibly because of high noise. In table 4.5, the performance of NFIS-DN is compared with the aforementioned neuro-fuzzy systems and it can be seen that NFIS-DN attains better accuracy with smaller/ similar number of rules.

FIGURE 4.4: Wind speed prediction using NFIS-DN

TABLE 4.5: Performance of NFIS-DN on wind speed problem

Network Model	#Rules	CPU time(s)	Test RMSE
eTS [52]	9	0.5	0.380
SAFIS [58]	16	0.7	0.376
SimpleTS [53]	11	0.4	0.394
McFIS [59]	10	0.9	0.230
SPATFIS [88]	06	0.2	0.140
NFIS-DN	10	2.1	0.135

4.3.3.2 Stock price forecasting

The stock price prediction problem has already been described in the last chapter 3 (refer to section 3.4.2.2). The value of N is set to 2 here. Figure 4.5 shows the actual price of the stock and the one day-ahead forecast by NFIS-DN. From the figure, it can be observed that NFIS-DN is able to follow the change in stock price very well. Even when there was high stock volatility and drastically large changes (time steps 200-500, which was during the financial crisis of 2008) which clearly indicates an abrupt change in system dynamics, NFIS-DN was able to track satisfactorily, and provide a great forecast. Having compared the prediction MSE result with other architectures, it was apparent that NFIS-DN attains better accuracy than the state-of-the-arts methods.

FIGURE 4.5: Stock price prediction using NFIS-DN

A rank based non-parametric test is also performed to further demonstrate the statistically significant performance of NFIS-DN (in terms of prediction accuracy) as presented in more details in the appendix B . It is demonstrated that NFIS-DN out-performs the other state-of-the-art neuro-fuzzy methods with regards to prediction accuracy, however it is slower than SPATFIS.

TABLE 4.6: Performance of NFIS-DN on stock price problem

Network Model	#Rules	CPU Time(s)	Test RMSE
eTS [52]	09	1.4	0.070
SAFIS [58]	12	1.6	0.071
SimpleTS [53]	10	2.3	0.059
McFIS [59]	12	1.5	0.036
SPATFIS [88]	03	0.5	0.020
NFIS-DN	10	1.7	0.019

4.4 Limitations of NFIS-DN

Having discussed the advantages of NFIS-DN it is time to explain one of its major drawbacks. As mentioned in section 4.1.1 the value of N in equation 4.4 (How many past instances to be considered) is set in an empirical manner. Thus all samples need to be present to perform a constructive-destructive test for choosing N , thereby making this process offline/ batch in nature. Hence in case of an online learning scenario or a fast data stream type situation, this method of choosing N will not hold. Therefore, we require a better and faster algorithm that is capable of estimating the temporal order (i.e. determining N) in an online manner. Once the estimation of N is automated and online, its application across recurrent neuro-fuzzy and recurrent neural networks will be substantial. The next chapter proposes a Bayesian probabilistic approach to address the problem of estimating N (i.e Temporal Dependency Estimation) explicitly.

Chapter 5

Learning Elastic Memory Online (LEMON)

From previous chapters, it is well established that any kind of time series algorithm requires past information to model the inherent temporal relationship present in the data. Traditionally, this temporal dependency or temporality (i.e. number of past samples required for a good prediction) is determined in an empirical manner (as demonstrated with NFIS-DN in Chapter 4), thus they are generally offline in nature (i.e. estimation of N in Dynamic Neurons), making them impractical to be adopted in the online or streaming context. Hence, a novel method of online temporality analysis is proposed in this chapter. Temporality change or drift can be a common occurrence in online time series and data streams. Hence a drift detection mechanism is also proposed. Once such drift is detected, a drift handling mechanism is triggered. The entire mechanism is termed as Learning Elastic Memory Online or LEMON as it estimates the past dependency and adjusts its memory accordingly (hence elastic). We also develop a neuro-fuzzy framework (with two novel variants) as an extension to LEMON and refer to it as Bayesian Neuro-Fuzzy Inference System. BaNFIS is designed to perform both temporal dependency estimation and uncertainty handling in an online manner.

1

---

¹The contents of this chapter are published in the following journals,
[108] Subhrajit Samanta et. al "Learning elastic memory online for fast time series forecasting." Neurocomputing (2019).
[109] Subhrajit Samanta et. al " Bayesian Neuro-Fuzzy Inference System (BaNFIS) for Temporal Dependency Estimation " IEEE Transaction on Fuzzy Systems (2020)

Definition of temporality : Any time series prediction problem is generally cast as a nonlinear system dynamics problem [110] where τ steps ahead actual output $y(k + \tau) \in \mathbb{R}^1$ is estimated with augmented input vector $\mathbf{v}(k) = [\mathbf{u}(k), y(k - 1)]^T \in \mathbb{R}^{P+1}$ such that predicted output,

$$\hat{y}(k + \tau) = h(\mathbf{v}(k), \mathbf{v}(k - 1), \dots, \mathbf{v}(k - (n - 1))) \quad (5.1)$$

where $h(\cdot)$ can be any arbitrary nonlinear function and $\mathbf{u}(k) \in \mathbb{R}^P$ is the exogenous input (factor) vector. From the formulation above we have n as the total number of samples to be retained (i.e. $n - 1$ past instances and 1 current instance) to model the data. This n is regarded as the temporality measure also referred to as temporal order or temporal dependency. To the best of our knowledge, there is no work in the literature to estimate this temporality in an online manner. Change in the temporal behaviour (dynamics) is a common occurrence for online time series data. Such changes in the temporality of the environment are regarded as temporal drift. When a drift occurs i.e. the temporality changes (i.e. from n_1 to n_2), it is crucial for the model to detect the drift and re-adjust its behavior so that the algorithm can continue to perform its prediction task efficiently.

5.1 Temporality Estimation (TE)

The proposed temporality estimation process is designed such way that it is agnostic of the underlying network (i.e. it can be built on any neural network platform). With the proposed method, the underlying network will be able to adapt its memory requirement hence it is also termed as Adaptive Temporal Neural Network. The learning will take place in a sequential chunk wise (i.e. mini-batch with no repetition) manner. In the very beginning, ATNN is trained in a feed-forward manner without any recurrence to impart a baseline knowledge; this can be performed with first few chunks using the online weight update method. After acquiring preliminary knowledge (i.e. warming up) the TE technique is triggered. The first step is to populate an initial window with past prediction errors. Then we utilize those errors to build the Bayesian framework. The posterior probabilities (PP) of past instances represent the importance of past lags for the prediction task. This is the target quantity which will lead to temporality estimation. However, for derivation of the posteriors, the likelihood and the priors need to be defined first. A

schematic diagram of the Temporality (i.e. Past Dependency) Estimation process is presented in figure 5.1 for easier comprehension. Following section provides a detailed step by step description of the TE method.

FIGURE 5.1: Overview of Temporality Estimation (TE) Process

5.1.1 Memory window initialization

ATNN starts with a memory window (W) initially of size W_{in} which can be regarded as the maximum number of past instances to be examined. W will be updated according to the estimated temporality (discussed in following section) hence it has to be initialized only once. With each data points, ATNN predicts the output and the past predictions are preserved in the memory window in a first-in-first-out setting. At k^{th} time instance the content of memory window is represented with vector $y^M(k) \in \mathbb{R}^W$ as,

$$y^M(k) = [\hat{y}(k), \hat{y}(k-1), \dots, \hat{y}(k-j) \dots \hat{y}(k-(W-1))]^T \quad (5.2)$$

for $j = 0, 1, 2, \dots, (W-1)$. $\hat{y}(k-j)$ represents the predicted output for $(k-j)^{th}$ datum. An error vector $e^M(k) \in \mathbb{R}^W$ is constructed based on the prediction vector

$y^M(k)$ such that,

$$e^M(k) = [e_0(k), e_1(k) \cdots e_{W-1}(k)]^T \quad (5.3)$$

where $e_j(k) = [y(k) - \hat{y}(k - j)]$ can be regarded as the prediction error if $(k - j)^{th}$ sample alone is employed to predict the actual k^{th} output i.e. $y(k)$.

5.1.2 Probability derivations

In a Bayesian probabilistic framework, to compute the posterior probabilities (of past instances which basically represents the importance of past lags) we first need to define the likelihood and the priors as shown below. After that Bayes' formula can be employed to compute the target posteriors.

5.1.2.1 Likelihood computation

It is observed that in most time series and dynamic system handling problems the prediction errors follow a normal distribution. For the sake of simplicity a zero mean (as the ideal prediction error should be zero) normal distribution is assumed here to define the likelihood (probability) of accurate prediction. Hence utilizing only the $(k - j)^{th}$ sample $\mathbf{v}(k - j)$, probability of correctly predicting output $y(k)$ i.e. $P(y(k)|\mathbf{v}(k - j))$ can be defined as,

$$\begin{aligned} P(y(k)|\mathbf{v}(k - j)) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y(k) - \hat{y}(k - j))^2}{2\sigma^2} \right] \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{e_j^2(k)}{2\sigma^2} \right] \end{aligned} \quad (5.4)$$

The likelihoods are calculated for all the entities the memory window i.e. $j = 0, 1, 2 \cdots (W - 1)$. σ is set from the variance in the error matrix $e^M(k)$ but if the resultant probabilities are very similar it is set empirically between 0.05 to 0.5. The definition of the likelihood in equation 5.4 indicates that a low error $e_j(k)$ will correspond to high prediction probability and vice versa; hence the likelihood definition is asserted proper.

5.1.2.2 Recursive prior probability

Prior probability assignment has been a heavily debated topic in the field of Bayesian statistics for a long time. Although a reasonable choice of informed prior is accepted in the community, no such assumptions are made in this work to assign the priors for the W past instances (retained in the memory window). Rather a flat or uniform prior is used for them in the very beginning of every chunks i.e. $P(\mathbf{v}(k-j)) = 1/W$ to compute the posteriors $P(\mathbf{v}(k-j)|y(k))$. Then the prior is updated with the resultant posteriors (for each sample) with a recursive weighted sum method as shown below,

$$P(\mathbf{v}(k+1-j)) = \alpha P(\mathbf{v}(k-j)) + (1-\alpha)P(\mathbf{v}(k-j)|y(k)) \quad (5.5)$$

for $j = 0, 1 \dots (W-1)$. The regulating factor α is set between $[0, 1]$ according to the problem. If the prior distribution is known then the same should be utilized.

5.1.2.3 Posterior probability

After calculating likelihoods and assigning priors next the posterior probabilities (PP) are computed. For each of the samples retained in the memory window (i.e. $\mathbf{v}(k-j)$) the posterior $P(\mathbf{v}(k-j)|y(k))$ represents the significance of $\mathbf{v}(k-j)$ in the prediction of $y(k)$ and the same is derived using Bayes,

$$\begin{aligned} P(\mathbf{v}(k-j)|y(k)) &= \frac{P(y(k)|\mathbf{v}(k-j)).P(\mathbf{v}(k-j))}{P(y(k))} \\ &= \frac{P(y(k)|\mathbf{v}(k-j)).P(\mathbf{v}(k-j))}{\sum_{j=0}^{W-1} [P(y(k)|\mathbf{v}(k-j)).P(\mathbf{v}(k-j))]} \end{aligned} \quad (5.6)$$

for $j = 0, 1, 2 \dots (W-1)$. This posterior probability distribution is utilized next to measure the temporality i.e. if PP is considerably high till $(k-j)^{th}$ lag, then $j+1$ (j past and one current samples) is considered as the temporality measure. The last step of the temporality estimation process is discussed next.

5.1.2.4 Measuring temporality

The temporality measure is computed from the posterior probability (PP) matrix (equation 5.6) using a *Pareto based method*. First the posteriors (PP) for past lags (upto $W - 1$) are averaged over all the samples in the chunk to form an aggregate posterior matrix (PP_{avg}). Then a Pareto inspired approach is employed to measure the final temporality.

Pareto or the 80-20 rule says that, 80% effect stems from 20% of the factors. Here we adopt a similar approach to estimate the temporality from the average posteriors. Starting with the recent most posterior ($j = 0$), we accumulate the posterior probabilities one by one till the total is 0.8 (i.e. Pareto threshold, $T = 80\%$ of the total probability; as the sum of these are 1). Total number of instances leading to this sum, is the final measurement of temporality (i.e. n).

$$\mathbf{IF} \sum_{j=0}^q PP_{avg}(j) \leq 0.8 \mathbf{THEN} n = q + 1$$

Hence, the temporal measure is $n = q + 1$ corresponding to current and q past instances i.e. $\mathbf{v}(k), \mathbf{v}(k - 1) \cdots \mathbf{v}(k - q)$.

5.2 Learning elastic memory

The autonomous LEMON method incorporates the TE technique along with a drift handling mechanism, all in an online, chunk by chunk manner. If all of the training data is available entirely then the data can be divided into chunks to mimic the sequences. None of the chunks are revisited after they are fed to the network once, which makes the learning process truly online and to do so the network weights are updated using an one shot projection based learning method [87] or PBL (chosen in this article for its ease of implementation). The PBL method is already described in detail in Chapter 3. LEMON adopts two strategies during the training process. It either learns from the retained samples (according to measured temporality) or in case of a temporality drift in the environment, it evolves the initial memory window and re-estimates the temporality to handle the drift.

5.2.1 Temporal learning

Once the temporality is measured to be n (from TE) then the memory of ATNN is updated accordingly i.e. $W = n$. The name elastic memory is used because the memory window is capable of adjusting itself with the temporality measurement. LEMON adopts the following strategy to learn from each of the samples. For k^{th} instance all the n retained samples in the memory window (i.e. $\mathbf{v}(k), \mathbf{v}(k-1) \cdots \mathbf{v}(k-(n-1))$) are utilized to predict the output $\hat{y}(k)$ where $y(k)$ is the actual output. Hence, ATNN trains from the samples by treating each of the past instances as separate features (i.e. a total of n features for this example) with PBL weight update.

Regular RNNs with backpropagation through time learning often suffer from the problems of exploding or vanishing gradients but the proposed temporal learning treats past information as separate features (like a feed-forward network) hence it is immune to the said problems. This strategy is maintained till there's a clear sign of drift in the temporal behavior of the data. The procedure to handle such drifts are discussed next.

5.2.2 Handling temporal drift

During the aforementioned temporal learning the joint probability of correct prediction is measured for each of the samples. Here also a zero mean normal distribution is adopted to define the conditional joint probability as shown below,

$$\begin{aligned} JP(k) &= P(y(k) | \mathbf{v}(k), \mathbf{v}(k-1) \cdots \mathbf{v}(k-(n-1))) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(y(k) - \hat{y}(k))^2}{2\sigma^2} \right] \end{aligned} \quad (5.7)$$

where $y(k)$ and $\hat{y}(k)$ are the actual and predicted outputs for k^{th} data point using all the current and retained inputs in memory, i.e. $\mathbf{v}(k), \mathbf{v}(k-1) \cdots, \mathbf{v}(k-(n-1))$. The joint probability is then averaged over all samples in the chunk.

5.2.2.1 Drift indicator and drift detection

The average joint probability JP_{avg} acts as the drift indicator $D(C)$ where C is the index of the chunk in hand,

$$D(C) = JP_{avg}(C) = \frac{\sum_{k=1}^{ChunkSize} JP(k)}{ChunkSize} \quad (5.8)$$

$ChunkSize$ is the number of samples in the C^{th} chunk.

Temporal drift originates from the change in temporality in the environment. So when there is no temporal drift, ATNN learns from more and more samples, improves its prediction ability, prediction error decreases and in turn joint probability increases. Hence the value of drift indicator increases i.e. $D(C) \geq D(C-1)$ in case of no drifts. However if there is a drift in temporality then employment of the same outdated measure will lead to poorer prediction, higher error and a decrease in JP_{avg} . Hence LEMON concludes the presence of drift in temporality from the $(C-1)^{th}$ chunk to the C^{th} chunk when,

$$D(C) < D(C-1) \quad (5.9)$$

Hence a decrease in drift indicator during temporal learning signifies a drift.

