A Study on Refined Neural Network Approach with Data Transformation for Software Fault Prediction

(ソフトウェアフォールト予測のためのデータ変換を用いたニューラルネットワークアプローチに関する考察)

Dissertation submitted in partial fulfillment for the
degree of Doctor of Engineering

Momotaz Begum

Under the supervision of
Professor Tadashi Dohi

Dependable Systems Laboratory,
Department of Information Engineering,
Graduate School of Engineering,
Hiroshima University, Higashi-Hiroshima, Japan

September 2017

Abstract

Software fault prediction is one of the most fundamental techniques in software fault management, and is used to assess the software product reliability and to control the development processes. Software reliability is an important facet of software quality, and is defined as the probability of failure-free software operation for a specified period of time in a specified environment. For the purpose of quantitative assessment, software reliability growth models (SRGMs) have been widely used during the last four decades. Since SRGMs are essentially stochastic models with abstractions, they must be built under several simplified mathematical assumptions, and, at the same time, their parameter estimation with the fault count data observed in software testing is not a trivial task. Because the maximum likelihood estimation, which is commonly used, is reduced to a multi-modal nonlinear optimization problem with constraints, and requires the much computation effort. Apart from SRGMs based on stochastic modeling, artificial neural network (ANN) has gained much popularity to deal with non-linear phenomena arising in applications to time series forecasting, pattern recognition, function approximation, etc. Comparing with non-trivial stochastic models, it is easy to implement the ANN for the software fault prediction, since the feed-forward back-propagation (BP) type of learning algorithm can be widely used to estimate the internal parameters, such as connection weights.

In this thesis, we concern the software fault prediction using a multilayer-perceptron neural network, where the underlying software fault count data is transformed to the Gaussian data, by means of the well-known five data transformation methods. More specially, we mainly consider the long-term prediction of the number of software faults, and propose a refined neural network approach with the grouped data, where the multi-stage look-ahead prediction is carried out with a simple multilayer perceptron neural network with multiple outputs. In details, we discuss two different research topics; one-stage look-ahead prediction and multi-stage look-ahead prediction.

In Chapter 2, we focus on a prediction problem with the common multilayer perceptron neural network of the cumulative number of software faults in sequential software testing. We apply the well-known back propagation algorithm

for feed forward neural network architectures. We also discuss not only the point estimation but also the sensitivity of the neural network architecture on input neuron and hidden neuron by applying the well-known " rules of thumb " techniques. It is revealed that the rule of thumb is rather accurate to obtain the nearly optimal network architecture for real failure data analyses.

In Chapter 3 and Chapter 4, we pay our attention to the software fault prediction and prediction interval for long term. Here we study the long-term prediction of the number of software faults, and propose a refined neural network approach with the grouped data, where the multi-stage look-ahead prediction is done with a simple multi-layer perceptron neural network with multiple outputs. Under the assumption that the software fault count data follows a Poisson process with an unknown mean value function, we transform the underlying Poisson count data to the Gaussian data via five data transformation methods. Next, we predict the long-term behavior and derive the predictive interval of the cumulative number of software faults in sequential software testing by the refined neural network. On the other hand, nonhomogeneous Poisson process (NHPP)-based SRGMs are widely used for modeling the detection of software faults in software testing. By comparing our proposed models with conventional ones, we show the utility of our neural network models. It is exposed which method is an appropriate one in the both viewpoints of point estimation and interval estimation, throughout our simulation experiments and real failure data analyses.

In Chapter 5, we concern the optimal software release time which minimizes the relevant expected software cost via a refined neural network approach with the grouped data, where the multi-stage look-ahead prediction is performed with a simple three-layer perceptron neural network with multiple outputs. To our best knowledge, there have no research result on the optimal software release problems for long-term prediction via a refined neural network approach. To analyze the software fault count data which follows a Poisson process with unknown mean value function, we transform the underlying Poisson count data to the Gaussian data by means of one of five data transformation methods, and predict the cost-optimal software release time via a neural network. We compare our neural network approach with the common NHPP -based SRGMs. Finally, we conclude the thesis with some remarks in Chapter 6.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Professor Tadashi Dohi, the supervisor of my study, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Professor Takio Kurita, and Associate Professor Hiroyuki Okamura, for their insightful comments, encouragements and checking the manuscript, but also for the hard question, which incented me to widen my research from various perspectives. And I also wish to thank the MEXT (Ministry of Education, Culture, Sports, Science, and Technology) Japan Government Scholarship for financial support during three years.

Finally, it is my special pleasure to acknowledge the hospitality and encouragement of the past and present members of the Dependable Systems Laboratory, Department of Information Engineering, Hiroshima University. Last but not the least, I would like to thank my family: my husband, my parents and to my brothers and sister for supporting me spiritually throughout writing this thesis and my life in general.

Contents

Abstract **iv**

Acknowledgements **ix**

1 Introduction **1**

1.1 Software Fault Prediction 1

1.2 Software Reliability Assessment 2

1.3 Artificial Neural Network (ANN) 4

1.4 Data Transformation 7

1.5 Predictive Interval (PI) 8

1.6 Optimal Software Release Problem (SRP) 9

1.7 Organization of Dissertation 11

2 Point Estimation of Cumulative Number of Software Faults : One stage look-ahead prediction **15**

2.1 Neural Network Approach 15

2.1.1 Preliminary Set-up 17

2.1.2 Neural Network Architecture 17

2.2 Numerical Examples 20

2.2.1 Data Sets 20

2.2.2 Predictive Performance 20

2.3 Effect of number of hidden neurons and input neurons 22

2.4 Numerical Illustrations 24

2.4.1 Setup 24

2.4.2 Effect of number of hidden nodes 25

2.4.3 Effect of number of input nodes 26

2.4.4 One-stage look ahead point prediction 30

3 Multi-stage look-ahead prediction : Refined Neural Network Architecture 31

3.1 Model Description 31

3.1.1 NHPP-based Software Reliability Modeling 31

3.1.2 A Refined Neural Network Approach 33

3.2 Numerical Experiments 38

3.2.1 Discussion 39

3.3 Simulation Experiments 49

3.3.1 Point Estimation 50

4 Predictive Interval: Refined Neural Network Approach 55

4.1 Delta Method 55

4.2 One-stage look ahead prediction 57

4.3 PI for multi-stage look-ahead prediction 63

4.4 Numerical Experiments 64

4.4.1 Interval Estimation 64

4.4.2 Statistical Properties of Estimators 72

5 Optimal Software Release Decision : Refined NN Approach 85

5.1 Point Estimation 85

5.1.1 NHPP-Based Software Reliability Modeling 85

5.1.2 Refined NN approach 87

5.2 Optimal Software Testing Policy 92

5.3 Numerical Illustrations 95

5.3.1 Predictive Performance 95

5.3.2 Results 95

6 Conclusions 107

6.1 Summary and Remarks 107

6.2 Future Works 109

Appendix A AE result for four DS1 $\sim$ DS4 data sets 111

A.1 DS1 113

A.2 DS2 113
A.3 DS3 113
A.4 DS4 113

Appendix B AE result for four DS5 $\sim$ DS8 data sets **129**
B.1 DS5 130
B.2 DS6 130
B.3 DS7 130
B.4 DS8 130