5.2.2.2 Memory window evolution

From the last section, it can be recalled that memory window was initialized with W_{in} at the beginning where W_{in} was the number of maximum lags to be examined (during temporality estimation). Once the drift is detected, it is important to re-estimate the temporality and to do so the window size W has to evolve. The process of window evolution is made autonomous with the size depending on the rate of drift between chunks.

If the drift rate is high then the window size should be smaller so that the network is able to quickly learn from near past and does not get hindered by the redundant knowledge of long pasts. On the other hand when the drift rate is relatively low then it can afford to retain longer pasts to learn more intricate details of the intrinsic temporal behavior from the data. The drift rate between the $(C-1)^{th}$ and the C^{th}

chunk is defined,

$$DR(C) = \frac{D(C) - D(C - 1)}{D(C - 1)} \times 100 \quad (5.10)$$

The memory window is initialized every time drift is detected in temporality, using the drift rate with an approach inspired from [111] as shown below,

$$W = \min(ChunkSize(e^{-DR} + \beta), W_{in}) \quad (5.11)$$

where $ChunkSize \times \beta$ results in user defined minimum window size. From equation 5.11, it is clear when the rate (DR) is very high the minimum window size is considered and when the rate is too low no more than W_{in} samples are taken into account.

5.2.2.3 Temporality re-estimation

Once the drift is detected and the memory window W is re-initialized then the temporality can be estimated again to handle the drift. As shown in equation 5.6, the posteriors (PP) for past lags (upto $W - 1$) are computed and then averaged over all the samples in the chunk to form an aggregate posterior matrix (PP_{avg}) as shown here,

$$PP_{avg}(j) = \frac{\sum_{k=1}^{ChunkSize} P(\mathbf{v}(k - j)|y(k))}{ChunkSize} \quad (5.12)$$

for $j = 0, 1, 2 \dots (W - 1)$. The Pareto based temporality estimation technique is applied next to determine the temporality measure. After re-estimating the new temporality, LEMON resumes with its temporal learning strategy.

Remark 1 : Irrespective of the learning strategy (temporal or drift handling) adopted in LEMON, the network weights are updated in one shot after each chunk using the projection based learning (PBL)[87] to make LEMON online in nature. Same can be done with stochastic gradient descent (SGD) [86] as well. The PBL method is utilized here for its simplicity and ease of implementation. PBL is already explained in details in Chapter 3 (as SPATFIS also utilizes the same method to update its parameter). The over-view of the entire LEMON framework is presented in figure 5.1 for summarizing.

FIGURE 5.2: Overview of LEMON

To recap, LEMON's prominent features are,

- Ability to estimate temporality and utilize it with temporal learning
- Ability to detect the presence of a temporal drift
- Ability to evolve memory window size and re-estimate temporality when drift occurs. Therefore it is able to handle the temporal drift.
- Ability to update weights in a fast and online manner with PBL

An algorithmic flow-chart is also provided as additional material in the Appendix C to summarize the entire LEMON mechanism.

5.3 Proof of concept with synthtetic data

The claims of temporality estimation and temporal drift detection techniques as proposed in this chapter are difficult to substantiate with real world data. Hence, we take aid of a synthetic data with varying (but known) temporality. The synthetic data is generated such that its output depends on certain past samples i.e. the temporality is known (instead of using a real world data whose temporal order is unknown) and we employ the data to show that LEMON is able to identify the order correctly along with the drifts in temporality. Thus the theoretical claims made in this article are corroborated. As mentioned earlier LEMON is independent of the underlying neural-network. In this study a basic 3 layer radial basis function (RBF) network [15] is chosen for its well known suitability in time series problems, for its robustness against noise and ease of implementation.

5.3.1 Nonlinear System Identification I

A popular nonlinear dynamical system problem commonly found in control literature [102, 112] is used to generate the SID data. SID contains 1800 data-points with first 400 having exogenous input $u(k)$ randomly distributed between $[-2, 2]$ while for rest of them, $u(k) = 1.05\sin(k/45)$ is used as input. During training and testing $\mathbf{v}(k) = [u(k), y(k-1)]^T$ is used as the overall input to the system for each sample whereas $y(k)$ is the target output.

The output is generated using 3 previous instances of the output, current and one previous instance of exogenous input using a nonlinear function as given by, $y(k) = f(y(k-1), y(k-2), y(k-3), u(k), u(k-1))$ where the function f is,

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2} \quad (5.13)$$

From the definition of $y(k)$ it is clear that to model SID, three past instances of $\mathbf{v}(k) = [u(k), y(k-1)]^T$ need to be retained hence as per the claims made in this chapter, LEMON should predict the temporal order of SID to be 3.

Experimental setup: The data is chronologically divided into chunks in order to simulate the incoming streams of data. A basic RBF network with 30 hidden nodes is used as the underlying network (i.e. ATNN). An initial window size of

10 (i.e. $W_{in} = 10$) and a chunk size of 90 (i.e. 20 chunks each with 5% of total samples) are used. In the beginning, the first two chunks are employed to train the RBF network (same as in SID experiment) in a feed-forward manner without any recurrence (using PBL) to provide it with a baseline knowledge so that it can start predicting.

Discussion: In the third chunk the TD takes place for the first time. As explained before, in this phase of LEMON, it estimates the temporal order of the model. The prior probabilities for different lags of inputs (i.e. $\mathbf{v}(k), \mathbf{v}(k-1) \dots$) are set with the recursive formula (as in equation 5.5) and the aggregated posterior probability PP_{avg} is found out from equation 5.12. The Pareto method is employed which indicates first 3 posteriors to be most significant (for $\mathbf{v}(k), \mathbf{v}(k-1), \mathbf{v}(k-2)$). So the estimated temporal order is 3. Hence LEMON is able to estimate SID's order correctly. Once the order is determined, it resorts to temporal learning with a memory window size of 3 (adapted to 3 from initial 10) from the next chunks while recording the average joint probability values (JP_{avg}) for identification of possible drift. From the generation of SID it is known that there is no change in temporal behavior midway and the same is reflected with LEMON as the JP_{avg} values can be seen to be monotonically increasing in Figure 5.2 ensuring no triggering of Swift Learning for the rest of the chunks.

FIGURE 5.3: Temporal dependency estimation for SID

5.3.2 Nonlinear System Identification II

This data is also generated from the same nonlinear dynamic function as the previous one however the problem is made more complex by introducing multiple drifts in temporality. ATNN is supposed to measure the temporality, detect drift and re-estimate the order successfully.

For the first 500 samples the output depends on 4 past instances of the output with a function of the form below, $y(k) = f_1(y(k-1), y(k-2), y(k-3), y(k-4), u(k), u(k-1))$ where the function f_1 has the following form,

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = \frac{x_1 x_2 x_3 x_4 x_6 (x_4 - 1) + x_5}{1 + x_2^2 + x_3^2 + x_4^2} \quad (5.14)$$

From the definition of $y(k)$ it is clear that 4 past instances of $\mathbf{v}(k) = [u(k), y(k-1)]^T$ are required to be retained. Hence, LEMON should estimate the temporality to be 4 for this part. For the next 700 samples, $y(k)$ depends on previous 2 instances of output as shown here, $y(k) = f_2(y(k-1), y(k-2), u(k), u(k-1))$ where the function f_2 is,

$$f_2(x_1, x_2, x_3, x_4) = \frac{x_1 x_2 x_4 (x_2 - 1) + x_3}{1 + x_2^2} \quad (5.15)$$

hence LEMON should be able to detect the drift and re-estimate the temporality to be 2 here. In the last part, 3 past instances are used to generate the output as given below, $y(k) = f_3(y(k-1), y(k-2), y(k-3), u(k), u(k-1))$ with the function f_3 as,

$$f_3(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2} \quad (5.16)$$

LEMON has to identify the drift again and re-estimate the measure to be 3.

Experimental setup: The data is divided again into 20 chunks (each with 5% of total samples) to simulate a sequential data stream. A regular 3 layer RBF with 30 hidden nodes is used as the underlying framework. First 2 chunks are employed to impart a baseline knowledge to the model with PBL and then $W_{in} = 10$ is initiated only once, so in no point of time more than 10 past lags will be examined. The prior regulating parameter is set to $\alpha = 0.9$. Please note that the same initialization is carried out throughout all the experiments.

Discussion: From the third chunk onward LEMON is triggered. Starting from an uniform prior of 1/10, it computes the posteriors, updates the priors and so on.

After visiting all the samples in the chunk the average posterior probability is found and the Pareto method is employed to find the temporality. After the temporality is measured to be 4 (shown in figure 5.4) LEMON resumes with temporal learning where all 4 retained samples (1 current and 3 past) are utilized for training. During temporal learning the drift indicator actively looks for a drift in the temporality. Whenever the drift indicator value drops the the temporality re-estimation process kicks in starting with the memory window evolution. The formation and evolution of memory window (on the basis of drift rate) can also be observed in figure 5.4. Once the re-estimation task is complete LEMON resumes with temporal learning. The drift indicator and temporality measures are also plotted in figure 5.4. From the figure it can be concluded that LEMON is able to measure temporality correctly (i.e. 4,2 and 3), detect and handle the drift.

FIGURE 5.4: Temporal dependency Estimation for SID2

Remark 3: The success of the proposed method is dependent on the choice of chunk size. In case of a large chunk size, there might be a possibility of multiple drifts and because of the averaging of posteriors, the intricate changes will get lost and those drifts might get unnoticed. A small chunk is also not ideal as this might lead to unnecessary triggering of temporality re-estimation because of outliers. Here the chunk size is set empirically to 5% of the total samples for best results. In an online data stream scenario the user might not have control over the chunk size, then a partition (of large chunk) or grouping (of small chunks) should be performed. In

future there should be research on an improved model which will be independent of chunk size.

5.4 Performance evaluation of LEMON

In this section we demonstrate how the knowledge of temporality enhances the performance of a regular predictive model. ATNN (RBF network within LEMON framework) is applied on three real world time series forecasting problems. For these problems the entire datasets are available hence to mimic an online scenario the datasets are divided into equal sized chunks and fed them sequentially to the network one by one for training; afterwards the test accuracy is measured.

5.4.1 Box-Jenkins problem

Detailed descriptions of the BJ data-set is already provided in the chapter 3 (refer to section 3.4.1.2).

Experimental Setup : During training, the samples are divided into equal sized chunks of 10 (i.e. 5% of total samples) to create an online scenario where the chunks are fed one by one without any revisit.

FIGURE 5.5: Temporal dependency estimation for BJ problem

FIGURE 5.6: Box Jenkins prediction using LEMON

Discussion : After the first two chunks are utilized to impart a baseline knowledge into ATNN, the temporality estimation process kicks in. The performance of LEMON in estimating temporality and drift can be observed in figure 5.5. Temporality of 3 (i.e. past 3 samples) is utilized throughout the training to update the weights of the underlying RBF network (with 30 hidden nodes). Once the weight learning is completed the testing is performed. Figure 5.6 plots the prediction performance in terms of actual vs predicted output during testing. Numerical details on performance matrices are provided in table 5.1, 5.2.

5.4.2 Wind speed forecasting

The wind speed prediction problem is already explained in the chapter 3 (refer to section 3.4.2.1).

Experimental Setup : The entire dataset is first divided into training set of 500 samples which consists of total 20 chunks with 25 samples in each to simulate the data stream. The test set comprises 100 samples. Here also a RBF network with 30 hidden nodes are utilized to build the model.

Discussion : Temporality is not an apriori knowledge for real world problems. LEMON is employed here with $W_{in} = 10$ and the temporality measure is found to be 1 at first. Followed by a drift, memory window evolves to $W = 7$ and temporality is re-estimated to be 4 and finally after another drift (making $W = 2$)

it is measured to be 1 again. Without LEMON the vanilla RBF network is able to produce a test RMSE of 0.138 whereas the same with LEMON provides a test RMSE of 0.130. The memory window and temporality measure along with drift indicator is provided in figure 5.7. The actual vs. prediction test plot is provided in figure 5.8. More numerical results are provided in table 5.1 and 5.2.

FIGURE 5.7: Temporal dependency estimation for Wind problem

FIGURE 5.8: Wind speed prediction using LEMON

5.4.3 Stock price forecasting

The stock price prediction problem is already described in the chapter 3 (refer to section 3.4.2.2).

Experimental Setup : The training dataset is first divided into 31 chunks with (each chunk with almost 5% of the total training samples set empirically like previous examples) to mimic the incoming online data stream. The same RBF network with 30 hidden nodes (as in the other experiments) is employed here as well to build our model on.

Discussion : The memory window W_{in} is initialized with 10 and the temporality is found to be 6 at first. During first drift the memory window changed to 5 (representing fast drift) and the temporality is found to be 2. Later in the training another drift is detected which makes the window $W = 5$ and temporality measure to be 3. After training is done testing is executed. Normalized test RMSE with a 3 layer RBF network (with 30 hidden nodes) is found to be 0.0236 whereas with LEMON, the same RBF exhibits a 30% improvement and a test RMSE of 0.0164.

FIGURE 5.9: Temporal dependency Estimation for stock problem

5.4.4 Numerical analysis

In this section a thorough numerical analysis of the experimental results is performed for mainly three purposes as listed below,

- To demonstrate that LEMON can enhance the predictive performance of an existing model (regular RBF network here) significantly

FIGURE 5.10: Stock price prediction using LEMON

- To show that the accuracy achieved by LEMON method is at par with state-of-the-art neuro-fuzzy time series models
- To present the speed of the proposed LEMON mechanism and thus affirm its online applicability

5.4.4.1 Performance improvement

The prediction accuracy (during test) are recorded in terms of Root Mean Square Error (RMSE) in table 5.1 for all the real-world problems. Here the result is presented in a before-after format where the test RMSE is presented for both cases i.e. with and without implementing LEMON.

Remark 5 : From table 5.1 it is clear that the proposed LEMON method is able to improve the predictive accuracy of the regular radial basis function (RBF) network significantly for all three problems.

TABLE 5.1: Performance improvement with LEMON

RBF with LEMON			
Problem	Temporality Measure	Training Time (in S)	Test RMSE
Box Jenkins Problem	3	0.120	0.040
Wind Speed Prediction	1,4,1	0.195	0.130
Stock Price Prediction	6,2,3	0.524	0.016
RBF without LEMON			
Problem	Temporality Measure	Training Time (in S)	Test RMSE
Box Jenkins Problem	NA	0.105	0.062
Wind Speed Prediction	NA	0.155	0.138
Stock Price Prediction	NA	0.450	0.024

5.4.4.2 Performance comparison

Next, the test RMSE is compared with popular neuro-fuzzy based time series methods to demonstrate LEMON's applicability and superiority. As discussed throughout this thesis, neural-fuzzy systems have propelled the time series research to a new high. Neural-fuzzy techniques utilize the data driven learning abilities of neural network and superior uncertainty handling capability of fuzzy logic system (stemming from the use of linguistic variables) making them highly efficient in modelling non-stationary time series data with noise, impreciseness and uncertainty. Besides RBF network are known to have functional similarities/ equivalence with Gaussian type 1 fuzzy systems. Hence the performance of LEMON is evaluated against some of the state-of-the-art neuro-fuzzy inference systems.

Methods for comparison : In this chapter the comparison is restricted to eight well known state-of-the-art neural fuzzy methods, i.e. evolving Takagi-Sugeno model (eTS) [52], Simplified eTS (SimpleTS) [53], Self Adaptive Fuzzy Inference System (SAFIS) [58], Meta cognitive Fuzzy Inference System (McFIS) [59], Parsimonious

Network based on Fuzzy Inference System (PANFIS) [60], Generic Evolving Neuro-Fuzzy Inference System (GENEFIS) [61], Spatio-Temporal Fuzzy Inference System (SPATFIS) [88] and Neural Fuzzy Inference System with Dynamic Neuron (NFIS-DN) [97]. These methods are picked from literature for their exceptional time series forecasting and online learning capability. They are rebuilt in MATLAB 2016 (same as in LEMON) for the purpose of a fair comparison. The performance metrics used here are test RMSE and training time.