Bibliography **147**

Publication List of the Author **155**

List of Figures

2.1 Architecture of feed forward MLP neural network. 20

2.2 Time-dependent behavior of relative error 22

2.3 Overall experimental process. 24

2.4 s+1 testing day predicated value (DS4 22:15:1). 29

2.5 Time-dependent behavior of relative error (DS7 10:7:1). 29

3.1 Architecture of back propagation type MIMO. 34

3.2 Configuration of prediction scheme via MIMO. 34

3.3 Time-dependent behavior of relative error with DS1 at 60% observation point. 41

3.4 Time-dependent behavior of relative error with DS6 at 60% observation point. 41

3.5 Determination of the transformation parameter λ (DS1 with 50% observation point). 48

3.6 Determination of the transformation parameter λ (DS2 with 50% observation point). 49

3.7 Case1 Fitted dataset 52

3.8 Case4 S-shaped dataset 52

3.9 Time-dependent behavior of relative error with Case1 ($k = 40$). 54

4.1 Prediction Interval 60

4.2 Predictive Interval with optimal hidden and input neurons. . . . 63

4.3 A algorithm for simulation based predictive interval method. . . 65

4.4 Prediction Interval for Delta method with DS1. 70

4.5 Predictive Interval for Simulation based method with DS1. . . . 71

4.6 Prediction Interval for delta method with Case1. 74

4.7 Prediction Interval for Simulation based method with Case1. . . 75
4.8 Prediction Interval for delta method with Case2. 76
4.9 Prediction Interval for simulation based method with Case2. . . . 77
4.10 Prediction Interval for delta method with Case3. 78
4.11 Prediction Interval for simulation based method with Case3. . . . 79
4.12 Prediction Interval for delta method with Case4. 80
4.13 Prediction Interval for simulation based method with Case4. . . . 81
4.14 Real predictive interval with Case1 $\sim$ Case4 82

5.1 Architecture of back propagation type MIMO. 93
5.2 Configuration of prediction scheme via MIMO. 94
5.3 DS1. 97
5.4 DS2. 98

List of Tables

2.1 Data transform formulae. 22

2.2 Average Relative error in point prediction. 23

2.3 Data sets observed in study. 25

2.4 Average Relative error in point prediction. 27

2.5 Average Relative error in point prediction. 28

3.1 NHPP-based SRGMs. 42

3.2 Data transform formulae. 43

3.3 Data sets observed for multi-stage look ahead prediction. 43

3.4 AE according to observation point for four (DS1 $\sim$ DS4) datasets 44

3.5 AE according to observation point for four (DS5 $\sim$ DS8) datasets 45

3.6 AE according to prediction length for four (DS1 $\sim$ DS4) datasets 46

3.7 AE according to prediction length for four (DS5 $\sim$ DS8) datasets 47

3.8 Comparison table for both cases with all datasets 48

3.9 Comparison of average relative errors for fifteen days prediction with Case1 $\sim$ Case4 ($l = 15$). 53

4.1 Prediction PI measures. 59

4.2 Prediction PI measures. 62

4.3 DS1$\sim$ DS4. 68

4.4 DS5$\sim$ DS8. 69

4.5 Case1$\sim$ Case4. 73

4.6 Real Predictive Interval. 74

4.7 Statistics of estimators of the software cumulative number of fault data. 83

5.1 Posteriori optimal software testing time for eight datasets. 92
5.2 Prediction results of optimal software testing time with DS1 with $c_2 = 4$. 98
5.3 Prediction results of optimal software testing time for DS1 with $c_2 = 9$. 99
5.4 Prediction of optimal software release time for DS2 with $c_2 = 4$. 99
5.5 Prediction of optimal software release time for DS2 with $c_2 = 9$. 100
5.6 Prediction results of optimal software testing time with DS3 with $c_2 = 4$. 100
5.7 Prediction results of optimal software testing time with DS3 with $c_2 = 9$. 101
5.8 Prediction results of optimal software testing time with DS4 with $c_2 = 4$. 101
5.9 Prediction results of optimal software testing time with DS4 with $c_2 = 9$. 102
5.10 Prediction results of optimal software testing time with DS5 with $c_2 = 4$. 102
5.11 Prediction results of optimal software testing time with DS5 with $c_2 = 9$. 103
5.12 Prediction results of optimal software testing time with DS6 with $c_2 = 4$. 103
5.13 Prediction results of optimal software testing time with DS6 with $c_2 = 9$. 104
5.14 Prediction results of optimal software testing time with DS7 with $c_2 = 4$. 104
5.15 Prediction results of optimal software testing time with DS7 with $c_2 = 9$. 105
5.16 Prediction results of optimal software testing time with DS8 with $c_2 = 4$. 105
5.17 Prediction results of optimal software testing time with DS8 with $c_2 = 9$. 106

A.1 Comparison of average relative errors for five days prediction with DS1 ($l = 5$). 114

A.2 Comparison of average relative errors for ten days prediction with DS1 ($l = 10$). 115

A.3 Comparison of average relative errors for fifteen days prediction with DS1 ($l = 15$). 116

A.4 Comparison of average relative errors for twenty days prediction with DS1 ($l = 20$). 117

A.5 Comparison of average relative errors for five days prediction with DS2 ($l = 5$). 118

A.6 Comparison of average relative errors for ten days prediction with DS2 ($l = 10$). 119

A.7 Comparison of average relative errors for fifteen days prediction with DS2 ($l = 15$). 120

A.8 Comparison of average relative errors for twenty days prediction with DS2 ($l = 20$). 120

A.9 Comparison of average relative errors for five days prediction with DS3 ($l = 5$). 121

A.10 Comparison of average relative errors for five days prediction with DS3 ($l = 10$). 122

A.11 Comparison of average relative errors for five days prediction with DS3 ($l = 15$). 123

A.12 Comparison of average relative errors for five days prediction with DS3 ($l = 20$). 123

A.13 Comparison of average relative errors for five days prediction with DS4 ($l = 5$). 124

A.14 Comparison of average relative errors for five days prediction with DS4 ($l = 10$). 125

A.15 Comparison of average relative errors for five days prediction with DS4 ($l = 15$). 126

A.16 Comparison of average relative errors for five days prediction with DS4 ($l = 20$). 127

B.1 Comparison of average relative errors for five days prediction with DS5 ($l = 5$). 131

B.2 Comparison of average relative errors for five days prediction with DS5 ($l = 10$). 132
B.3 Comparison of average relative errors for five days prediction with DS5 ($l = 15$). 133
B.4 Comparison of average relative errors for five days prediction with DS5 ($l = 20$). 134
B.5 Comparison of average relative errors for five days prediction with DS6 ($l = 5$). 135
B.6 Comparison of average relative errors for five days prediction with DS6 ($l = 10$). 136
B.7 Comparison of average relative errors for five days prediction with DS6 ($l = 15$). 137
B.8 Comparison of average relative errors for five days prediction with DS6 ($l = 20$). 138
B.9 Comparison of average relative errors for five days prediction with DS7 ($l = 5$). 139
B.10 Comparison of average relative errors for five days prediction with DS7 ($l = 10$). 140
B.11 Comparison of average relative errors for five days prediction with DS7 ($l = 15$). 141
B.12 Comparison of average relative errors for five days prediction with DS7 ($l = 20$). 142
B.13 Comparison of average relative errors for five days prediction with DS8 ($l = 5$). 143
B.14 Comparison of average relative errors for five days prediction with DS8 ($l = 10$). 144
B.15 Comparison of average relative errors for five days prediction with DS8 ($l = 15$). 145
B.16 Comparison of average relative errors for five days prediction with DS8 ($l = 20$). 146

Acronyms and abbreviations

SRM	Software reliability model
AIC	Akaike information criterion
AT1	Anscombe square-root transform
AT2	Asymptotically unbiased transformation
BP	Back propagation
BT	Bartlett transform
EM	Expectation and maximization
FT	Fisz transform
MIMO	Multiple input multiple output
MLP	Multi-layer perceptron
NHPP	Nonhomogeneous Poisson process
SRATS	Software reliability assessment tool with spread sheet
SRGM	Software reliability growth model
SSE	Sum of squared errors

Chapter 1

Introduction

Our modern society depends on computer systems, so that there are numerous different applications, such as nuclear reactors, financial trading systems and medical systems, etc. Therefore, failure may occur even when software is used. Nowadays, predicting software faults is an applicable issue and a major anxiety for software developers and engineers. In a software development project, most of software engineers set goals to provide high quality software.

1.1 Software Fault Prediction

Software fault prediction is one of the most fundamental techniques in software fault management, and is used to measure the software product reliability and to control the development processes. Software reliability is an important facet of software quality, and is described as the probability of failure-free software operation for a specified period of time in a specified environment. For the purpose of quantitative assessment, SRGMs have been extensively used during the last four decades. Since SRGMs are essentially stochastic models with abstractions, they must be built under several simplified mathematical assumptions, and, at the same time, their parameter estimation of the fault count data observed in software testing is not a trivial task. Because the maximum likelihood estimation, which is commonly used, is reduced to a multi-modal nonlinear optimization problem with constraints, and requires the much computation effort. Apart from SRGMs based on stochastic modeling, ANN has gained much popularity to deal with non-linear phenomena arising in applications to time series forecasting, pattern recognition, function approximation, etc. Comparing with non-trivial stochastic models, it is easy to implement the ANN for the software fault prediction, since the feed-forward back-propagation (BP) type of learning algorithm can be widely used to estimate the internal parameters, such as connection weights. Cai *et al.* [7] examined that handling datasets with ‘smooth’ trends is more effectiveness in the neural network approach than handling datasets with large fluctuations. They concluded that the neural network approach is much better in prediction than SRGMs. Sherer [67] predicted software faults in several NASA projects with neural networks [70], [71]. It should be pointed out that the neural network-based approach has some drawbacks for application in software reliability assessment. The major problem is the design of neural network architecture, involving the number of input neurons in each layer and the number of hidden layers, so that both of them must be determined carefully through trial-and-error heuristics. Caruana [69] showed generalization results on a variety of problems as the size of the networks varies. Second, in related works on neural network application to predict the number of software faults, the prediction is always deterministic. In other words, the point prediction of the number of software faults detected at each testing day is given as an output of complex non-linear functions with trained parameters. In this case, it can be anticipated that the accuracy of future point forecast significantly reduces.

In this thesis, we concern the software fault prediction using a multilayer perceptron (MLP) neural network (NN), where the underlying software fault count data is transformed to the Gaussian data, by means of the well-known data transformation methods. More specially, we mainly consider the long-term prediction of the number of software faults, and propose a refined neural network approach with the grouped data, where the multi-stage look-ahead prediction is carried out with a simple MLP NN with multiple outputs. In details, we discuss two different research areas; one-stage look-ahead prediction and multi-stage look ahead prediction.

1.2 Software Reliability Assessment

In our modern society, computer systems become more essential and complicated. Especially, critical software applications increase in size and complexity day by day. Software reliability is a still challenging issue because almost all computer-based systems are controlled by software. Especially, quantification of software reliability is quite important from the standpoint of product liability. Since the quantitative software reliability is defined as the probability that software failures caused by software faults do not occur for a given period of time, SRGM has been extensively studied in both software engineering and reliability engineering community. In fact, during the last four decades, a large number of SRGMs have been proposed in the literature, and some of them have been used to assess software reliability and to control quantitatively software testing [29],[34]. Since SRGMs are stochastic models with abstractions, they must be built under several simplified mathematical assumptions, and, at the same time, their parameter estimation with the fault count data observed in software testing is not a straightforward task. Because the maximum likelihood estimation, which is commonly used, is reduced to a multi-model nonlinear optimization problem with constraints, and it requires a large amount of computation. Another problem with SRGMs is the model selection from a great number of SRGM candidates. During the last four decades, over three hundred SRGMs have been proposed in the literature. Among them, SRGMs based on NHPPs have been extensively used for describing the stochastic behavior of the number of detected faults, from their tractability and goodness-of-fit performance. Achcar *et al.* [3], Goel and Okumoto [14], Goel [15], Gokhale and Trivedi [16], Ohba [38], Ohishi *et al.* [39], Okamura *et al.* [40], Yamada *et al.* [55], Zhao and Xie [58], among others, are well-known as representative NHPP-based SRGMs. These NHPP-based SRGMs can be classified into parametric models, where the mean value function or the intensity function characterizing NHPP-based SRGMs is known in advance. More precisely, since the parametric SRGMs depend on the statistical property of fault-detection time, the choice of NHPP-based SRGMs is equivalent to choosing the fault-detection time distribution. However, the lesson learned from a number of empirical researches reported during the last four decades suggests that the best parametric SRGM does not exist, which can fit every type of software fault data. This fact means that the best parametric NHPP-based SRGM has to be selected carefully from many candidates by checking their goodness-of-fit performance in each software development project. The conclusion from the empirical research suggests that the best SRGM in terms of the goodness-of-fit performance depends on the kind of software fault count data. In other words, there does not exist in the best SRGM which can fit every software fault count data. Sharma *et al.* [49] proposed a selection method of the best SRGM by using the concept of *distance*. However, it appears to be a questionable method and does not motivate us to use the maximum likelihood estimation. In addition, it is worth mentioning that the best SRGM with the highest goodness-of-fit to the past observation does not always possess the best predictive performance to the future (unknown) software fault detection pattern.

On the other hand, there are two classes of SRGMs; analytical SRGMs and data-driven artificial neural network models. However, none of these models satisfy the requirement levels of software developers [50]. Furthermore, prediction of software faults is an important measurement to find reliable software in the software operational phase. Sometimes professional managers use any predictor for software quality after release. Abaei *et al.* [2] explained software fault prediction based on different machine learning techniques such as decision trees, decision tables, random forest, neural network, nave Bayes and distinctive artificial immune systems classifiers. They made a conclusion that random forest outperform the other methods. In this thesis, we consider NHPP-based SRGMs to predict point estimation for long term prediction.