TABLE 5.2: Performance of LEMON on BJ problem

Problem	Network Model	Test RMSE	Time(s)
BJ	eTS [52]	0.049	0.7
	SimpleTS [53]	0.055	0.8
	SAFIS [58]	0.071	1.5
	McFIS [59]	0.045	0.4
	PANFIS [60]	0.070	0.5
	GENEFIS [61]	0.050	0.6
	SPATFIS [88]	0.045	0.3
	NFIS-DN [97]	0.033	1.8
	LEMON	0.040	0.1

TABLE 5.3: Performance of LEMON on wind speed problem

Problem	Network Model	Test RMSE	Time(s)
Wind	eTS [52]	0.380	0.5
	SimpleTS [53]	0.394	0.4
	SAFIS [58]	0.376	0.7
	McFIS [59]	0.230	0.9
	PANFIS [60]	0.190	1.0
	GENEFIS [61]	0.153	0.3
	SPATFIS [88]	0.140	0.2
	NFIS-DN [97]	0.135	2.1
	LEMON	0.130	0.2

TABLE 5.4: Performance of LEMON on stock price problem

Problem	Network Model	Test RMSE	Time(s)
Stock	eTS [52]	0.070	1.4
	SimpleTS [53]	0.059	2.3
	SAFIS [58]	0.071	1.6
	McFIS [59]	0.036	1.5
	PANFIS [60]	0.049	2.0
	GENEFIS [61]	0.036	0.6
	SPATFIS [88]	0.020	0.5
	NFIS-DN [97]	0.019	1.7
	LEMON	0.016	0.5

Remark 6 : From the results provided in table 5.2,5.3,5.4 it is evident that the RBF network coupled with LEMON is able to achieve state-of-the-art performance (in most cases) in spite of learning online in a sequential manner. Moreover the training time for LEMON is consistently less than the other methods which further affirms its efficacy as a fast learning mechanism.

5.5 Extension of LEMON with Neuro-Fuzzy Inference System

In the previous section, we discussed in detail how to estimate temporal dependency or temporality using a Bayesian probabilistic framework and how to use that within a neural-network platform. In the following section, I develop a new neuro-fuzzy inference system that utilizes the temporality estimated from LEMON. Therefore it can handle both the system dynamics and the uncertainty associated with time series really well. The network is referred to as Bayesian Neuro-Fuzzy Inference System or BaNFIS. Two variants of BaNFIS is proposed here i.e. Global and Local. Next, we discuss briefly the architecture, variants and performance of BaNFIS.

5.5.1 BaNFIS architecture

BaNFIS is built on a basic 5 layer structure with an input layer, fuzzification layer, rule firing layer, normalization layer, and output layer (as in [58]) to model the

following zeroth order Takagi-Sugeno (TS) fuzzy rule,

Rule r : If $(x_1 \text{ is } \phi_{1r})$ and $\dots$ and $(x_n \text{ is } \phi_{nr})$ then $(\hat{y}_r \text{ is } w_r)$ where $\mathbf{x} = [x_1, \dots, x_n]^T$ are the crisp input features, $[\phi_{1r}, \dots, \phi_{nr}]$ are the fuzzy membership values and w_r is the weight value which acts as the consequent $(\hat{y}_r)$ for the corresponding r^{th} rule.

5.5.1.1 Global BaNFIS

Once the basic structure of BaNFIS is defined, the Temporality Estimation (of LEMON) can be utilized to introduce recurrence into the structure in two ways. In the first category, the temporal dependency is estimated on a global scale to estimate the number of past samples to be fed as input to BaNFIS. This is referred to as Global BaNFIS.

FIGURE 5.11: Global BaNFIS Architecture

With global BaNFIS we bring in the temporality in the input space. The motivation of such an arrangement is to address the problem of system dynamics in a traditional auto-regressive (AR) manner. In an AR process past instances of

outputs (and inputs) are fed to the model as inputs for the prediction task and the same holds true for the global BaNFIS as well. The diagram for the global BaNFIS is provided in figure 5.11

5.5.1.2 Local BaNFIS

The other variant is Local BaNFIS. In this variant, the temporality is estimated by treating the normalized firing strengths as inputs to an additional Gaussian hidden layer (hidden nodes with Gaussian activation). Here the number of required past instances (of the normalized firing strength vector) to be retained for the prediction task, is estimated locally hence named Local BaNFIS. Network diagram of local BaNFIS is provided next in figure 5.12.

FIGURE 5.12: Local BaNFIS Architecture

To perform local Temporality Estimation (TE) first, an Additional Gaussian Layer (AGL) is inserted before the output layer as shown in figure 5.4. The AGL here acts like a hidden layer between $\mathbf{F}(k)$ as input and $\hat{y}(k)$ as the output. Thus TE can be performed locally here on the consequent part of the network alone (marked in

green in figure 5.4). Local TE follows the same principle as a regular TE (discussed in LEMON) and estimates the temporal dependency for $\mathbf{F}(k)$ to be n . Once TE is over, local BaNFIS utilizes $[\mathbf{F}(k), \mathbf{F}(k-1) \cdots \mathbf{F}(k-(n-1))]$ as input to the additional Gaussian layer for rest of the chunks for learning.

Local BaNFIS can be thought of as a recurrent neuro-fuzzy network with recurrence in its hidden layer (here AGL). Finite past instances of the latent outputs (normalized firing strengths : $\mathbf{F}(k), \mathbf{F}(k-1) \cdots \mathbf{F}(k-(n-1))$) are utilized locally to trace the intricate system dynamics. Local BaNFIS also differs from the global version in terms of the additional layer which theoretically makes it even better in non-linear function approximation problems. Thereby, it also requires much less number of fuzzy rules as the non-linearity can be handled by the AGL.

Remark 7 : Please note that the temporality estimation process and the parameter tuning method for BaNFIS will remain identical to LEMON.

5.5.2 BaNFIS performance evaluation

To evaluate the performance of BaNFIS compared to popular neuro-fuzzy systems, we have utilized the same problems as before i.e. the Box-Jenkins Problem, Wind Speed Forecasting Problem and Stock-Price Forecasting Problem for consistency. On top of that we also employ two large scale time series problems to compare its performance against other Machine Learning (ML) and Deep Learning (DL) methods.

5.5.2.1 Standard real-world problems

BaNFIS (like LEMON) is evaluated with the Box-Jenkins (BJ), Wind speed and Stock price forecasting problems. Details of these problems are already available in section 3. The prediction performance is measured in terms of Root Mean Square (RMSE) and training time (in Seconds). Both Global and Local BaNFIS are employed to perform this forecast and comparisons are made with state-of-the-art neuro-fuzzy methods in table 5.5-5.7.

TABLE 5.5: Performance of BaNFIS on BJ problem

Problem	Network Model	Test RMSE	Training Time (s)
BJ	eTS [52]	0.049	0.7
	SimpleTS [53]	0.055	0.8
	SAFIS [58]	0.071	1.5
	McFIS [59]	0.045	0.4
	PANFIS [60]	0.070	0.5
	GENEFIS [61]	0.050	0.6
	SPATFIS [88]	0.045	0.3
	NFIS-DN [97]	0.033	1.8
	Global BaNFIS	0.043	0.2
	Local BaNFIS	0.063	0.3

TABLE 5.6: Performance of BaNFIS on wind speed problem

Problem	Network Model	Test RMSE	Training Time (s)
Wind	eTS [52]	0.380	0.5
	SimpleTS [53]	0.394	0.4
	SAFIS [58]	0.376	0.7
	McFIS [59]	0.230	0.9
	PANFIS [60]	0.190	1.0
	GENEFIS [61]	0.153	0.3
	SPATFIS [88]	0.140	0.2
	NFIS-DN [97]	0.135	2.1
	Global BaNFIS	0.136	0.4
	Local BaNFIS	0.133	0.3

From the tables it can be seen that both variants of BaNFIS achieve significant (around 5-10%) improvement over the other neuro-fuzzy methods (non-Bayesian) across three standard problems of online time series forecasting. It is also observed that Global BaNFIS performs better with problems of auto-regressive (where past instances of outputs need to be fed as input directly) nature whereas local BaNFIS is generally faster as it requires less number of fuzzy rules.

TABLE 5.7: Performance of BaNFIS on stock price problem

Problem	Network Model	Test RMSE	Training Time (s)
Stock	eTS [52]	0.070	1.4
	SimpleTS [53]	0.059	2.3
	SAFIS [58]	0.071	1.6
	McFIS [59]	0.036	1.5
	PANFIS [60]	0.049	2.0
	GENEFIS [61]	0.036	0.6
	SPATFIS [88]	0.020	0.5
	NFIS-DN [97]	0.019	1.7
	Global BaNFIS	0.016	0.5
	Local BaNFIS	0.044	0.4

5.5.2.2 Large-scale real-world problems

In this section, we employ two different large scale time series problems to demonstrate the efficacy of the proposed BaNFIS model. In previous chapters we have compared the performance of the proposed models with small scale time-series problems hence other neuro-fuzzy methods were used as peers. However, for large scale time series problems, traditional Machine Learning and Deep Learning methods are much more effective hence they are used here as benchmarks.

Financial indicator prediction : The financial indicator (i.e. stock price) problem deals with the prediction of daily opening stock prices with the previous day's information. The data is collected from 50 different organizations (from ten different sectors) for a period of 2007-2016. The dataset is downloaded directly from the Github repository ². We utilize 2007-2014 data for training, 2015 data for validation and 2016 data for testing for each of these 50 cases. More details on the data are available in [113]. During testing, test Mean Squared Error (MSE) is computed for each of these fifty stock time series and their overall average is reported as the performance index in table 5.8. To train the BaNFIS models, previous day's high price, low price, close price and volume of sale as the input features $\mathbf{u}(k)$ are employed to predict next day's opening price.

²<https://github.com/z331565360/State-Frequency-Memory-stock-prediction>

Power consumption prediction : The power consumption pattern in a household in France is predicted in this large scale forecasting problem. The dataset is downloaded from UCI-ML repository ³ and it contains the details of electric power consumption pattern in one household in France over 4 years with a sampling rate of one minute. For simplicity however, only 2010's data is utilized in this experiment. The voltage sequence is the target time series as per the standard practice whereas the sub-meter readings and active power are used as external features. More details on the problem and train-validation-test partition can be found in [42].

Baseline ML/DL Methods for comparison The above two problems are large scale in nature hence for performance comparison we have utilized popular machine learning and deep learning methods. Next, these methods are briefly described below.

- **Naive** approach as the name suggests considers the previous output as the prediction.
- **ARIMA** [1] is a well-known classical method of time series prediction as introduced in the literature.
- **Support Vector Regression (SVR)** is the modified version of support vector machine for regression. More details on the design of SVR is available in [114, 115].
- **Artificial Neural Network (ANN)** are popular in the machine learning community for their data-driven learning capabilities as reviewed in detail in the literature section. In this study we utilize ANN from [116].
- **Long Short Term Memory (LSTM)** is a variant of recurrent neural network mainly utilized for language modeling which can also be adopted for time series. More details on the design are available in [31].
- **Convolutional Neural Network (CNN)** are widely practised in the domain of computer vision and image classification. It can perform time series forecast as well, with convolution on wavelet transformed time series i.e. scalogram ([117]).

---

³<https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption>

The experiments are performed both for 1 step and 5 step ahead time series predictions. We utilize test MSE (Mean Squared Error) to compare the performances as per the standard practice with these datasets.

TABLE 5.8: Performance of BaNFIS on large scale time-series problems

Model	Stock	MSE	Power	MSE
	1 step	5 step	1 step	5 step
Naive	5.62	28.78	0.42	1.67
ARIMA [1]	5.70	34.33	0.42	1.60
SVR [115]	5.58	28.69	0.42	1.55
ANN [116]	5.60	28.69	0.41	1.61
LSTM [31]	5.60	28.49	0.40	1.59
CNN [117]	5.65	28.29	0.42	1.54
Global BaNFIS	5.58	28.50	0.40	1.60

Discussion : The advantages of the proposed Bayesian Neuro-Fuzzy Inference System (BaNFIS) are apparent from the numerical results reported in table 5.8. From the table, we observe that the one and five step ahead prediction performances from BaNFIS are adequate. The deep learning methods require a huge number of parameters to train over many epochs whereas FTFNIS learns in one epoch with only around hundreds of fuzzy rules and still manages to provide good accuracy. All results (other than of BaNFIS), are quoted from recent a literature [42].

5.6 Limitation of LEMON/ BaNFIS

So far the discussions in this thesis have focused heavily on proposing ways to bring past information or recurrence either in the framework of a neural network (as shown with LEMON) or a neuro-fuzzy system (SPATFIS, NFIS-DN, BaNFIS) in order to capture the intrinsic dynamics of the time series data. However, utilization of past outputs only to model a time series (in an auto-regressive manner) can lead to a rather serious problem that often goes unnoticed hence unaddressed in the literature. A network based time series model trained with historical data alone

can only provide a lagged time series where the predicted sequence is always trailing the actual sequence. This results in the very poor forecasts in terms of movement predictions (especially in the financial market) but not so much in terms of MSE/RMSE error therefore it goes overlooked very often. In the next chapter, we define this problem in greater detail and propose a hybrid dual network solution to address this fundamental issue of time series modeling.

Chapter 6

Dual Network Solution (DNS)

‘Lag’ in the predicted sequence can be a predominant issue in time series forecasting. Unfortunately, this does not contribute to a high prediction error (i.e. a high MSE/ RMSE) hence it is often overlooked in most of the literature. However, it can lead to a rather poor forecast in terms of movement prediction. In this chapter, we tackle this fundamental problem of time series modeling with a novel trend driven mechanism. Trend, defined as the inherent pattern of the data, is extracted here and utilized next to perform a lag-free forecasting. A Dual Network Solution (DNS) is proposed in this chapter, where the first network predicts the trend and the second network utilizes that predicted trend along with its past information to capture the dynamical behavior of the time series. DNS exhibits a substantially improved performance compared to more complex and resource-intensive state-of-the-art algorithms in large scale regression problems. Apart from the traditional Mean Squared Error (MSE), we also propose a novel Movement Prediction Metric or MPM (for detection of lag in time series) as a new complementary performance metric to evaluate the efficacy of DNS better.

1

---

¹The contents of this chapter is published in the following conference, [118] Subhrajit Samanta et. al ”A Dual Network Solution (DNS) for lag-free time series forecasting”, International Joint Conference on Neural Networks (IJCNN 2020)

6.1 Problem of ‘Lag’ in time series forecasting

A time series model utilizing historical data alone can detect a change in the trend only after the historical data also experiences the same change. Naturally, the change prediction is late which leads to the problem of lagging/ trailing. To explain this problem further we take an example case of wind speed dataset ² (as described in chapter 3). The prediction task is performed here with a Multi-Layer Perceptron network or MLP (100 hidden node with Sigmoid activation in the single hidden layer) with only past values of the target sequence as inputs and the actual vs prediction plot is provided in figure 6.1. We see that, from time instance 25 to 31 in figure 6.1b the trend is upward which leads the MLP to predict an upward trend at point 31 (marked in the figure 6.1b) in spite of an actual downward trend. This results in a predicted sequence with a distinct lag (or trail) as seen in the figure 6.1a and 6.1b.

The lag in the predicted sequence results in a ‘close-by, past value’ prediction hence it does not contribute to a high prediction error such as MSE. Besides, MSE does not take into account the movement direction of the prediction (as the squared error ignores the sign of the error), therefore it is unable to detect the presence of lag. For instance, the wind forecast in figure 6.1a looks like a good forecast however suffers from the lag problem. Sometimes due to over-fitting of the historical values, the result can be precisely a curve that mimics the real values almost perfectly, with just a little delay i.e. ‘some time late’. For further confirmation, it is sufficient to observe the curve in the proximity of the local maximums (i.e near point 31 in figure 1b). Let us take another example of stock price prediction, a lag of such nature can lead the forecast to predict an upwards movement when the actual movement is already downwards, thus possibly leading to a financial loss thereby further affirming the significance of lag problem. This issue is very common throughout most time series literature and to the best of our knowledge has not been addressed properly yet as discussed in the literature review.