1.3 Artificial Neural Network (ANN)

ANN is a computational metaphor inspired by the brain and nervous system study, and consists of an input layer with some inputs, multiple hidden layers with hidden neurons and one output layer. The input layer of neurons can be used to capture the inputs from the outside world. Since the hidden layer of neurons has no communication with the external world, the output layer of neurons sends the final output to the external world. Hence, determining an appropriate number of hidden neurons is an important design issue in neural computation.

Apart from the stochastic modeling, ANN has gained much popularity to deal with non-linear phenomena arising in applications to time series forecasting, pattern recognition, function approximation, *etc.* Comparing with non-trivial stochastic models, it is easy to implement the ANN for the software fault prediction, since the feed-forward back-propagation (BP) type of learning algorithm is widely used to estimate the internal parameters, such as connection weights. Software fault prediction using the ANN was proposed first by Karunanithi *et al.* [21], Karunanithi and Malaiya [22],[23]. They applied simple MLP feed forward neural networks, which enjoy a universal approximation ability [6] to represent an arbitrary nonlinear mapping with any degree of accuracy, in the prediction of software fault-detection time in software testing. Since their seminal contributions, the ANN approach has been frequently applied to different estimation /prediction problems in software engineering. Khoshgoftaa *et al.* [24],[26], Khoshgoftaar and Szabo [25] considered the problems to identify fault-prone modules in software quality assessment. Site [48] compared some ANN approaches with a set of SRGMs from the viewpoint of prediction. Pai and Hong [42], Tian and Noore [51],[52], Su and Huang [50], Noekhah *et al.* [37], Mahajan *et al.* [33] utilized some evolutionary computing techniques and machine learning techniques to improve the predictive performance for the classical ANN models. Hu *et al.* [17] applied the ANN approach to predict both software fault detection and correction processes simultaneously.

On the other hand, it should be pointed out that the neural network-based approach has some drawbacks for application in software reliability assessment. First, the design of neural network architecture, involving the number of neurons in each layer and the number of hidden layers, is arbitrary so that it must be determined carefully through try-and-error heuristics. Second, the above references with neural application to predict the number of software faults are based on the deterministic output. In other words, the point prediction of the number of software faults detected at the next testing day is given as an output of complex non-linear functions with trained parameters. Hence, it may be clear that the quantitative software reliability as a probability cannot be obtained in this framework. The basic but implicit assumption in reliability engineering is that the number of faults is given by a non-negative integer-valued random variable, which is deeply related to the software reliability. Apart from the practical requirement in software engineering, it is often pointed out that the unsatisfactorily low prediction performance arises when neural networks provide ambiguity on the data. In this case, the accuracy of future point forecasts significantly drops. More specifically, when the training data in neural networks is sparse, the point prediction in neural computing may be less reliable. Dissimilar to the familiar SRGMs, it is impossible to quantify the software reliability as a probability by applying the common ANN approach. This feature penalizes us to use the ANN when quantifying the software reliability measures such as the software reliability, mean time to software failure, *etc.* On one hand, through the ANN is a simple connectionist model depending on the architecture, it can be considered as a statistically nonparametric model without specific model assumptions. As mentioned above, a huge number of SRGMs have been developed in the literature [1], [3], [14], [15], [16], [28], [38], [39], [40], [55], [58], but almost all of them are based on some parametric assumptions whose cannot been validated for every fault count data. In that sense, the ANN approach can be viewed as one of nonparametric models with no specific model assumptions. In fact, the ANN approach provides a data-driven modeling framework and can bridge several kinds of machine learning techniques. Yang *et al.* [57] applied a model mining technique to provide a generic data-driven SRGM. Xiao and Dohi [54] proposed a nonparametric wavelet method to estimate NHPP-based SRGM. Park *et al.* [61] also compared several data-driven approaches with the existing SRGMs.

BP is a common method of training ANN. The BP algorithm is the well-known gradient descent method to update the connection weights, to minimize the squared error between the network output values and the teaching signals. There are two special inputs; bias units which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. We refine the existing ANN approach as a data-driven model from the view point of software fault prediction. In particular, we deal with the grouped data which consists of the number of software fault counts detected at each testing date, although the existing ANN approaches focus on only the software fault-detection time data which are not easily available in actual software testing. It is worth noting that the ANN approaches in early works [17], [21], [22], [23], [24], [25], [26], [33], [37], [48], [50], [51], [52] just considered the one-stage look-ahead prediction, and did not consider the long-term prediction of the cumulative number of software faults detected in future. It is a surprising fact because such a problem does not occur in the common SRGMs [46].

1.4 Data Transformation

It is common to input real value data in the MLP neural network. Since our problem is to predict the number of software faults newly detected at the next testing day, however, the underlying data should be integer. In general, it is convenient to treat a real number in almost all neural network computing, and to apply the useful property of the Gauss distribution for constructing prediction intervals approximately (e.g. see [27]). Hence we suppose that the software fault count is described by the Poisson law [29], [34], [43], [53]. In the existing literature, some of the authors concern the prediction of the software fault-detection time and handle the real number in their neural network calculations. Such a pre-data processing is common in the wavelet shrinkage estimation [54], but has not been considered in the software fault prediction via the ANN. According to the idea by Xiao and Dohi [54], we apply five data transform techniques from the Poisson data to the Gaussian data. More specifically, we use Bartlett transform (BT) [5], Anscombe transform (AT1) [4], Fisz transform (FT) [12], Asymptotically unbiased transformation (AT2) [32] and BoxCox power transformation [8] as the most major normalizing and variance-stabilizing transforms.

The AT1 is widely used to pre-process the Poisson data before processing the Gaussian data. Taking the AT1, the cumulative number of software fault data can be approximately transformed to the Gaussian data. The AT1 is a natural extension of the well-known BT, which is known as the most fundamental data transform tool in statistics. The FT is characterized by square root transformation as an extension of BT. As the forward Anscombe transformation is nonlinear, it generally leads to biased estimates if one uses either its direct algebraic inverse or the asymptotically unbiased inverse (AT2). An experimental analysis using a few state-of-the-art-denoising algorithms which in a very efficient filtering solution that is competitive with some of the best existing methods for Poisson image denoising [32]. Recently [59] AT2 used for Wavelet shrinkage estimation (WSE) for estimating NHPP-based SRM. To reduce the bias of the wavelet estimator used the algebraic unbiased inverse transformation for software reliability assessment. The numerical study with real software fault count data, they show the effectiveness of the WSE combined with the unbiased inverse transformation. The main objective of Box and Cox [8] considered two approaches. The first approach is used by the Maximum Likelihood method (MLE). It ' s conceptually easy and the profile likelihood function is easy to compute in this case. Also it ' s easy to obtain an approximate confidence interval (CI) for because of the asymptotic property of MLE. The second approach outlined is to use a Bayesian method. We need to first ensure that the model is fully identifiable. Problem is that it has an unknown parameter λ. Their suggestion was to " fix one, or possibly a small number, of λs and go ahead with the detailed estimation " . In their examples, they used what ' s usually called " snap to the grid " methods to choose the estimate of λ.

1.5 Predictive Interval (PI)

Reliability of a software system can be adversely affected by the number of residual faults present in the system. The main goal of software developers is to minimize the number of faults in the delivered code. A predictive interval(PI) is an estimate of an interval in which future observations will fall, with a certain probability, given what has already been observed. PIs are often used in regression analysis and future observation. In the traditional statistics, the interval estimation is useful to take account of the uncertainty of point estimate itself, though it is difficult to obtain analytically the statistical estimator distribution of the target output. Hwang and Ding [18] and Veaux *et al.* [60] give the fundamental theory and the learning algorithms to obtain predictive intervals for neural networks, with help of nonlinear regression models. In general, there are three conventional methods to calculate predictive intervals in nonlinear regression problems; delta method, Bayesian method and bootstrap method. MacKay [35]considers a Bayesian predictive interval with application to classification problems. Nix and Weigend [36] develop a probabilistic neural network and propose an approximate method to obtain the output probability distribution by means of the mean and variance of the output. On an excellent survey on predictive intervals with neural networks, see Khosravi *et al.* [27]. In this thesis, we obtain predictive intervals of the cumulative number of software faults detected at each testing day using the multilayer perceptron (MLP) neural network, where the simplest three layers MLP is assumed with the well-known back-propagation algorithm. To predict the number of software faults, we impose a plausible assumption that the underlying fault-detection process obeys the Poisson law with unknown parameters. In numerical experiments with eight actual software development project data sets and four simulation experiment data sets, we evaluate the one-stage and multi-stage look-ahead prediction interval in sequential software testing, and compare these data transformation methods in terms of coverage rate and prediction interval width. Since our method is based on a combination of Monte Carol and normal BP in spite of BPs existing nature, we show that our methods are useful to calculate the predictive intervals on the on-going progress of software debugging process.

1.6 Optimal Software Release Problem (SRP)

The optimal software testing policy which is equivalent to the optimal software release policy has been also considered by many authors. The problem to determine the optimal timing to stop software testing is called the *optimal software release problem (SRP)*. In fact, there is a well-recognized tradeoff relationship in software costs. If the length of software test is much shorter, then the total testing cost can be reduced but the larger debugging cost after releasing software may occur in the operational phase, because the debugging cost in operational phase is much more expensive than that in the testing phase. Conversely, the longer testing period may result in higher software reliability, but leads to increase the testing cost. Thus, it is important to find an appropriate software release time taking account of the expected total software cost. Okumoto and Goel [14] derived the optimal software release time such that the software reliability attains a certain requirement level, for their NHPP-based SRGM. Alternative way introduced in [14] is to stop software testing so as to minimize the expected total software cost, by taking account of the tradeoff relationship. Since the seminal work, many authors consider a number of cost-based software release problems under different model assumptions and optimization criteria.

A number of optimization problems have been formulated under different modeling assumptions (see [10], [31], [44], [45], [72]). More specifically, they formulate the optimal software testing problems under the assumption that the software fault count process in both testing and operational phases follow a completely known stochastic point process such as NHPP-based SRGM. On the other hand, it is known that there does not exist the best SRGM to fit every software fault count data, the formulation of the optimal software testing problems strongly depend on the exact statistical estimation of software fault count process. In fact, non-parametric inference of software fault count process under the incomplete knowledge on the fault counting phenomenon is another issue in SRGM research. They propose an interesting graphical method to find directly an optimal software testing time which minimizes an expected software cost. However, they just treat software fault-detection time data as well, and fail to make the multi-stage look-ahead prediction in long-term. Kaneishi and Dohi [62], Saito and Dohi [47] and Xiao and Dohi [54] develop novel statistical estimation approaches for software fault count process in testing. However, these methods fail to predict the long-term behavior of software fault count process and are not applicable to the optimal software testing problems. Recently, Saito *et al.* [63] propose a non-parametric method to predict approximately the optimal software testing time by minimizing the upper or lower bound of the expected software cost. Dohi *et al.* [11] consider an optimal software testing problem by means of the MLP-based software fault prediction, and estimates the optimal software testing time.

We present a novel method to estimate the optimal software release time which minimizes the relevant expected software cost via a refined neural network approach with the grouped data. The MLP-based software fault prediction has also considered as an alternative non-parametric approach of software fault count. Since neural networks with BP learning algorithm are conventional to deal with not only time series analysis but also pattern recognition and classification (see Blum and Li, [6]). There is a significant challenge on how to predict the software release time minimizing the relevant expected cost. Especially, we have refined a neural network approach considered in Dohi *et al.* [11] by introducing the MIMO type MLP. Since the computation technique on artificial neural network has been improved during the last three decades, it will be possible to apply more sophisticated neural computation technique to predict the optimal software testing time more accurately.

1.7 Organization of Dissertation

This thesis is organized as follows:

In Chapter 2, we focus on a prediction problem with the cumulative number of software faults detected at each testing day using the MLP neural network, where the simplest three layers MLP is assumed with the well-known back-propagation algorithm. To predict the number of software faults, we impose a plausible assumption that the underlying fault-detection process obeys the Poisson law with unknown parameters. Since it is appropriate to input the training data as real number in the conventional MLP neural network, we propose to apply three data transformation methods from the Poisson count data to the Gaussian data: BT [5], AT1 [4], and FT [12]. One of the major problems facing in software fault prediction is the selection of input and hidden neurons number for the architecture of neural networks. To keep relatively small errors on software fault prediction, it is very important to train the neural network. The minimal error reflects better stability, and higher error reflects worst stability. When the neural network is trained, the error of training set becomes small for training data. On the other hand, when new data is available to the neural network, the error may be extremely large. Therefore, we concentrate on this issue and try to determine the optimal number of hidden neurons and input neuron through experiments.

In numerical experiments with eight real software development project data sets, we evaluate the one-stage look-ahead point prediction in sequential software testing, and compare these data transformation methods in terms of the average relative error.

In Chapter 3, we refine the existing ANN approach as a data-driven model from the view point of software fault prediction. In particular, we deal with the grouped data which consists of the number of software fault counts detected at each testing date, although the existing ANN approaches focus on only the software fault-detection time data which are not easily available in actual software testing. We apply five data transformation techniques [4], [5], [12], [8] and [32] from the Poisson law to the Gaussian law, and translate the underlying software fault prediction problem to a nonlinear Gaussian regression problem with the ANN. Next, we make the long-term prediction of the software fault count by creating the multiple output. This idea comes from Cheng *et al.* [9] [61]. It is worth noting that the ANN approaches in early works [17], [21], [22], [23], [24], [26], [25], [33], [37], [48], [50], [51], [52] just considered the one-stage look ahead prediction, and did not consider the long-term prediction of the cumulative number of software faults detected in future. It is a surprising fact because such a problem does not occur in the common SRGMs [46]. We combine the above two ideas in the ANN approach and predict the number of software faults in the sense of long run. In numerical examples with eight actual software fault count data sets, we compare our refined ANN methodology with eleven NHPP-based SRGMs [41] in terms of predictive performance with the average relative error.