---

²<http://mesonet.agron.iastate.edu/request/awos/1min.php>

(A) Actual vs prediction plot

(B) Zoomed in to show Lag

FIGURE 6.1: Demonstration of ‘Lag’ in time series forecasting

6.2 Overview of the solution

First the trend is extracted here from the time series data, with two proposed methodologies i.e. Moving Average Method and Slope Difference Method. Then we utilize the extracted trend as the target sequence (instead of the actual values) in a supervised manner and learn to predict the trend. The predicted trend drives the final forecast.

FIGURE 6.2: Overview of the Dual Network Solution (DNS)

A novel Dual Network Solution (DNS) is proposed for this purpose. The first network (Trend Prediction Network) learns to map the exogenous input features/ factors to the extracted trend. The predicted trend (from the first network) along with its finite past instances are passed on to the second network (which uses LEMON framework) to learn the intricate details of the system dynamics. Therefore, we have a trend driven dual network system which is able to provide an improved forecast where the predicted curve follows the actual curve closely without trailing. An overview of DNS is provided in figure 6.2.

Besides, a new Movement Prediction Metric (MPM) is proposed as a complementary measurement to evaluate time series models better because most of the error based performance metrics (such as MSE/ RMSE) fail to detect the presence of lag in time series forecast.

6.3 Trend extraction

As mentioned in the overview, the trend needs to be extracted first to train the first network. Two separate methods can be employed for this purpose,

6.3.1 Moving Average Method (MAM)

We employ the principle of moving average on the time series sequence to get the trend. First, a window of size $W + 1$ is set. For each data point in the sequence, the W neighboring samples centering it are considered and their mean is obtained. This process is done for all the data points in the sequence i.e. $y(k)$. The extracted trend $y_T(k)$ is,

$$y_T(k) = \frac{1}{W+1} \sum_{i=-W/2}^{W/2} y(k+i) \quad (6.1)$$

The average over the neighboring samples helps in smoothing out the sequence hence MAM is able to get rid of noise and frequent fluctuations from the time series and extract the trend as shown figure 6.3. The bigger the window size the smoother is the trend curve. For example, to extract the trend in figure 6.3 a window size of 21 (i.e. $W = 20$) is utilized .

FIGURE 6.3: Trend in wind dataset extracted with MAM

Instead of mean, a median should be utilized to extract the trend if the time series data is prone to outliers.

6.3.2 Slope Difference Method (SDM)

The moving average method is offline in nature and requires all the training samples to be present. Hence it is not suitable in case of an online or sequential learning context, hence the second method is proposed.

First, the slope is computed for the current trend. With $y(k)$ and $y(k+1)$, we can represent the trend at k^{th} instance with slope $m_T(k) = \frac{y(k+1)-y(k)}{(k+1)-k} = y(k+1)-y(k)$. Let us assume a new $(k+i)^{th}$ sample arrives and makes a slope of $m(k+i)$ with the k^{th} sample. If the new slope is significantly different than $m_T(k)$ then SDM decides that there is a change in trend and reassign the latest slope made by $(k+i)^{th}$ sample as the current trend and so on. To summarize,

$$\begin{aligned} m_T(k+i) &= m(k+i) \quad \text{if} \quad |m(k+i) - m_T(k)| \geq \epsilon \\ &= m_T(k) \quad \text{if} \quad |m(k+i) - m_T(k)| < \epsilon \end{aligned} \quad (6.2)$$

with ϵ as the threshold. From the slope defined in equation 6.2 the trend can be retrieved easily using,

$$\begin{aligned} y_T(k+i) &= m_T(k) \cdot ((k+i) - i) + y(k) \\ &= m_T(k) \cdot i + y(k) \end{aligned} \quad (6.3)$$

FIGURE 6.4: Trend in wind dataset extracted with SDM

Figure 6.4 demonstrates an example of the extracted trend for the same wind dataset using the slope difference method with $\epsilon = 0.01$. It is clear that the trend extracted from SDM is much less smooth than MAM, however it can be useful in certain applications. Next, we discuss in detail the proposed dual network solution (DNS) as briefed in the outline.

6.4 Dual Network System

As outlined above we propose a hybrid dual network solution to address the issue of lag in predicted time series sequence. The two networks are referred to as Trend Prediction Network (N_1) and Temporal Network (N_2). The detailed descriptions of these network are provide next along with the training algorithm for these networks.

6.4.1 Trend prediction network (N_1)

The first network denoted with N_1 is accountable for mapping the exogenous input features (factors) to the extracted trend (as shown in the last section). Let us denote the feature vector as $\mathbf{u}(k) \in \mathbb{R}^P$ where the number of input features are P and k is the time instance. From previous section we already have the trend associated with the data points i.e. $y_T(k)$. Unlike regular time series model, here we utilize the extracted trend as the target sequence (i.e. $y_T(k)$ instead of $y(k)$) to train our network N_1 (which is feed-forward in principle). Thus the network learns to predict the trend $\hat{y}_T(k)$ from the exogenous input vector. Therefore the main task of N_1 becomes a non-linear regression problem (function approximation) and it can be written as,

$$\hat{y}_T(k) = f_1(\mathbf{u}(k)) \quad (6.4)$$

Remark 1 : One of the major advantages of DNS lies in the fact that any kind of neural network can be employed to build the N_1 platform (making DNS generic) as the primary goal here is to approximate the function $f_1(.)$ in equation 6.4 from the data (like any other supervised regression problem).

Network structure and update : For the experiments conducted in this study we have utilized a radial basis function [15] network as N_1 , primarily for its ease

of implementation along with its well suitability in time series modeling. Hence, N_1 here is a 3 layer network, with P input nodes, R hidden nodes and 1 output node which provides weighted ($\mathbf{w} \in \mathbb{R}^R$) summation of the Gaussian activations $\mathbf{F} \in \mathbb{R}^R$ such that $\hat{y}_T(k) = \mathbf{w}^T \mathbf{F}$. To train the network, a new mini-batch variant of projection based learning or PBL [87] method (Basic PBL is already introduced in Chapter 3) is proposed. First, we divide the training dataset into smaller batches and compute the prediction errors. The sum of the squared errors over a mini-batch (of size c) is used as the cost function here i.e. $J_1 = \frac{1}{2} \sum_c (y_T(k) - \hat{y}_T(k))^2 = \frac{1}{2} \sum_c (y_T(k) - \mathbf{w}^T \mathbf{F})^2$. Optimal weight $\mathbf{w}^*$ is obtained when the cost function is minimized or $\frac{dJ_1(\mathbf{w})}{d\mathbf{w}}|_{\mathbf{w}=\mathbf{w}^*} = 0$. Rearranging we get,

$$\mathbf{w}^* = A^{-1}B \quad (6.5)$$

where, $A \in \mathbb{R}^{RXR}$ consists of a_{rr^*} and $B \in \mathbb{R}^{RX1}$ of b_r s.t.

$$a_{rr^*} = \sum_c F_r \cdot F_{r^*} \quad , \quad b_r = \sum_c F_r \cdot y_T \quad (6.6)$$

for $r, r^* = 1, 2, \dots, R$. The proposed network N_1 learns from these mini-batches over multiple epochs to reach the optimal weights.

Remark 2 : A regular mini-batch gradient based backpropagation can also be adopted to train N_1 . However, the proposed mini-batch PBL method is adopted here, as it does not suffer from the problems of exploding or vanishing gradients.

6.4.2 Temporal network (N_2)

Generally historical values of the actual sequence are used to model a time series (AR type), however using them alone often leads to a ‘close by past value’ prediction with lag (as explained in previous sections). Hence in the second network (N_2) of the DNS, we utilize the predicted trend instead, to obtain the lag-free forecast. N_2 is made recurrent (temporal) in nature to capture the system dynamics efficiently. It utilizes the predicted trend $\hat{y}_T(k)$ from N_1 along with its past instances i.e. $\hat{y}_T(k-1), \hat{y}_T(k-2) \dots \hat{y}_T(k-n)$ to perform the final prediction,

$$\hat{y}(k) = f_2(\hat{y}_T(k), \hat{y}_T(k-1), \hat{y}_T(k-2) \dots \hat{y}_T(k-n)) \quad (6.7)$$

here $f_2(\cdot)$ is the nonlinear function representing the forecasting task and n is the number of past instances N_2 requires to perform the function approximation. Thereby, N_2 is also independent of the underlying neural network platform.

Remark 3 : We feed the predicted trend $\hat{y}_T(k)$ to N_2 instead of the actual trend $y_T(k)$ because, during testing (or application), the actual trend will not be available yet.

To design the temporal network, we need to know the required number of past instances n of $\hat{y}_T(k)$ (from equation 6.7). The Bayesian method of Temporality Estimation or TE is adopted from LEMON (as introduced in Chapter 5). A schematic of the TE is also presented in figure 5.1 for a quick recap. Once the temporal dependency is estimated we can conclude that N_2 requires n number of past instances (of the predicted trend) to approximate the function $f_2(\cdot)$ for the final forecast.

Network structure and update : To build N_2 another RBF network is adopted (similar to N_1). We feed the second RBF network with the predicted trend (from N_1) along with n number of its past instances i.e. $\hat{y}_T(k), \hat{y}_T(k-1), \hat{y}_T(k-2) \cdots \hat{y}_T(k-n)$ to build N_2 . It learns the inherent dynamics of the time series from the historical data using the mini-batch PBL method. The function used here for learning is $J_2 = 1/2 \sum_c (y(k) - \hat{y}(k))^2$. Therefore N_2 provides a final forecast which is driven by the trend hence the predicted sequence we get is lag-free.

6.5 Performance evaluation of DNS

In this study, an RBF network with 100 hidden nodes is adopted to build the first network N_1 . A second RBF network (also with 100 hidden nodes) is used to build N_2 . Temporality Estimation (TE) process (with $M = 10$ and $\alpha = 0.9$) estimates the number of past instances (of the predicted trend) required i.e. n to design N_2 . First two mini-batches are used for this purpose (first one for preliminary knowledge, next one for the TE). A mini-batch size of ($c = 50$) is used to train both N_1 and N_2 with the mini-batch PBL method over 100 epochs. We explore the effect of both MAM (with varying window size $W + 1$) and SDM (with varying threshold ϵ) on the final forecast. For the actual vs predicted plots in figure 6.5, figure 6.6, figure 6.7 and performance comparison in table 6.5, we pick the best performing ones between MAM and SDM.

6.5.1 Discussion on hyperparameter setting

The initial memory window M is the maximum number of past samples to examine during TE hence the estimated dependency will always be less than M i.e. $n \leq M$. A smaller or bigger memory window can lead to different measurements of n (because of the distribution of the priors and posteriors in the memory changes with its width). Here $M = 10$ provided the best accuracy. The priors are updated with last posteriors as shown in equation 5.5 and for that a prior regulating factor α of 0.9 is used. A higher α is employed so that the prior values do not become very similar to the posteriors, only after a few samples. A Pareto inspired approach is adopted here to estimate the temporal dependency where past instances accounting for 80% of the posteriors are regarded as the measurement of past dependency. A threshold of less or more than 80% can also be utilized for the same however they might lead to under or over-estimation (For example, at 70%, TE might lead to underestimation). For more details on this process please refer to chapter 5 where LEMON is introduced in detail.

6.5.2 Performance indicators

As per standard practice in the time series literature, we utilize Mean Squared Error or MSE to compare the performance of DNS with other state-of-the-art methods during testing. Mean squared error is defined as the average of the squared prediction errors,

$$MSE = \frac{1}{S} \sum_S (y(k) - \hat{y}(k))^2 \quad (6.8)$$

with S as the number of samples. However, MSE (or RMSE) cannot be the sole indicator of performance as it often fails to trace the presence of lag in the predicted sequence (as explained elaborately in previous section 1.1). Hence, in addition to MSE, we propose a novel Movement Prediction Metric (MPM) as a complementary measurement.

Proposed MPM: MSE does not consider the direction of prediction as it squares the error. Hence, we propose an indicator which is not error based rather it looks only at the prediction movement. If $y(k+1) \geq y(k)$ then it is considered as a real upward movement and vice versa. If DNS prediction : $\hat{y}(k+1) \geq \hat{y}(k)$ then it is counted as a True Up prediction or TU. Similarly True Down (TD), False Up

(FU) and False Down (FD) predictions can be counted and we can propose the Movement Prediction Metric (MPM) as,

$$MPM = \frac{TU + TD}{TU + FU + TD + FD} \quad (6.9)$$

If there is lag happening then the predicted curve will fail to track the actual curve suitably in the presence of optimums, hence it will lead to an overall low MPM value. For a highly over-fitted prediction mimicking the historical data as in figure 6.1a, the MPM value hovers around 50%. Conversely, Higher MPM value indicates to an improved forecast without lag.

Next, the aforementioned wind dataset is employed to demonstrate the improved lag-free forecasting capability of DNS. The efficacy of MAM and SDM on the final prediction performance is also explored here. In the second category, we employ two real-world large scale time series problems (as introduced in BaNFIS performance evaluation) namely financial indicator and power consumption prediction to show the superior performance achieved with our method compared to other state-of-the-art ML/ DL algorithms.

6.5.3 Demonstration of lag-free forecast

We collect wind speed data along with the directional information for February 1, 2011, to February 28, 2011, containing 600 samples from the Iowa (USA) Department of Transport website. More details can be found in [95]. We have used the directional information as the feature $u(k)$ to map the extracted trend in N_1 . TE estimates $n = 4$ for the second network.

Moving Average Method (MAM) : Different sizes of windows (from small to large) are employed to extract the trend and utilized in N_1 . Predicted trend along with its historical data are utilized in N_2 . The corresponding MSE and MPM are recorded in table 6.1.

We see that a larger window size results in higher MSE, however the decrease in the movement prediction is not very drastic. A window size of 5 produced the best $MPM \approx 75\%$, indicating an improved lag-free forecast.

TABLE 6.1: Performance of DNS on wind speed problem with MAM

Window size ($W + 1$)	MSE	MPM (in %)
3	0.0064	74.10
5	0.0077	74.46
11	0.0086	74.10
21	0.0164	69.46

Slope Difference Method (SDM) : Similarly, the slope difference method is utilized with varying threshold values to evaluate its efficacy on the forecasting performance of the proposed dual network system. The results are provided in table 6.2.

TABLE 6.2: Performance of DNS on wind speed problem with SDM

Difference Threshold (ϵ)	MSE	MPM (in %)
0.001	0.0059	78.03
0.005	0.0061	77.85
0.010	0.0064	77.67
0.050	0.0077	74.43

The extracted trend from SDM is not as smooth as the trend extracted from MAM. The trend here follows the actual curve more closely than MAM when the threshold is low. For the wind problem, SDM provided a better MSE and MPM as shown in table 6.2. However with SDM, the result deteriorates faster with higher value of the threshold ϵ as with high threshold value, a lot of slope change information gets ignored (from equation 6.2-6.3). In figure 6.5A the actual vs predicted plot is provided for the best case from the use cases (i.e. SDM with $\epsilon = 0.001$) whereas figure Figure 6.5B shows the lag-free improved forecast in a zoomed format.

6.5.4 Performance comparison

In this section, we employ two different large scale time series problems to show the superiority of the proposed DNS model compared to other state-of-the-art time series methods.

(A) Performance of SDM with $\epsilon = 0.001$

(B) Zoomed in to show lag-free forecast

FIGURE 6.5: Wind speed prediction with DNS

6.5.4.1 Financial indicator prediction

This problem deals with the prediction of daily opening stock prices with the previous day's information. The details of this problems are already discussed in chapter 5 in section 5.5.2.2. The stock data of 50 different organizations (from ten different sectors) is collected for a period of 2007-2016. The dataset is downloaded

directly from the Github repository ³. The data for 2007-2014 is used for training, 2015 data for validation and 2016 data for testing. More details on the data are available in [113]. During testing, test MSE is computed for each of the fifty stock time series and their overall average is reported as the performance index in table 6.5.