In Chapter 4, software reliability in one of the most important attributes in software quality metrics. To predict the number of software faults detected in testing phase, a number of approaches have been applied during the last four decades. Among them, the neural network approach plays a significant role to estimate and predict the number of software fault counts. In here, we derive the predictive interval of the cumulative number of software faults in sequential software testing. We apply the well-known approximation based method delta [27] method and simulation based predictive interval method to construct the predictive interval. Throughout examples with real software fault data and simulation experiment, it is shown that our ANN approach affords a more appropriate prediction device and tends to have an enhanced performance from the viewpoint of predictability.

In Chapter 5, we consider a software release decision to stop the software testing by minimizing the expected total software cost. We present a novel method to estimate the optimal software release time which minimizes the relevant expected software cost via a refined neural network approach with the grouped data. The MLP-based software fault prediction has also considered as an alternative nonparametric approach of the software fault count. An idea on multiple input multiple output (MIMO) by Park et al. [61] and propose a refined neural network approach to predict the long-term behavior of the software fault count with the grouped data. They impose a plausible assumption that the underlying fault count process obeys the Poisson law with an unknown mean value function, and propose to utilize three data transform methods from the Poisson count data to the Gaussian data; [4], [5], [12], [8] and [32]. In numerical examples with eight actual software fault count data, we compare our neural network approach with the common NHPP-based SRGMs. It is shown that our proposed method could provide a more accurate and more flexible decision making than the common stochastic modeling approach.

Finally, the thesis is concluded with some remarks and future directions in Chapter 6.

Chapter 2

Point Estimation of Cumulative Number of Software Faults : One stage look-ahead prediction

In this chapter, we consider on a prediction problem with the common MLP neural network of the cumulative number of software faults in sequential software testing. We apply the well-known BP algorithm for feed forward neural network architectures. To predict the number of software faults, we impose a plausible assumption that the underlying fault-detection process obeys the Poisson law with unknown parameters. Since it is appropriate to input the training data as real number in the conventional MLP neural network, we propose to apply three data transformation methods from the Poisson count data to the Gaussian data: BT[5], AT1 [4] and FT [12] of the cumulative number of software faults in one-stage look-ahead prediction. To keep relatively small errors on software fault prediction, it is very important to train the neural network. We experiment with different numbers of hidden neurons and number of input neurons for all datasets.

2.1 Neural Network Approach

Notation;

x_i**:** the cumulative number of software faults detected at i

$\tilde{x}_i$: any data transformation method at $i(i = 1,\ 2,\ \ldots,\ n)$, be the input for the neural network

s: number of output neuron

k: one hidden layer with k $(= 1, 2, \ldots)$ hidden neurons

$\tilde{x}_s$: output for the neural network

w_{ij}: the connection weight from i-th input neuron to j-th hidden neuron

n: the number of software fault count data

w_{0j}: the bias weights for j-th hidden neuron

w'_{0s}: the bias weights for s-th output neuron

h_j: weighted sum of j-th hidden neuron at $j = 0, 1, \ldots, k$

$f(h_j)$: sigmoid function for respective hidden neuron

$\tilde{x}_s$: summative and weighted inputs from respective hidden neuron in the output layer

w_{js}: the weight is connected from j-th hidden neuron to s-th output neuron.

$f(\tilde{x}_s$: sigmoid function for s-th output neuron

N: the prediction period (integer value)

$\tilde{x}^o_s$: the teaching signal

α: the momentum

η: the learning rate

δh_j: the output gradient of j-th hidden neuron

$\delta \tilde{x}_s$: the output gradient of s-th output neuron

$\Lambda(t)$: expected cumulative number of failures occurred (equivalently the cumulative number of minimal repairs) by time t

$\lambda(t)$: failure intensity function of component

$X^*_{k,i}$: the original failure time data for $i = 1,\ 2,\ \ldots,\ n$, at k-th simulation

$\hat{\tau}^*_{(k)}$: the software fault during time τ^*

2.1.1 Preliminary Set-up

In the common neural computation, it is noted that the ANN including the simplest MLP with only one output neuron is regarded as a nonlinear regression model, where the explanatory variables are randomized by the Gaussian white noise. In other words, the output data in the MLP is implicitly assumed to be a realization of a nonlinear Gaussian model. On the other hand, since one handles the Poisson count data as integer values in the software fault prediction, the underlying data shall be transformed to the Gaussian data in advance. Such a pre-data processing is common in the wavelet shrinkage estimation [54], but has not been considered in the software fault prediction via the ANN. According to the idea by Xiao and Dohi [54], we apply three data transform techniques from the Poisson data to the Gaussian data. More specifically, we use BT [5], AT1 [4], FT [12] as the most major normalizing and variance-stabilizing transforms. Table 2.1 summarizes the data transform techniques and their inverse transform formulae. In the table, x_i denotes the cumulative number of software faults detected at i ($i = 1,\ 2, \ldots, n$)-th testing day. Then, we have the transformed data $\tilde{x}_i$ by means of any data transformation method. Let $\tilde{x}_i$ ($i = 1, 2, \ldots, n$) and $\tilde{x}_s$ ($s = 1$) be the input and output of the neural network, respectively. Then, the prediction of the cumulative number of software faults are given by the inversion of the data transform. Figure 2.1 depicts the architecture of back propagation type MIMO, where n is the number of software fault count data and s is the prediction length. We suppose that there is only one hidden layer with k ($= 1, 2, \ldots$) hidden neurons in our neural network.

2.1.2 Neural Network Architecture

ANNs are widely used for functional approximation and statistical inference. The term of “ artificial neural network” usually refers to mathematical model employed in neural network computing and artificial intelligence. NNs have the learning ability based on the training data or initial experiences, similar to the human brain. Although the neural network in the human brain is composed of a large number of highly interconnected processing elements (neurons) working in parallel, much simpler structure with input layer, hidden layer and output layer is assumed for the common MLP feed forward artificial neural network. It consists of an input layer with some inputs, hidden layer with hidden neurons and one output layer. The input layer of neuron can be used to capture the inputs from the outside world. The hidden layer of neurons has no communication with the external world, but the output layer of neuron sends the final output to the external world. The main function of hidden layer neurons is to receive the inputs and weights from the previous layer and to transfer the aggregated information to the output layer by any transfer function. This output can act as an input of the output layer. The input layer neurons have no computational task. It just receives inputs and associated weights, and passes them to the next layer. For the value coming out of an input neuron, $\tilde{x}_i$ $(i = 1,\ 2,\ \ldots,\ n)$, it is common to add two special inputs; bias units which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. Let $w_{ij} \in [-1, 1]$ be the connection weight from i-th input neuron to j-th hidden neuron, where w_{0j} and w'_{0s} denote the bias weights for j-th hidden neuron and s-th output neuron, respectively for the training phase with $i = 0, 1, \ldots, n$, $j = 0, 1, \ldots, k$ and $s = 1$. Each hidden neuron calculates the weighted sum of the hidden neuron, h_j, in the following equation:

$$h_j = \sum_{i=1}^{n} \tilde{x}_{ij} w_{ij} + w_{0j}. \tag{2.1}$$

$$\tilde{x}_s = \sum_{j=1}^{k} f(h_j) w'_{js} + w'_{0s}. \tag{2.2}$$

Because $\tilde{x}_s$ are also summative and weighted inputs from respective hidden neuron in the output layer, the weight w'_{js} is connected from j-th hidden neuron to s-th output neuron. The output value of the network in the training phase, $\tilde{x}_s$, is calculated by $f(\tilde{x}_s) = 1/\exp(-\tilde{x}_s)$. In the BP algorithm, the error is propagated from an output layer to a successive hidden layer by updating the weights, where the error function is defined by

$$\mathrm{SSE} = \frac{\sum_1^N (\tilde{x}_s^o - \tilde{x}_s)^2}{(N-1)} \tag{2.3}$$

with the prediction value $\tilde{x}_s$, the teaching signal $\tilde{x}_s^o$ and N is the prediction period (integer value). Next we quickly overview the BP algorithm. It updates the weight parameters so as to minimize SSE between the network output values $\tilde{x}_s$ $(s = 1)$ and the teaching signals $\tilde{x}_s^o$, where each connection weight is adjusted using the gradient descent according to the contribution to SSE in Eq.(2.3). The momentum, α, and the learning rate, η, are controlled to adjust the weights and the convergence speed in the BP algorithm, respectively. Since these are the most important tuning parameters in the BP algorithm, we carefully examine these parameters in pre-experiments. Here we set $\alpha = 0.25 \sim 0.90$ and $\eta = 0.001 \sim 0.500$. Then, the connection weights are updated in the following:

$$\begin{aligned} w_{ij(new)} &= w_{ij} + \alpha w_{ij} + \eta \delta h_j \tilde{x}_i \\ &\qquad (i = 1, 2, \ldots, n, j = 1, \ldots, k), \qquad (2.4) \\ w'_{js(new)} &= w'_{js} + \alpha w'_{js} + \eta \delta \tilde{x}_s \tilde{x}_s \\ &\qquad (j = 1, 2, \ldots, k, s = 1), \qquad (2.5) \end{aligned}$$

where δh_j and $\delta \tilde{x}_s$ are the output gradient of j-th hidden neuron and the output gradient in the output layer, and are defined by

$$\begin{aligned} \delta h_j &= f(h_j)(1 - f(h_j)), \qquad (2.6) \\ \delta \tilde{x}_s &= \tilde{x}_s(1 - \tilde{x}_s)(\tilde{x}_s^o - \tilde{x}_s), \qquad (2.7) \end{aligned}$$

respectively. Also, the updated bias weights for hidden and output neurons are respectively given by

$$\begin{aligned} w_{0j(new)} &= w_{0j} + \alpha w_{0j} + \eta \delta h_j, \qquad (2.8) \\ w'_{0s(new)} &= w'_{0s} + \alpha w'_{0s} + \eta \delta \tilde{x}_s. \qquad (2.9) \end{aligned}$$

The above procedure is repeated until the desired output is achieved.

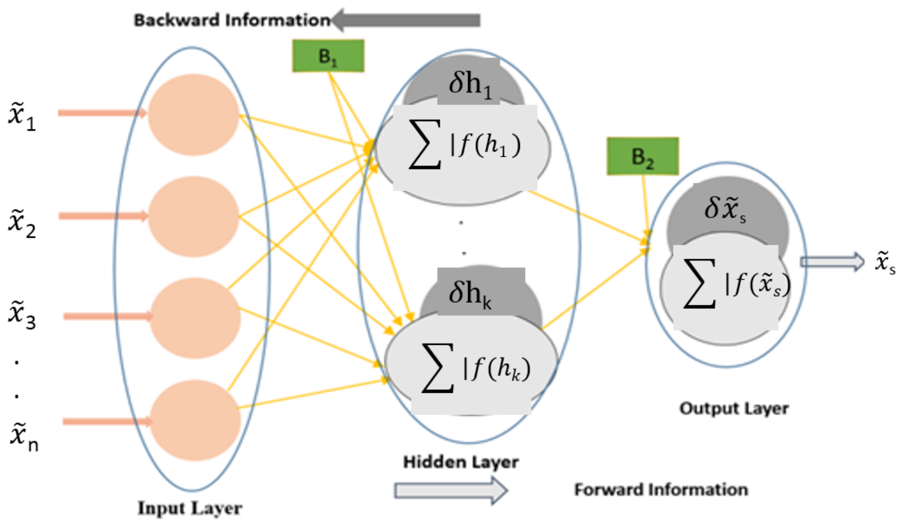


Figure 2.1: Architecture of feed forward MLP neural network. .

2.2 Numerical Examples

2.2.1 Data Sets

We use four real project data sets cited in the reference [29]; DS1∼DS4, which consist of the software fault count (grouped) data. In these data sets, the length of software testing and the total number of detected software faults are given by (62, 133), (22, 54), (41, 351) and (114, 188) respectively. To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent, where the initial guess of weights, w_{ij}, w'_{js}, w_{0j} and w'_{0s}, are given by the uniform random varieties ranged from -1 to +1, the number of total iterations in the BP algorithm run is 1,000 and the convergence criteria on the minimum error is 0.001.

2.2.2 Predictive Performance

The predictive performance is evaluated with the software fault count data through the sequential testing, so that we make the one-stage look ahead prediction based on the past observation. More specifically, we use all the software fault count data experienced before the prediction point as the input data and predict the next one fault count (number of software faults detected at the next testing day). For this purpose, our MLP neural network architecture is somewhat different from [19], [20], [21], [22], [23], [24], [25], [26] because the number of input neurons increases one by one as the observation point goes on. In the first step, the fault count data detected at each testing day is transformed to the Gaussian data by BT, AT1 and FT. Next, we input the transformed data into the MLP neural network, where the momentum, learning rate and initial weights are given carefully. By means of the back propagation, we update all the weights and obtain one output which is regarded as a predicted number of software fault counts at the next testing day.

Suppose that the observation point is given by the n-th testing day. In this case, n software fault counts data are used for training the MIMO neural network. The capability of the prediction model is measured by the average relative error (AE),

$$AE_N = \frac{\sum_{s=1}^{N} RE_s}{N}, \tag{2.10}$$

where RE_s is called the relative error for the next prediction period N and is given by

$$RE_s = \left| \frac{(\tilde{x}_s^o - \tilde{x}_s)}{\tilde{x}_s^o} \right| \quad (s = 1, 2, \ldots, N). \tag{2.11}$$

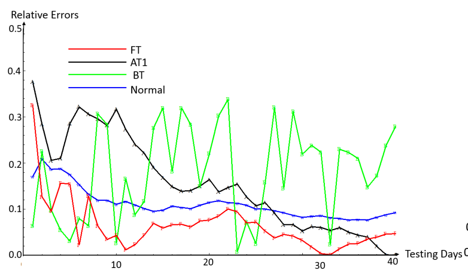
So we regard the prediction model with smaller AE as a better prediction model.