To train the first network we have utilized previous day's high price, low price, close price and volume of sale as the input features $\mathbf{u}(k)$ to map the trend (of next day's opening). Three past instances are utilized in N_2 (from TE) for the final prediction. We pick one stock (Amazon) out of the 50 stocks to present the efficacy of the trend extraction methods in terms of MSE (non-normalized) and movement prediction in table 6.3. A window size of 3 produced the best movement prediction for Amazon stock price prediction. Sections of these actual vs. predicted plots are provided in figure 6.6A and 6.6B.

TABLE 6.3: Performance of DNS on single step Amazon stock price prediction

MAM Window ($W + 1$)	Test MSE	Test MPM in%
3	3.512	77.00
5	3.636	76.00
SDM Threshold (ϵ)	Test MSE	Test MPM in %
0.001	6.813	70.00
0.003	11.213	58.00

6.5.4.2 Energy consumption prediction

The energy consumption pattern in a household in France is predicted in this forecasting task. This problem is already described in detail in chapter 5 in section 5.5.2.2. Part of the data is downloaded from UCI-ML repository⁴ and it contains the details of electric power consumption in one household for the year of 2010. The voltage sequence is the target time series as per the standard practice. More details on the problem and train-validation-test partition are in [42].

We utilize sub-meter readings and active power as external features to map the trend. TE estimates $n = 5$ for N_2 . Performance of MAM and SDM on this single

³<https://github.com/z331565360/State-Frequency-Memory-stock-prediction>

⁴<https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption>

(A) Training plots for Amazon stock

(B) Testing plots for Amazon stock

FIGURE 6.6: Amazon stock price prediction with DNS

step ahead prediction are provided in table 6.4. SDM with $\epsilon = 0.001$ provides the best result here and the corresponding actual vs predicted curves (for a section of the data) are provided in Figure 6.7A and Figure 6.7B. The full Performance comparison is provided in table 6.5.

Remark 4 : Stock price data often contains a higher amount of volatility (therefore the frequent fluctuations as can be seen in figure 6.6A) compared to the power consumption data. Hence MAM is more suitable for financial indicator data to

TABLE 6.4: Performance of DNS on single step energy consumption prediction

MAM Window ($W + 1$)	Test MSE	Test MPM in %
3	0.223	92
5	0.461	90
SDM Threshold (ϵ)	Test MSE	Test MPM in %
0.001	0.272	94
0.010	0.354	92

(A) Training plots for power consumption

(B) Testing plots for power consumption

FIGURE 6.7: Power consumption prediction with DNS

extract the trend. The past dependency measure is also lesser for the stock problem, signifying a low reliance on long past which is intuitive considering the swift changing nature of the stock market.

6.5.4.3 Baseline methods for comparison

For the purpose of performance comparison, we resort to MSE as per the standard practise in time series literature. We provide test MSE for short term (1 step ahead) and long-term (5-step ahead) predictions. The baselines are described briefly,

- **Naïve** approach as the name suggests considers the previous output as the prediction.
- **ARIMA** [1] is a well-known classical method of time series prediction.
- **Support Vector Regression (SVR)** is the modified version of support vector machine for regression. More details on the design is available in [114, 115].
- **Artificial Neural Network (ANN)** are popular in the machine learning community for their data-driven learning capabilities. In this study we utilize ANN from [116].
- **Long Short Term Memory** is a variant of recurrent neural network mainly utilized for language modeling. More details on the design are available in [31].
- **State Frequency Memory (SFM)** [35] separates dynamical patterns in the frequency domain for detailed time-series analysis.
- **Attentive Neural Network** [42] traces local time-frequency features along with the long-term pattern to perform prediction.
- **Convolutional Neural Network (CNN)** are widely practised in the domain of computer vision and image classification. Here it performs forecast with convolution on wavelet transformed time series i.e. scalogram ([117]).
- **Ensemble of CNN and LSTM** as the name suggests, fuses these two networks (LSTM and CNN) to perform time series forecasting as introduced in TreeNET[119].
- **LEMON** [108] as introduced in chapter 5, LEMON employs Gaussian activation in its hidden layer. The RBF network utilizes the required past instances (estimated from TE) of target value as input i.e. $y(k-1), y(k-2) \cdots y(k-n)$ to train the network and perform prediction.

TABLE 6.5: Performance of DNS on large scale time-series problems

Model	Stock	MSE	Power	MSE
	1 step	5 step	1 step	5 step
Naive	5.62	28.78	0.42	1.67
ARIMA [1]	5.70	34.33	0.42	1.60
SVR [115]	5.58	28.69	0.42	1.55
ANN [116]	5.60	28.69	0.41	1.61
LSTM [31]	5.60	28.49	0.40	1.59
CNN [117]	5.65	28.29	0.42	1.54
SFM [35]	5.57	28.90	-	-
Attention NN [42]	5.57	28.22	0.40	1.54
TreeNET [119]	5.61	28.46	0.41	1.56
LEMON [108]	5.58	28.50	0.40	1.60
DNS	4.56	24.16	0.27	1.48

The purpose of the last comparison is to emphasize on the performance improvement achieved when the same RBF network (with LEMON mechanism) is utilized in the proposed dual network framework. The detailed numerical results are provided in table 6.5.

6.5.5 Numerical analysis and discussion

The advantages of the proposed dual network solution is apparent from the numerical results reported in table 6.5. The results of the baseline methods are produced from recently published literature of [42]. We use the exact same train-validation-test partition as in [42] on the dataset to ensure fairness of comparison. From the table 6.5, it can be observed that proposed DNS consistently outperforms the baseline and the state-of-the-arts methods by significant margins (around 10% lesser MSE from the next best model). The RBF network (100 hidden nodes) with past dependency estimated (i.e. LEMON) performs second best among the competing methods, however, it cannot attain a good movement prediction and the best MPM it achieves is around $\approx 48 - 52\%$. On the other hand with DNS the same

RBF network can provide a **MPM** $\approx 77 - 94\%$ as shown in table 6.3 and 6.4. This further indicates to an overall improved lag-free forecast attained by the same network when adopting DNS.

The superior performance of DNS can be attributed to the followings,

- N_1 maps external features to the extracted trend (with MAM or SDM) which allows DNS to predict a sudden change in trend direction even when the consecutive past movements have been similar in nature thus **high MPM** is achieved (table 6.3 and 6.4).
- N_2 utilizes the predicted trend from N_1 to drive the forecast. Thereby the predicted curve follows the actual curve closely in a lag-less manner hence, DNS achieves **lower MSE** than its peers.
- TE helps to reduce unwanted under or over-fitting (like utilization of less or more past instances than required can lead to those).
- DNS is advantageous in terms of computational cost as well. Compared to the resource-heavy deep networks (i.e. LSTM, CNN, CNN-LSTM), DNS only learns 200 weights (N_1+N_2) over 100 epochs.

6.5.6 Limitation of DNS

A regular time series model can detect a change in the trend only when the historical data has also gone through the same, hence the prediction is always late leading to the problem of lag. In this chapter, we have presented a solution where external features are employed to predict the trend behavior. The predicted trend is then utilized to drive the forecast closely with the target sequence in a lag less manner. Thus the Dual Network Solution provides a better forecast both in terms of low MSE and better movement prediction (high MPM) when compared to other methodologies. However, DNS is still offline in nature and requires all the data points to be present to learn the trend behavior. The two networks in DNS are also hand-crafted. Therefore DNS, is not really suitable in an online time series scenario. Therefore, in the future, I plan to build a self-evolving variant of DNS capable of handling online time series forecasting and data stream problems without the problem of lag.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, an in-depth description is provided on the development of different neuro-fuzzy and neural network frameworks to address some of the major challenges of time series forecasting. Starting with SPATFIS and how it addresses both uncertainty and the system dynamics with its memory neurons then we moved on to chapter 4 where we explained how the dynamic neurons in NFIS-DN address the issue of infinite past prominent in SPATFIS. In chapter 5 a Bayesian approach for estimation of temporal dependency is presented in an online manner and finally chapter 6 addresses the issue of ‘lag’ in the predicted time series sequence. The conclusions from these chapters are summarized below,

In Chapter 3 we have explored a self-adaptive recurrent neural fuzzy system capable of swift online learning, dubbed as SPATFIS. It employs memory type neurons in the crisp layers to capture the system dynamics effectively. The efficacy of SPATFIS in online time series forecasting is exhibited with the aid of two benchmark and two real-world problems. In all the aforementioned scenarios SPATFIS exhibits superior execution time with competitive predictive accuracy while maintaining an economical fuzzy rule base. The results produced in this chapter are statistically proven to be significant as well. Stability is thoroughly analyzed here and SPATFIS is confirmed to be stable throughout both the structure and parameter learning process. However, SPATIFS fails to adapt to rapidly changing system dynamics as

the memory neurons retain the effect of an infinite past. To address this problem we developed NFIS-DN.

In Chapter 4, we have proposed a novel dynamic neuron that retains only the required amount of past information. The dynamic neurons make the proposed neural fuzzy network (NFIS-DN) highly capable of tracking quickly changing temporal behavior. Therefore it is able to address the issue of infinite past introduced by the memory neurons in SPATFIS. Over-fitting during training is avoided here with the proposed self-regulation technique. The end result is a superior algorithm with high predictive accuracy and a parsimonious rule base (i.e. less complexity). However, the required amount of past information is estimated here in an empirical manner therefore making the process batch in nature. Hence, we introduce the LEMON mechanism to estimate the temporal dependency in an online fashion.

In Chapter 5, a novel Bayesian approach is proposed for estimating temporal dependency and detecting temporal drift. Employing this knowledge, LEMON is able to improve the predictive performance of an existing model significantly and achieve state-of-the-art accuracy. The fast, autonomous and online nature of the proposed method enables it to be highly suitable for real-time online time series prediction as well. In BaNFIS, we extend the LEMON framework into a neuro-fuzzy network with two novel variants (Global and Local BaNFIS). Both variants are able to not only handle the system dynamics but also the uncertainty and impreciseness of the real-world time series data.

In the first three chapters, we have addressed four prominent challenges of time series modeling (capturing system dynamics, handling uncertainty, online learning and temporal dependency estimation). However, all of these network-based time series models suffer from one common problem. When utilizing only past instances the predicted sequence tends to trail the actual sequence (i.e. lag).

In Chapter 6, the problem of lag is explored in greater detail. We explain, why a regular time series model can detect a change in the trend only when the historical data has also gone through the same, hence the prediction is always late leading to the problem of lag. Then a Dual Network solution is presented where external features are employed to predict the trend behavior in the first network. The predicted trend is then utilized in the second network to drive the forecast closely with the target sequence in a lag less manner. Thus the Dual Network Solution is

able to provide a better forecast both in terms of low MSE and better movement prediction when compared to other methodologies, which is further evaluated with a novel performance metric: MPM (that we propose).

7.2 Future work

The future research vision should be to explore the field of time series even further, address other fundamental challenges of this field as well as expand the application domain of this thesis. Among the many possible areas of future work that can emerge from this thesis, some of the most prominent ones are the followings,

Self-evolving DNS: As explained in Chapter 6 the one limitation of the Dual Network Solution is that it learns in an offline manner and both of its networks are hand-crafted. Hence in the future, a data-driven self-evolving variant of DNS needs to be developed which can learn from the online time series data (or data streams) and evolve the structures for both of its networks on the go. A self-evolving DNS would solve the challenges of both online time series forecasting and the lag in the predicted sequence. Further, this self-evolving DNS can be adopted into a neuro-fuzzy framework so that it can handle all four challenges of time series forecasting effectively (namely handling of system dynamics, handling of uncertainty, online learning, and problem of lag).

Time series imputation : With real-world supervised machine learning problem missing data can be a significant issue which can lead to different sorts of biases (i.e. selection bias) in the model if not addressed properly. Similarly, time series modeling is also prone to this issue and therefore we need proper imputation techniques for missing values in time series. Usually, when the amount of missing values is low, we can do a mean/ median/ forward value/ backward value imputation. There have been works on imputation techniques based on kNN [120], ARIMA [121, 122], Matrix Factorization [120] for better handling of this problem. Most recent works in this field include Multiple Imputation by Chained Equations (MICE) [123], ImputeTS [124] (R based package), STMVL [125] (spatio-temporal multiview-based learning) and Bidirectional Recurrent Imputation for Time Series (BRITS) [126]. However, there is scope for improvement when the amount of missing data is substantial and not completely at random. We can utilize a framework

built on the basis of LEMON which looks at the bi-directional temporal dependency (from left to right and right to left) of the available data to impute the missing values.

Remaining Useful Life (RUL) prediction : In the manufacturing industry where lots of machinery are involved, wear and tear is a very common phenomenon. Naturally being able to successfully predict the remaining useful life (RUL) of a machine can be very beneficial. There have been many physics model-based methods such as [127–130] for this purpose. However in recent times the data driven methods such as transition probability matrix [131, 132], artificial neural network based models [133–136] have gained more popularity as they require less expert knowledge. However, such data-driven condition based monitoring (CBM) problems often suffer from the challenges of skewed and imbalanced data. Most of the times a machine runs smoothly and whenever it does fail it is stopped or repaired very quickly. So we have very little data on the faulty state of such a machine. This invokes a novel challenge in time series forecasting which needs to be addressed. There are not many works in the area of recurrent neuro-fuzzy systems to address the challenge of RUL prediction. Given their ability to learn the system dynamics well along with their superior uncertainty handling ability, they can be particularly useful in such applications. There need to be further developments on the base framework of NFIS-DN, SPATFIS or BaNFIS and their learning paradigm to address the issue of imbalanced/ skewed data in order to address the RUL prediction problem effectively.

Appendix A

Chapter 3 Additional Materials

A.1 Proof of Lemma 1

The proof is presented here by computing the second derivative of the cost function with respect to the weight vector and showing that it is positive. Differentiating the cost function in equation 3.16 with respect to the weight vector we get,

$$\dot{J}(\mathbf{w}) = \frac{dJ(\mathbf{w})}{d\mathbf{w}} = \sum_k (-y_k \mathbf{F}^T + \mathbf{w}^T \mathbf{F}_k \mathbf{F}_k^T) \quad (\text{A.1})$$

where $\dot{J}(\mathbf{w})$ denotes the first derivative. Taking the second derivative of the same with respect to $\mathbf{w}$ we will get,

$$\begin{aligned} \frac{d^2 J(\mathbf{w})}{d\mathbf{w}^2} &= \frac{d\dot{J}(\mathbf{w})}{d\mathbf{w}} = \sum_k \frac{d(\mathbf{w}^T \mathbf{F}_k \mathbf{F}_k^T)}{d\mathbf{w}} \\ &= \sum_k \mathbf{F}_k \mathbf{F}_k^T \end{aligned} \quad (\text{A.2})$$

Let λ be the eigenvalue and q be the eigenvector for $\mathbf{F}\mathbf{F}^T$. Then $(\mathbf{F}\mathbf{F}^T)q = \lambda q$. Pre-multiplying both sides by q^T we get $q^T(\mathbf{F}\mathbf{F}^T)q = \lambda q^T q$ which means $\lambda = \frac{q^T \mathbf{F}\mathbf{F}^T q}{q^T q} = \frac{z^T z}{q^T q}$, where $z = \mathbf{F}^T q$. It is evident that $q^T q > 0$ and $z^T z \geq 0$ thus making $\lambda \geq 0$ which means the eigenvalues of $\mathbf{F}\mathbf{F}^T$ to be non-negative. It is known that if the eigenvalues of a matrix are non-negative then the matrix is positive semi-definite or positive definite. As the result of the second derivative of the cost function i.e

$\mathbf{F}\mathbf{F}^T$ (as in equation A.2) is a positive definite matrix then we can conclude that the cost function J is convex.

Remark 1: It is proved that the cost function is a convex function and solution of the system of equations in equation 3.19 exists and the same provides a minimal point at $\mathbf{w}^*$.

A.2 Proof of Lemma 2

From equation 3.9 we have the normalized firing strength for r^{th} rule $\bar{F}_r(k) = \frac{F_r(k)}{\sum_{r=1}^R F_r(k)}$ with R number of rules in the rule base. Let's say the p^{th} rule is found to be redundant following the above criteria. Now if we can show that elimination of p^{th} rule does not change the value of $\bar{F}_r(k)$ significantly, then we can conclude that rule elimination does not affect SPATFIS performance.

Let's denote the firing strengths as $\bar{F}_r^R(k)$ and $\bar{F}_r^{R-1}(k)$, before and after pruning the p^{th} rule,

$$\begin{aligned}\bar{F}_r^{R-1}(k) &= \frac{F_r(k)}{\sum_{r=1}^{R-1} F_r(k)} \\ \bar{F}_r^{R-1}(k) &= \frac{F_r(k)}{\sum_{r=1}^R F_r(k) - F_p(k)} \\ \frac{1}{\bar{F}_r^{R-1}(k)} &= \frac{\sum_{r=1}^R F_r(k) - F_p(k)}{F_r(k)} \\ \frac{1}{\bar{F}_r^{R-1}(k)} &= \frac{\sum_{r=1}^R F_r(k)}{F_r(k)} - \frac{F_p(k)}{F_r(k)} \\ \frac{1}{\bar{F}_r^{R-1}(k)} &= \frac{1}{\bar{F}_r^R(k)} - \frac{F_p(k)}{F_r(k)}\end{aligned}\tag{A.3}$$

For simplicity of calculation let's take a single feature (i.e. $\hat{P} = 1$) SPATFIS. From equation 3.8, we have $F_p(k) = \Pi\phi_p(k).\phi_p^m(k)$. From the condition of rule elimination it can be assumed that $\phi_p(k) < 0.011$ and $\phi_p^m(k) < 0.011$. This means, $F_p(k) < 0.1 \times 0.1 = 0.012$. For a redundant rule the $F_p(k)$ value is often much lower than 0.012. Hence the $\frac{F_p(k)}{F_r(k)}$ term in equation A.3 can be neglected,

$$\begin{aligned}\frac{1}{\bar{F}_r^{R-1}(k)} &\approx \frac{1}{\bar{F}_r^R(k)} \\ \bar{F}_r^{R-1}(k) &\approx \bar{F}_r^R(k)\end{aligned}\tag{A.4}$$

Remark 2: It is proved that the elimination of a redundant fuzzy rule does not affect the firing strengths of other rules. For effective rule elimination, N_P is set to 10% of the sample size empirically (i.e. the elimination check will be performed 10 times). In the realm of true data stream environments where the number of training samples is unknown, the evaluation window N_p can be set with respect to the drift rate[137].

A.3 Proof of Lemma 3

When p^{th} and q^{th} rules are found very similar and unified to form the M^{th} rule, the corresponding consequent weights are set in such a manner that, the response from the merged rule will be equal to the cumulative response from the un-merged rules. The same is shown below,

$$w_M(k) = \frac{(\bar{F}_p(k).w_p(k) + \bar{F}_q(k).w_q(k))}{\bar{F}_M(k)} \quad (\text{A.5})$$

In a similar manner $w_M^m(k)$ can be also initialized.

Remark 3: This weight Initialization method for the unified rule, ensures that rule unification is not affecting overall SPATFIS performance.

A.4 Proof of stability during learning

Instead of updating the weights in one shot (summing over all samples and performing a matrix inverse as shown in equation 3.19), we can also learn from each of those samples incrementally with the corresponding instantaneous error, utilizing the following update rule which is derived rearranging the PBL equation A.1 as,

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) - \eta_k \cdot \frac{dJ(\mathbf{w}(k))}{d\mathbf{w}(k)} \\ &= \mathbf{w}(k) - \eta_k \cdot e(k) \cdot \mathbf{F}^T(k) \end{aligned} \quad (\text{A.6})$$

Here η_k is the adaptive learning parameter taking care of the learning pace and is defined by $\eta_k = \eta / (1 + \|\mathbf{F}(k)\|^2)$ where $\|\mathbf{F}(k)\| = \mathbf{F}^T(k) \cdot \mathbf{F}(k)$ and $0 < \eta \leq 1$. We

also define a normalized prediction error (to be utilized for stability proof),

$$e_N(k) = e(k)/(1 + \max_k(\|\mathbf{F}(k)\|^2)) \quad (\text{A.7})$$

If the average normalized prediction error of SPATFIS is upper bounded for infinite number of samples then the system is input output stable. Hence if we can show,

$$\lim_{T \rightarrow \infty} \frac{1}{T} \cdot \sum_{k=1}^T \|e_N(k)\|^2 \leq c \quad (\text{A.8})$$

where c is the aforementioned upper bound, then SPATFIS will be regarded as a stable system. We utilize the class K functions with Lyapunov Input to State Stability theorem for the stability proof.

Proof : A positive scalar $V(k)$ is selected as below.

$$V(k) = \|\mathbf{w}(k)\|^2 \quad (\text{A.9})$$

From the equation above we have the following inequality, $\min(\mathbf{w}^2(k)) \leq V(k) \leq \max(\mathbf{w}^2(k))$ where both $\min(\mathbf{w}^2(k))$, $\max(\mathbf{w}^2(k))$ are Class K_∞ ¹ functions hence satisfies the first condition of the Lyapunov ISS theorem².

---

¹**Definition:**

- Class K : Continuous function $\alpha : [0, a] \rightarrow R^+$ where $\alpha(0) = 0$ and $\alpha(\cdot)$ is strictly increasing then it's a Class K function.
- Class K_∞ : Continuous function $\alpha : R^+ \rightarrow R^+$ where $\alpha(0) = 0$ and $\alpha(\cdot)$ is strictly increasing and $\alpha(r) \rightarrow \infty$ as $r \rightarrow \infty$ then it's a Class K_∞ function.

²**Lyapunov Input to State Stability (ISS) Theorem [138]:** A continuously differentiable function is an ISS Lyapunov function on D if $V : D \rightarrow R$ if $\exists$ Class K_∞ functions $\alpha_i, i = 1, 2, 3$ and class K function α_4 such that,

- $\alpha_1(s) \leq V(s) \leq \alpha_2(s)$ where $s \in D$
- $\Delta V(k) \leq -\alpha_3(\|x(k)\|) + \alpha_4(\|u(k)\|)$

If a non-linear dynamic system (discrete) adopts an ISS Lyapunov function then it can be stated that the system is a input-to-output stable and the behaviour of the system remains bounded when inputs are bounded.

The change in $V(k)$ can be obtained using equation A.6 and $e(k) = y(k) - \hat{y}(k)$ ($y(k)$ being the actual output) as below,

$$\begin{aligned}
\Delta V(k) &= V(k+1) - V(k) \\
&= \|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \\
&= \|\mathbf{w}(k) - \eta_k \cdot e(k) \cdot \mathbf{F}^T(k)\|^2 - \|\mathbf{w}(k)\|^2 \\
&= \|\mathbf{w}(k)\|^2 - 2\eta_k \cdot \|e(k) \cdot \mathbf{F}^T(k) \cdot \mathbf{w}(k)\| \\
&\quad + \eta_k^2 \|e(k) \cdot \mathbf{F}(k)\|^2 - \|\mathbf{w}(k)\|^2 \\
&= \eta_k^2 \|e(k) \cdot \mathbf{F}(k)\|^2 - 2\eta_k \cdot \|e(k) \cdot \mathbf{F}^T(k) \cdot \mathbf{w}(k)\| \\
&= \eta_k^2 \|e(k) \cdot \mathbf{F}(k)\|^2 - 2\eta_k \cdot \|e(k) \cdot \hat{y}(k)\| \\
&= \eta_k^2 \|e(k) \cdot \mathbf{F}(k)\|^2 - 2\eta_k \cdot \|e(k) \cdot [e(k) - y(k)]\|
\end{aligned} \tag{A.10}$$

Using the inequality properties $\|p - q\| \geq \|p\| - \|q\|$ and $2\|pq\| \leq \|p\|^2 + \|q\|^2$ we will get the following inequality from equation A.10 ,

$$\begin{aligned}
\Delta V(k) &\leq \eta_k^2 \|e(k) \cdot \mathbf{F}(k)\|^2 - 2\eta_k \cdot \|e(k)\|^2 \\
&\quad + 2\eta_k \cdot \|e(k) \cdot y(k)\| \\
&\leq \eta_k^2 \|e(k) \cdot \mathbf{F}(k)\|^2 - 2\eta_k \cdot \|e(k)\|^2 \\
&\quad + \eta_k \cdot \|e(k)\|^2 + \eta_k \cdot \|y(k)\|^2 \\
&= -\eta_k \cdot \|e(k)\|^2 (1 - \eta_k \cdot \|\mathbf{F}(k)\|^2) + \eta_k \|y(k)\|^2
\end{aligned} \tag{A.11}$$

Using the adaptive learning parameter, ($\eta_k = \eta / (1 + \|\mathbf{F}(k)\|^2)$) the following inequality can be derived;

$$\begin{aligned}
\eta_k \cdot (1 - \eta_k \cdot \|\mathbf{F}(k)\|^2) &= \eta_k \cdot \left(1 - \frac{\eta}{1 + \|\mathbf{F}(k)\|^2} \cdot \|\mathbf{F}(k)\|^2\right) \\
&\geq \eta_k \cdot \left(1 - \frac{\max_k(\|\mathbf{F}(k)\|^2)}{1 + \max_k(\|\mathbf{F}(k)\|^2)}\right) \\
&= \frac{\eta_k}{1 + \max_k(\|\mathbf{F}(k)\|^2)} \\
&\geq \frac{\eta}{[1 + \max_k(\|\mathbf{F}(k)\|^2)]^2} \\
&\geq \pi
\end{aligned} \tag{A.12}$$

where $\pi = \frac{\eta}{[1+max_k(||\mathbf{F}(k)||^2)]^2}$ and $\pi \cdot ||e(k)||^2 = \eta ||e_N(k)||^2$ (from equation A.8). Hence the inequality in A.11 can be rewritten,

$$\begin{aligned}\Delta V(k) &\leq -\pi ||e(k)||^2 + \eta ||y||^2 \\ &\leq -\eta ||e_N(k)||^2 + \eta \acute{y}\end{aligned}\tag{A.13}$$

where $\acute{y} = max_k(||y(k)||^2)$. $\pi ||e(k)||^2$ is an Class K_∞ function whereas $\eta ||y||^2$ is a Class K function. Hence satisfies the second condition of the theorem. So it can be safely concluded, $V(k)$ adopts an ISS Lyapunov function where the 'Input' and 'State' (i.e. $u(k), x(k)$ from theorem) corresponds to the target output and the error of the system and (i.e. $y(k), e(k)$ of the system). So a bounded actual output will always result in bounded error which is proved by performing the same task for T time steps and then summing up the result of equation A.13,

$$\begin{aligned}V(T) - V(1) &\leq -\sum_{k=1}^T \eta ||e_N(k)||^2 + T\eta \acute{y} \\ \sum_{k=1}^T \eta ||e_N(k)||^2 &\leq V(1) - V(T) + T\eta \acute{y} \\ \frac{1}{T} \sum_{k=1}^T \eta ||e_N(k)||^2 &\leq \frac{1}{T} V(1) + \eta \acute{y}\end{aligned}\tag{A.14}$$

Lastly limit of both side is taken with respect to $T \rightarrow \infty$,

$$\begin{aligned}\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{k=1}^T \eta ||e_N(k)||^2 &\leq \lim_{T \rightarrow \infty} \left(\frac{1}{T} V(1) + \eta \acute{y} \right) \\ \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{k=1}^T \eta ||e_N(k)||^2 &\leq \eta \acute{y} \\ \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{k=1}^T ||e_N(k)||^2 &\leq \acute{y} \\ \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{k=1}^T ||e_N(k)||^2 &\leq c\end{aligned}\tag{A.15}$$

where the upper bound from equation A.8, i.e. $c = \acute{y}$.

Remark 4: It is concluded that SPATFIS follows the proposed stability hypothesis which demonstrates (with equation A.14) that the average normalized prediction error over infinite number of time steps (samples) is upper bounded which was

hypothesized in the first place (equation A.8). Hence SPATFIS is proved to be Lyapunov Input to State stable (or ISS).

A.5 Rank based statistical tests

Six well known neural fuzzy inference systems, i.e. eTS [52], SimpleTS [53], SAFIS [58], McFIS [59], PANFIS [60] and GENEFIS [61] were picked from literature and rebuilt in MATLAB 2016 for the purpose of a fair and unbiased comparison. A non-parametric Friedman's test is conducted followed by a pairwise Bonferroni-Dunn posthoc test [139] to demonstrate the significance of improvement achieved with SPATFIS.

Ranks are assigned to each of the algorithms based on their training time (lowest training time in the tables 3.1-3.3 gets rank 1, then the next best gets 2 and so on). The non-parametric Friedman's F-score value is computed to be 3.33 (details are provided in the supplementary material with more experiment) whereas the F-statistic value $F_{6,30}$ is found from the F-table to be 2.42. As Friedman's score is greater than the F-value at 95 %confidence level, the null hypothesis is rejected and it is concluded that the average ranks of the algorithms employed in this study are statistically different.

Once the Friedman's null hypothesis is rejected, one can highlight the better performance of SPATFIS over any other algorithm using the pairwise Bonferroni-Dunn test. The test statistics z value is computed (refer to supplementary material) for all pairs of methods. Conducting the pairwise tests it is found that SPATFIS is significantly faster than the other algorithms (except for GENEFIS) at a 95% confidence interval and at a 90% confidence interval for GENEFIS.

In a similar manner, both tests are performed on the ranks based on accuracy and rule base size. SPATFIS was found to be better than the other algorithms at 95% confidence interval in both cases.

A.6 Extension of SPATFIS

SPATFIS performs decently in terms of handling uncertainty with its type-1 fuzzy inference engine. However a type-1 fuzzy-engine may sometimes fail to handle the uncertainty well and in such cases it is important to utilize type-2 inference systems. Hence we have proposed a type-2 variant of SPATFIS hereby referred to as, Recurrent Interval Type 2 Fuzzy Inference System or RIT2FIS [90]. RIT2FIS adopts an interval type 2 fuzzy engine for superior handling of uncertainty. The memory neurons employed in its hidden and output layer are similar to SPATFIS. They retain the temporal information, making RIT2FIS highly proficient in tracing system dynamics at a granular level. RIT2FIS also benefits from incorporating a k-Means algorithm inspired approach to cluster the data in an unsupervised manner. An 'Elbow Method' is utilized next to determine the optimal clustering which is then employed as the optimal fuzzy rule base for RIT2FIS eliminating the necessity of expert knowledge for fuzzy initiation. The antecedent and consequent parameters of RIT2FIS are updated using a gradient descent based backpropagation through time algorithm where the learning is made self-regulatory to avoid over-fitting and ensure generalization. Performance of RIT2FIS is evaluated against popular neuro-fuzzy methods on different benchmark and real-world time series problems which distinctly indicates an improved accuracy and a parsimonious rule base.

In a separate extension to SPATFIS, the memory neurons are employed in another neuro-fuzzy framework where the rule growing and pruning criteria are different. This work is termed as Meta-cognitive Recurrent Neuro-Fuzzy Inference System with Memory Neurons (McRFIS-MN) [89] as it adopts the structure learning techniques from McFIS [59]. This model is able to perform adequately in the benchmark problems.

Appendix B

Chapter 4 Additional Materials

B.1 NFIS-DN learning flow-chart

Input: NFIS-DN Learns from the samples $\mathbf{v}(k) = [y(k-1), \mathbf{u}(k)]$.

Output: Different parameters of NFIS-DN architecture such as the weights i.e. w_{1r}, w_{2r}, β , the membership parameters i.e. μ_{jri}, σ_r and the dynamic neuron learning parameters i.e. $\alpha^F, \alpha_r^R, \alpha_i^I$ need to be updated.

Initialize all the aforementioned weights, membership parameters and adaptive recurrence factors with random values (between 0 and 1) and set all the recurrent outputs to 0, and the number of rules empirically.

Algorithm 1: NFIS-DN Training Algorithm

```
while number of training epochs reached do
  while samples are learned (for  $k = 1, 2, \dots, S$ ) do
    Compute NFIS-DN output  $\hat{y}(k)$  (equation 4.20);
    Calculate l2-norm prediction loss  $E$  (equation 4.24);
    if  $e(k)$  is greater than the update threshold then
      the network learns from that sample; all the network parameters
      are updated according to equation 4.27-4.35;
    else
      the sample is not learned from;
    end
  end
  Update the Self-regulated learning threshold with equations 4.23;
end
```

B.2 Additional experiment: Sunspot problem

Sunspots are dark patches on the surface of the sun, first observed in the 1700s after discovery of the telescope. Sunspots are responsible for many solar activities (i.e. change in solar magnetism etc.) yet the exact reasons behind this occurrence is still unknown hence it has been used as a benchmark time series problem for years [1]. The monthly American sunspot data contains 778 samples for the range of 1944 December to 2009 October.¹ The first 699 samples are used for training and the remaining 79 samples are used for testing the network. The goal is to forecast a single step ahead (one month ahead) value of the sunspot number using the sunspot numbers of past two months. As can be seen in table B.1 the Root Mean Square Error (RMSE) is used as performance measure and N was kept at 2.

From table below, it can be observed that, for all the aforementioned standard problems NFIS-DN is able to achieve better accuracy than the other state-of-the-art algorithms while using network of a smaller (or similar) size. It is however slower than SPATFIS.

TABLE B.1: NFIS-DN Performance on sunspot problem (additional experiment)

Problem	Network Model	#Rules	Time(s)	Test RMSE	Rank
Sun-spot	eTS [52]	23	3.5	0.047	2
	SAFIS [58]	21	4.4	0.100	6
	SimpleTS [53]	20	3.2	0.050	3
	McFIS [59]	12	4.2	0.060	4
	SPATFIS [88]	10	0.5	0.070	5
	NFIS-DN	09	2.5	0.035	1

¹<https://www.ngdc.noaa.gov/stp/space-weather/solar-data/solar-indices/sunspot-numbers/american/lists>

B.3 Rank based statistical tests

As mentioned before, four popular neuro-fuzzy inference systems are rebuilt in MATLAB 2016 environment (Windows 10, 16gb RAM) i.e. SAFIS [58], eTS [52], SimpleTS [53] and McFIS [59] for performance evaluation (along with SPATFIS [88]). The rank based Friedman test (non-parametric) is conducted over multiple algorithms (NFIS-DN and the 5 others) and multiple problems (total 7 dataset including sunspot problem) to detect the significant differences in their prediction performance. The pair-wise post-hoc Bonferroni-Dunn test is chosen next to highlight that NFIS-DN has better mean prediction accuracy compared to others. The degree of freedom is $7*5=35$ and well suited for the aforementioned tests.

First, ranks are assigned to each of the algorithms according to their test accuracy (RMSE) from the tables 4.1-4.6 such that algorithm with lowest RMSE gets rank 1, then the next best gets 2 and so on. Then for each of these algorithms mean is computed over their ranks in all 7 dataset. A non-parametric Friedman's test is performed next and for that we compute the F-score value using the average ranks. For further reading on the detailed procedure on this please refer to [139]. We perform the experiment at 95% confidence as per the standard convention. The corresponding critical value is obtained from the F-table using the notation $F_{6,7(6-1),0.05}=F_{6,35,0.05}= 2.79$. The computed Friedman's score is found to be greater than the critical F-value of 2.79 at 95% confidence level. Hence, the null hypothesis can be rejected and we can conclude that the algorithms used in this study perform statistically different.

After rejecting the Friedman's null hypothesis, one can highlight the better performance of NFIS-DN using a pairwise post-hoc Bonferroni-Dunn test. The Bonferroni-Dunn critical difference (CD) is computed to be 1.29 (for detailed procedure please refer to [139]). It can be observed that average rank difference of all the algorithms is greater than this CD value. Hence, it can be concluded that accuracy of NFIS-DN, is significantly better than the other neuro-fuzzy methods with a confidence level of 95%. A similar test on the basis of training time would however suggest, SPATFIS to be significantly faster (at 95% confidence) than all the other algorithms.

Appendix C

Chapter 5 Additional Materials

Algorithm 2: Method of Learning Elastic Memory Online (LEMON)

Input data $[\mathbf{v}(k), y(k)]$ in chunks

Learn: from first couple of chunks with PBL (equation 14-15)

Set: initial memory window size to W_{in}

Set: temporality re-estimation indicator to 1

while *learning from chunks* **do**

if *temporality re-estimation indicator* = 1 **then**

Evolve the memory window W using drift rate (Equation 5.10-5.11)

Estimate the temporal order n using equation 6,12

Modify the memory window as per the temporality measure n

Learn from the chunk using PBL (Equations 3.20-3.21)

else

Utilize past n inputs for prediction in temporal learning

Compute the drift indicator using Equations 5.7-5.8

if *drift is detected* **then**

Set temporality re-estimation indicator = 1

else

Set temporality re-estimation indicator = 0

Learn from the chunk using PBL (Equations 3.20-3.21)

List of Author's Publications¹

Journal Articles

1. Subhrajit Samanta, Mahardhika Pratama, and Suresh Sundaram. "A novel spatio-temporal fuzzy inference system (spatfis) and its stability analysis." *Information Sciences* 505 (2019): 84-99.
2. Subhrajit Samanta, Sundaram Suresh, J. Senthilnath, and N. Sundararajan. "A new neuro-fuzzy inference system with dynamic neurons (nfis-dn) for system identification and time series forecasting." *Applied Soft Computing* 82 (2019): 105567.
3. Subhrajit Samantat, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. "Learning elastic memory online for fast time series forecasting." *Neurocomputing* (2019).
4. Subhrajit Samanta, Mahardhika Pratama, and Suresh Sundaram. "Bayesian Neuro-Fuzzy Inference System (BaNFIS) for Temporal Dependency Estimation." *IEEE Transactions on Fuzzy Systems* (2020).
5. Abhay Aradhya* & Subhrajit Samanta*, Mahardhika Pratama and Suresh Sundaram "Deep Temporal Information Mining Engine (DeepTIME) for exploring regional brain activities in ADHD using rs-fMRI", Under Review in *IEEE Transactions on Cybernetics*(2020)

---

¹The superscript * indicates joint first authors

Conference Proceedings

6. Subhrajit Samanta, Shubhangi Ghosh, and Suresh Sundaram. “A meta-cognitive recurrent fuzzy inference system with memory neurons (mcrfis-mn) and its fast learning algorithm for time series forecasting.” In 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2231-2237. IEEE, 2018.
7. Subhrajit Samanta, Andre Hartanto, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. “RIT2FIS: a recurrent interval type 2 Fuzzy Inference System and its rule base estimation.” In 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1-8. IEEE, 2019.
8. Das, Monidipa, Mahardhika Pratama, Andri Ashfahani, and Subhrajit Samanta. “FERNN: A Fast and Evolving Recurrent Neural Network Model for Streaming Data Classification.” In 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1-8. IEEE, 2019.
9. Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. “A Dual Network Solution (DNS) for lag-free time series forecasting”, 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020.

Bibliography

- [1] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015. [1](#), [14](#), [50](#), [52](#), [71](#), [104](#), [105](#), [123](#), [124](#), [140](#)
- [2] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994. [1](#), [14](#)
- [3] Helmut Lütkepohl. Introduction. In *New Introduction to Multiple Time Series Analysis*, pages 1–7. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. [4](#), [15](#), [16](#)
- [4] Junsoo Lee. Univariate time series modeling and forecasting (box-jenkins method). *Econ 413, lecture 4*. [15](#)
- [5] John H Cochrane. Time series for macroeconomics and finance. *Manuscript, University of Chicago*, 2005. [15](#), [16](#)
- [6] Keith W Hipel and A Ian McLeod. *Time series modelling of water resources and environmental systems*, volume 45. Elsevier, 1994. [4](#), [16](#)
- [7] Jerome H Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67, 1991. [4](#), [17](#)
- [8] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003. [4](#)
- [9] Qiang Song and Brad S Chissom. Fuzzy time series and its models. *Fuzzy Sets and Systems*, 54(3):269–277, 1993. [5](#)
- [10] Ebrahim H Mamdani and Sedrak Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975. [5](#), [27](#)
- [11] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, 1985. [5](#), [27](#)
- [12] Andreas C Neves, Ignacio González, John Leander, and Raid Karoumi. A new approach to damage detection in bridges using machine learning. In *International Conference on Experimental Vibration Analysis for Civil Engineering Structures*, pages 73–84. Springer, 2017. [18](#)

- [13] Çağrı Kaymak and Ayşegül Uçar. A brief survey and an application of semantic image segmentation for autonomous driving. In *Handbook of Deep Learning Applications*, pages 161–200. Springer, 2019. [18](#)
- [14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. [19](#)
- [15] Mark JL Orr et al. Introduction to radial basis function networks, 1996. [21](#), [87](#), [113](#)
- [16] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999. [21](#)
- [17] IJ Leontaritis and Stephen A Billings. Input-output parametric models for non-linear systems part i: deterministic non-linear systems. *International journal of control*, 41(2):303–328, 1985. [22](#)
- [18] Asriel U Levin and Kumpati S Narendra. Control of nonlinear dynamical systems using neural networks. ii. observability, identification, and control. *IEEE trans. on neural networks*, 7(1):30–42, 1996. [22](#)
- [19] C. Svarer, L. K. Hansen, and J. Larsen. On design and evaluation of tapped-delay neural network architectures. In *IEEE International Conference on Neural Networks*, pages 46–51 vol.1, 1993. [22](#)
- [20] Ying-Chung Wang, Chiang-Ju Chien, and Ching-Cheng Teng. Direct adaptive iterative learning control of nonlinear systems using an output-recurrent fuzzy neural network. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 34(3):1348–1359, June 2004.
- [21] Jie Zhang and A. J. Morris. Recurrent neuro-fuzzy networks for nonlinear process modeling. *IEEE Trans. on Neural Networks*, 10(2):313–326, Mar 1999.
- [22] George C Mouzouris and Jerry M Mendel. Dynamic non-singleton fuzzy logic systems for nonlinear modeling. *IEEE Trans. on Fuzzy Systems*, 5(2):199–208, 1997. [22](#)
- [23] Robust recurrent fuzzy neural network control for linear synchronous motor drive system. *Neurocomputing*, 50:365 – 390, 2003. [22](#)
- [24] Narmax time series model prediction: feedforward and recurrent fuzzy neural network approaches. *Fuzzy Sets and Systems*, 150(2):331 – 350, 2005. [22](#)
- [25] P. S. Sastry, G. Santharam, and K. P. Unnikrishnan. Memory neuron networks for identification and control of dynamical systems. *IEEE Trans. on Neural Networks*, 5(2):306–319, Mar 1994. [22](#), [23](#), [33](#), [38](#), [65](#)

- [26] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. [23](#), [35](#)
- [27] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997. [23](#)
- [28] Jeen-Shing Wang and Yen-Ping Chen. A fully automated recurrent neural network for unknown dynamic system identification and control. *IEEE Trans. on Circuits and Systems I: Regular Papers*, 53(6):1363–1372, June 2006. [23](#)
- [29] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. [23](#)
- [30] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. [23](#)
- [31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. [23](#), [104](#), [105](#), [123](#), [124](#)
- [32] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005. [24](#)
- [33] Jan Koutník, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014. [24](#)
- [34] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016. [24](#)
- [35] Hao Hu and Guo-Jun Qi. State-frequency memory recurrent neural networks. In *International Conference on Machine Learning*, pages 1568–1577, 2017. [24](#), [123](#), [124](#)
- [36] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. [24](#)
- [37] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004. [24](#)
- [38] James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010. [24](#)
- [39] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040. Citeseer, 2011. [24](#)

- [40] Roni Mittelman. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015. [24](#)
- [41] Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. Probabilistic recurrent state-space models. *arXiv preprint arXiv:1801.10395*, 2018. [24](#)
- [42] Yi Zhao, Yanyan Shen, Yanmin Zhu, and Junjie Yao. Forecasting wavelet transformed time series with attentive neural networks. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1452–1457. IEEE, 2018. [24](#), [104](#), [105](#), [120](#), [123](#), [124](#)
- [43] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.
- [44] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017. [24](#)
- [45] James J Buckley. Universal fuzzy controllers. *Automatica*, 28(6):1245–1248, 1992. [25](#)
- [46] L-X Wang. Fuzzy systems are universal approximators. In *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*, pages 1163–1170. IEEE, 1992.
- [47] Hao Ying. Sufficient conditions on general fuzzy systems as function approximators. *Automatica*, 30(3):521–525, 1994. [25](#)
- [48] J. S. R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(3):665–685, May 1993. [29](#), [30](#)
- [49] Edwin Lughofer, Carlos Cernuda, Stefan Kindermann, and Mahardhika Pratama. Generalized smart evolving fuzzy systems. *Evolving Systems*, 6(4):269–292, 2015. [29](#)
- [50] Edwin Lughofer, Robert Pollak, Alexandru-Ciprian Zavoianu, Mahardhika Pratama, Pauline Meyer-Heye, Helmut Zörrer, Christian Eitzinger, Julia Haim, and Thomas Radauer. Self-adaptive evolving forecast models with incremental pls space updating for on-line prediction of micro-fluidic chip quality. *Engineering Applications of Artificial Intelligence*, 68:131–151, 2018. [29](#)
- [51] Gang Leng, Thomas Martin McGinnity, and Girijesh Prasad. An approach for on-line extraction of fuzzy rules using a self-organising fuzzy neural network. *Fuzzy sets and systems*, 150(2):211–243, 2005. [29](#)

- [52] P. P. Angelov and D. P. Filev. An approach to online identification of takagi-sugeno fuzzy models. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 34(1):484–498, Feb 2004. [29](#), [43](#), [50](#), [51](#), [52](#), [53](#), [55](#), [70](#), [71](#), [72](#), [73](#), [75](#), [96](#), [97](#), [98](#), [102](#), [103](#), [137](#), [140](#), [141](#)
- [53] Plamen Angelov and Dimitar Filev. SimplLets: a simplified method for learning evolving takagi-sugeno fuzzy models. In *Fuzzy Systems, 2005. FUZZ'05. The 14th IEEE International Conference on*, pages 1068–1073. IEEE, 2005. [29](#), [50](#), [51](#), [52](#), [53](#), [55](#), [70](#), [71](#), [72](#), [73](#), [75](#), [96](#), [97](#), [98](#), [102](#), [103](#), [137](#), [140](#), [141](#)
- [54] M. Prasad, Y.Y. Lin, C.T. Lin, M.J. Er, and O.K. Prasad. A new data-driven neural fuzzy system with collaborative fuzzy clustering mechanism. *Neurocomputing*, 167:558–568, 2015. [29](#)
- [55] Yevgeniy V. Bodyanskiy, Oleksii K. Tyshchenko, and Daria S. Kopaliani. Adaptive learning of an evolving cascade neo-fuzzy system in data stream mining tasks. *Evolving Systems*, 7(2), 2016. [29](#)
- [56] Rong-Jing Bao, Hai-Jun Rong, Plamen P Angelov, Badong Chen, and Pak Kin Wong. Correntropy-based evolving fuzzy neural system. *IEEE Transactions on Fuzzy Systems*, 26(3):1324–1338, 2017. [29](#)
- [57] Y. Bodyanskiy and O. Vynokurova. Hybrid adaptive wavelet-neuro-fuzzy system for chaotic time series identification. *Information Sciences*, 220:170 – 179, 2013. ISSN 0020-0255. Online Fuzzy Machine Learning and Data Mining. [29](#)
- [58] Hai-Jun Rong, N Sundararajan, Guang-Bin Huang, and P Saratchandran. Sequential adaptive fuzzy inference system (safis) for nonlinear system identification and prediction. *Fuzzy sets and systems*, 157(9):1260–1275, 2006. [29](#), [30](#), [43](#), [50](#), [51](#), [52](#), [53](#), [55](#), [67](#), [70](#), [71](#), [72](#), [73](#), [75](#), [96](#), [97](#), [98](#), [102](#), [103](#), [137](#), [140](#), [141](#)
- [59] K Subramanian and Sundaram Suresh. A meta-cognitive sequential learning algorithm for neuro-fuzzy inference system. *Applied soft computing*, 12(11):3603–3614, 2012. [29](#), [30](#), [43](#), [50](#), [51](#), [52](#), [53](#), [55](#), [68](#), [70](#), [71](#), [72](#), [73](#), [75](#), [96](#), [97](#), [98](#), [102](#), [103](#), [137](#), [138](#), [140](#), [141](#)
- [60] Mahardhika Pratama, Sreenatha G Anavatti, Plamen P Angelov, and Edwin Lughofer. Panfis: A novel incremental learning machine. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):55–68, 2014. [30](#), [43](#), [50](#), [51](#), [52](#), [53](#), [55](#), [97](#), [98](#), [102](#), [103](#), [137](#)
- [61] Mahardhika Pratama, Sreenatha G Anavatti, and Edwin Lughofer. Genefis: Toward an effective localist network. *IEEE Trans. Fuzzy Systems*, 22(3):547–562, 2014. [30](#), [50](#), [51](#), [52](#), [53](#), [55](#), [97](#), [98](#), [102](#), [103](#), [137](#)
- [62] Hai-Jun Rong, N Sundararajan, Guang-Bin Huang, and Guang-She Zhao. Extended sequential adaptive fuzzy inference system for classification problems. *Evolving Systems*, 2(2):71–82, 2011. [30](#)

- [63] José de Jesús Rubio and Abdelhamid Bouchachia. Msafis: an evolving fuzzy inference system. *Soft Computing*, 21(9):2357–2366, 2017. [30](#)
- [64] José de Jesús Rubio. Error convergence analysis of the sufin and csufin. *Applied Soft Computing*, 72:587–595, 2018. [30](#)
- [65] Md Meftahul Ferdaus, Mahardhika Pratama, Sreenatha Anavatti, and Matthew A Garratt. Palm: An incremental construction of hyperplanes for data stream regression. *IEEE Transactions on Fuzzy Systems*, 2019. [30](#)
- [66] P. A. Mastorocostas and J. B. Theocharis. A recurrent fuzzy-neural model for dynamic system identification. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 32(2):176–190, Apr 2002. [30](#), [31](#)
- [67] Ching-Hung Lee and Ching-Cheng Teng. Identification and control of dynamic systems using recurrent fuzzy neural networks. *IEEE Trans. on Fuzzy Systems*, 8(4):349–366, Aug 2000. [31](#)
- [68] Cheng-Jian Lin and Cheng-Chung Chin. Prediction and identification using wavelet-based recurrent fuzzy neural networks. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 34(5):2144–2154, Oct 2004. [31](#)
- [69] Qi-Zhi Zhang, Woon-Seng Gan, and Ya li Zhou. Adaptive recurrent fuzzy neural networks for active noise control. *Journal of Sound and Vibration*, 296(4):935 – 948, 2006. [31](#)
- [70] Y. Y. Lin, J. Y. Chang, and C. T. Lin. Identification and prediction of dynamic systems using an interactively recurrent self-evolving fuzzy neural network. *IEEE Trans. on Neural Networks and Learning Systems*, 24(2):310–321, Feb 2013. [32](#)
- [71] Generalized predictive control using recurrent fuzzy neural networks for industrial processes. *Journal of Process Control*, 17(1):83 – 92, 2007. [32](#)
- [72] Chih-Min Lin and Chun-Fei Hsu. Supervisory recurrent fuzzy neural network control of wing rock for slender delta wings. *IEEE Trans. on Fuzzy Systems*, 12(5):733–742, Oct 2004. [32](#)
- [73] Faa-Jeng Lin and Rong-Jong Wai. Hybrid control using recurrent fuzzy neural network for linear induction motor servo drive. *IEEE Trans. on Fuzzy Systems*, 9(1):102–115, Feb 2001. [32](#)
- [74] A recurrent self-evolving fuzzy neural network with local feedbacks and its application to dynamic system processing. *Fuzzy Sets and Systems*, 161(19):2552 – 2568, 2010.
- [75] C. F. Juang and C. D. Hsieh. A locally recurrent fuzzy neural network with support vector regression for dynamic-system modeling. *IEEE Trans. on Fuzzy Systems*, 18(2):261–273, April 2010. [32](#)

- [76] Robust friction state observer and recurrent fuzzy neural network design for dynamic friction compensation with backstepping control. *Mechatronics*, 20(3):384 – 401, 2010. [32](#)
- [77] Rafik A Aliev, BG Guirimov, Bijan Fazlollahi, and Rashad R Aliev. Evolutionary algorithm-based learning of fuzzy neural networks. part 2: Recurrent fuzzy neural networks. *Fuzzy Sets and Systems*, 160(17):2553–2566, 2009.
- [78] Chia-Feng Juang and Chin-Teng Lin. A recurrent self-organizing neural fuzzy inference network. *IEEE Trans. on Neural Networks*, 10(4):828–845, Jul 1999. [32](#), [33](#)
- [79] Chia-Feng Juang. A tsk-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms. *IEEE Trans. on Fuzzy Systems*, 10(2):155–170, Apr 2002. [30](#), [33](#), [67](#), [68](#)
- [80] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986. [33](#)
- [81] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992. [33](#), [34](#)
- [82] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. [35](#)
- [83] Kenji Doya. Recurrent networks: Supervised learning. In *The handbook of brain theory and neural networks*, pages 796–800. MIT Press, 1998. [35](#)
- [84] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. [35](#)
- [85] Sharad Singhal and Lance Wu. Training multilayer perceptrons with the extended kalman algorithm. In *Advances in neural information processing systems*, pages 133–140, 1989. [35](#)
- [86] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010*, pages 177–186. Springer, 2010. [35](#), [85](#)
- [87] G Sateesh Babu and Sundaram Suresh. Meta-cognitive rbf network and its projection based learning algorithm for classification problems. *Applied Soft Computing*, 13(1):654–666, 2013. [35](#), [82](#), [85](#), [114](#)
- [88] Subhrajit Samanta, Mahardhika Pratama, and Suresh Sundaram. A novel spatio-temporal fuzzy inference system (spatfis) and its stability analysis. *Information Sciences*, 505:84–99, 2019. [37](#), [70](#), [71](#), [72](#), [73](#), [75](#), [97](#), [98](#), [102](#), [103](#), [140](#), [141](#)
- [89] Subhrajit Samanta, Shubhangi Ghosh, and Suresh Sundaram. A meta-cognitive recurrent fuzzy inference system with memory neurons (mcrfis-mn) and its fast learning algorithm for time series forecasting. In *2018 IEEE*

- Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018. [37](#), [138](#)
- [90] Subhrajit Samanta, Andre Hartanto, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. Rit2fis: A recurrent interval type 2 fuzzy inference system and its rule base estimation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019. [37](#), [138](#)
  - [91] Nikola K Kasabov and Qun Song. Denfis: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *IEEE transactions on Fuzzy Systems*, 10(2):144–154, 2002. [43](#)
  - [92] Arthur Albert. *Regression and the Moore-Penrose pseudoinverse*. Elsevier, 1972. [45](#)
  - [93] Edwin Lughofer, Jean-Luc Bouchot, and Ammar Shaker. On-line elimination of local redundancies in evolving fuzzy systems. *Evolving Systems*, 2(3):165–187, Sep 2011. [49](#)
  - [94] R Scott Crowder. Predicting the mackey-glass time series with cascade-correlation learning. In *Proc. 1990 Connectionist Models Summer School*, pages 117–123. Carnegie Mellon Univ., Pittsburgh, PA, 1990. [50](#), [51](#), [67](#), [71](#)
  - [95] AK Das, Nguyen Anh, Sundaram Suresh, and N Srikanth. An interval type-2 fuzzy inference system and its meta-cognitive learning algorithm. *Evolving Systems*, 7(2):95–105, 2016. [53](#), [117](#)
  - [96] Ankit Kumar Das, Kartick Subramanian, and Suresh Sundaram. An evolving interval type-2 neurofuzzy inference system and its metacognitive sequential learning algorithm. *IEEE Trans. on Fuzzy Systems*, 23(6):2080–2093, 2015. [53](#), [64](#)
  - [97] S Samanta, S Suresh, J Senthilnath, and N Sundararajan. A new neuro-fuzzy inference system with dynamic neurons (nfis-dn) for system identification and time series forecasting. *Applied Soft Computing*, 82:105567, 2019. ISSN 1568-4946. [57](#), [97](#), [98](#), [102](#), [103](#)
  - [98] S Suresh, SN Omkar, V Mani, and TN Guru Prakash. Lift coefficient prediction at high angle of attack using recurrent neural network. *Aerospace Science and Technology*, 7(8):595–602, 2003. [60](#), [61](#)
  - [99] Jae S Lim. Two-dimensional signal and image processing. *Englewood Cliffs, NJ, Prentice Hall, 1990, 710 p.*, 1990. [60](#)
  - [100] Sundaram Suresh, Keming Dong, and HJ Kim. A sequential learning algorithm for self-adaptive resource allocation network classifier. *Neurocomputing*, 73(16):3012–3019, 2010. [64](#)

- [101] A. K. Das, K. Subramanian, and S. Suresh. A computationally fast interval type-2 neuro-fuzzy inference system and its meta-cognitive projection based learning algorithm. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 1510–1516, July 2014. [64](#)
- [102] Kumpati S Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks*, 1(1):4–27, 1990. [67](#), [68](#), [70](#), [87](#)
- [103] Ilhami Colak, Seref Sagiroglu, and Mehmet Yesilbudak. Data mining and wind power prediction: A literature review. *Renewable Energy*, 46:241–247, 2012. [68](#)
- [104] K Subramanian, Ramaswamy Savitha, and Sundaram Suresh. A complex-valued neuro-fuzzy inference system and its learning mechanism. *Neurocomputing*, 123:110–120, 2014. [68](#)
- [105] E Michael Azoff. *Neural network time series forecasting of financial markets*. John Wiley & Sons, Inc., 1994. [68](#)
- [106] Sau Wai Tung, Chai Quek, and Cuntai Guan. et2fis: An evolving type-2 neural fuzzy inference system. *Information Sciences*, 220:124–148, 2013. [68](#)
- [107] José de Jesús Rubio. Sofmls: online self-organizing fuzzy modified least-squares network. *IEEE Trans. on Fuzzy Systems*, 17(6):1296–1309, 2009. [68](#)
- [108] Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. Learning elastic memory online for fast time series forecasting. *Neurocomputing*, 390:315–326, 2020. [77](#), [123](#), [124](#)
- [109] Samanta Subhrajit, Pratama Mahardhika, and Suresh Sundaram. Bayesian neuro-fuzzy inference system (banfis) for temporal dependency estimation. *IEEE Transactions on Fuzzy Systems*, 2020. [77](#)
- [110] Helmut Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005. [78](#)
- [111] Geoffrey I Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. Analyzing concept drift and shift from sample data. *Data Mining and Knowledge Discovery*, 32(5):1179–1199, 2018. [85](#)
- [112] Chia-Feng Juang. A tsk-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 10(2):155–170, 2002. [87](#)
- [113] Liheng Zhang, Charu Aggarwal, and Guo-Jun Qi. Stock price prediction via discovering multi-frequency trading patterns. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2141–2149, 2017. [103](#), [120](#)

- [114] Li-Juan Cao and Francis Eng Hock Tay. Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Transactions on neural networks*, 14(6):1506–1518, 2003. [104](#), [123](#)
- [115] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. [104](#), [105](#), [123](#), [124](#)
- [116] Lin Wang, Yi Zeng, and Tao Chen. Back propagation neural network with adaptive differential evolution algorithm for time series forecasting. *Expert Systems with Applications*, 42(2):855–863, 2015. [104](#), [105](#), [123](#), [124](#)
- [117] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. [104](#), [105](#), [123](#), [124](#)
- [118] Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. A dual network solution (dns) for lag-free time series forecasting. [107](#)
- [119] Tao Lin, Tian Guo, and Karl Aberer. Hybrid neural networks for learning the trend in time series. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, number CONF, pages 2273–2279, 2017. [123](#), [124](#)
- [120] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001. [129](#)
- [121] David S Fung. Methods for the estimation of missing values in time series. 2006. [129](#)
- [122] Andrew C Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990. [129](#)
- [123] Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 20(1):40–49, 2011. [129](#)
- [124] Steffen Moritz and Thomas Bartz-Beielstein. imputets: time series missing value imputation in r. *R J.*, 9(1):207, 2017. [129](#)
- [125] Xiuwen Yi, Yu Zheng, Junbo Zhang, and Tianrui Li. St-mvl: filling missing values in geo-sensory time series data. 2016. [129](#)
- [126] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. In *Advances in Neural Information Processing Systems*, pages 6775–6785, 2018. [129](#)

- [127] Daniel J Inman, Charles R Farrar, Vicente Lopes Junior, and Valder Steffen Junior. *Damage prognosis: for aerospace, civil and mechanical systems*. John Wiley & Sons, 2005. [130](#)
- [128] Sean Marble and Brogan P Morton. Predicting the remaining life of propulsion system bearings. In *2006 IEEE Aerospace Conference*, pages 8–pp. IEEE, 2006.
- [129] Gregory J Kacprzynski, Michael J Roemer, Girish Modgil, Andrea Palladino, and Kenneth Maynard. Enhancement of physics-of-failure prognostic models with system level features. In *Proceedings, IEEE Aerospace Conference*, volume 6, pages 6–6. IEEE, 2002.
- [130] C James Li and Hyungdae Lee. Gear fatigue crack prognosis using embedded model, gear dynamic model and fracture mechanics. *Mechanical systems and signal processing*, 19(4):836–846, 2005. [130](#)
- [131] Andrew KS Jardine, Daming Lin, and Dragan Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical systems and signal processing*, 20(7):1483–1510, 2006. [130](#)
- [132] D Banjevic, AKS Jardine, V Makis, and M Ennis. A control-limit policy and software for condition-based maintenance optimization. *INFOR: Information Systems and Operational Research*, 39(1):32–50, 2001. [130](#)
- [133] Aiwina Heng, Sheng Zhang, Andy CC Tan, and Joseph Mathew. Rotating machinery prognostics: State of the art, challenges and opportunities. *Mechanical systems and signal processing*, 23(3):724–739, 2009. [130](#)
- [134] Runqing Huang, Lifeng Xi, Xinglin Li, C Richard Liu, Hai Qiu, and Jay Lee. Residual life predictions for ball bearings based on self-organizing map and back propagation neural network methods. *Mechanical systems and signal processing*, 21(1):193–207, 2007.
- [135] PW Tse and DP Atherton. Prediction of machine deterioration using vibration based fault trends and recurrent neural networks. 1999.
- [136] Zhigang Tian and Ming J Zuo. Health condition prediction of gears using a recurrent neural network approach. *IEEE transactions on reliability*, 59(4):700–705, 2010. [130](#)
- [137] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295. Springer, 2004. [133](#)
- [138] Eduardo D Sontag and Yuan Wang. On characterizations of the input-to-state stability property. *Systems & Control Letters*, 1995. [134](#)
- [139] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006. [137](#), [141](#)