Table 2.2 presents the predictive performance based on AE for four datasets with and without data transformation, where the values in round brackets denote the number of input layer neurons and the number of hidden layer neurons. From this table, the data transformation does not lead to the better predictive performance, because AE is independent of the Gaussian distribution property.

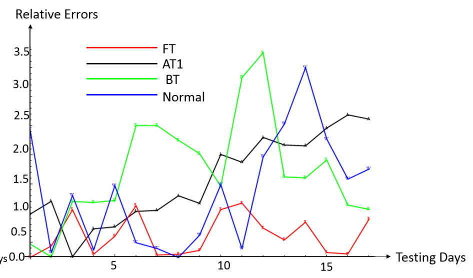
In Figure 2.2, we illustrate the time-dependent behavior of RE and compare data transformation methods with DS1. From this result, it can be observed that the most stable method is the FT and the method without transformation (Normal) does not always give better results in the middle and later testing phases. In comparison of three transformation methods, the most classical Bartlett transform works poorly and cannot give the decreasing trend as the software testing goes on.

Table 2.1: Data transform formulae.

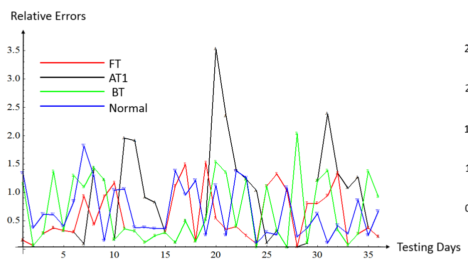
	Formulae	
Method	Data transform	Inversion transform
AT1 [4]	$\tilde{x}_i = 2\sqrt{x_i + 3/8}$	$\frac{\tilde{x}_s^2 - 3/2}{4}$
BT [5]	$\tilde{x}_i = 2\sqrt{x_i + 1/2}$	$\frac{\tilde{x}_s^2 - 2}{4}$
FT [12]	$\tilde{x}_i = \sqrt{x_i + 1} + \sqrt{x_i}$	$\frac{\tilde{x}_s^2 + \tilde{x}_s^{-2} - 2}{4}$



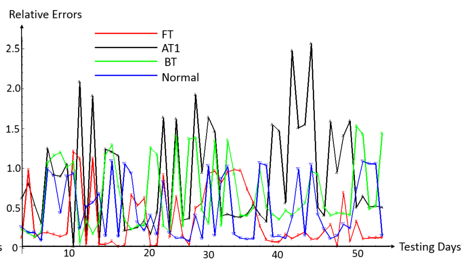
(a) DS1



(b) DS2



(c) DS3



(d) DS4

Figure 2.2: Time-dependent behavior of relative error

2.3 Effect of number of hidden neurons and input neurons

Artificial neural networks consist of several neurons. These neurons receive information from neurons of the previous layer and pass them on to the next layer. Neurons are affiliated with each other by edges. The intensity of the connection between two neurons is represented by a weight value and this weight value stores the knowledge of the neural network. There are a number of factors that affect the performance of neural network such as the number of neurons in the hidden layer, the number of hidden layer, learning rate etc. We have studied

Table 2.2: Average Relative error in point prediction.

Dataset	Average Relative Error	
	Data transformation	AE
DS1(10,8)	Normal	0.3161
	AT1	0.6154
	FT	0.4837
	BT	0.7213
DS2(5,7)	Normal	0.6555
	AT1	1.1302
	FT	1.1925
	BT	1.7521
DS3(5,9)	Normal	0.4749
	AT1	1.9412
	FT	1.2615
	BT	1.4662
DS4(15,10)	Normal	0.7387
	AT1	1.7872
	FT	0.6247
	BT	1.2671

the effect of the number of neurons in the hidden layer and input neurons in the input layer. The overall experimental process is described in Figure 2.3 In the first step, we consider two cases on preprocessing the dataset; one is data transforming from the Poisson data to the Gaussian data with BT, AT1 and FT (see Table 2.1), another without transform. The data with/without transform are input to the neural network, where the output is the point prediction of the cumulative number of software faults detected at the next testing day. The architecture of NN is same as previous section (see Figure 2.1)

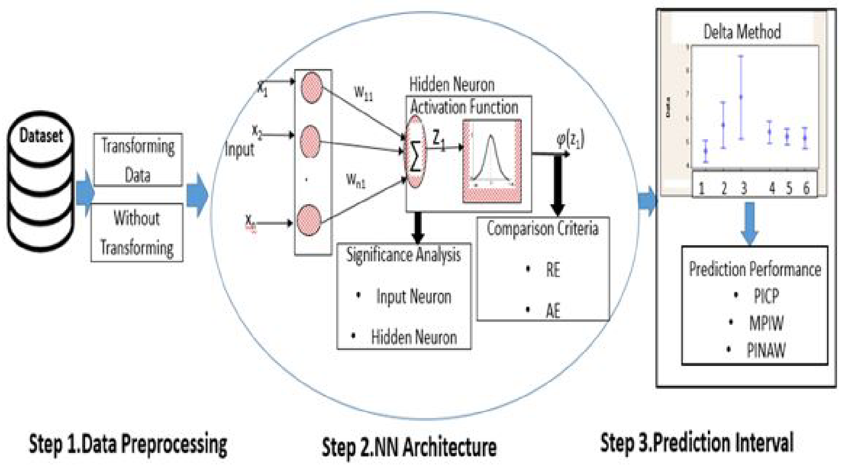


Figure 2.3: Overall experimental process.

2.4 Numerical Illustrations

2.4.1 Setup

We use eight real project data sets cited in [29]; DS1∼DS8, which consist of the software-fault count data. Table 2.3 summarizes the data sets and their cumulative numbers of software faults detected in testing. To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent, where the initial guess of weights, w_{ij}, w'_{js}, w_{0j} and w'_{0s}, are given by the uniform random varieties ranged from -1 to +1, the number of total iterations in the BP algorithm run is 1,000 and the convergence criteria on the minimum error is 0.001.

The prediction performance is evaluated in sequential software testing, so that we make the one-stage look-ahead prediction based on past observation and sequentially evaluate the prediction performance. As a prediction performance measure, we introduce the average relative error. Suppose that the observation point is the n-st testing day and that n software fault counts data are available. For the actual value on the number of software fault counts at the s-th testing day, the relative error (RE) Eq. 2.11 and average relative error (AE) Eq. 2.10 takes account of the past history.

Table 2.3: Data sets observed in study.

Dataset #	Testing days	Number of faults	Project type
1	62	133	Command and Control subsystem
2	41	351	Flight Data subsystem
3	114	144	Command and Data subsystem
4	22	54	Real Time Command & Control
5	73	367	Commercial Subsystem
6	181	224	Command and Data subsystem
7	81	461	Brazilian Electronic Switching System
8	140	100	Telecommunications switch software

2.4.2 Effect of number of hidden nodes

Hidden layers play a vital role in the performance of back Propagation neural network. A major problem in applying neural networks is specifying the size of the network. Even for moderately size networks the number of parameters may become large compared to the number of data. Many researchers put their best effort in analyzing the solution to the problem that how many neurons are kept in hidden layer in order to get the best result, but unfortunately no body succeed in finding the optimal formula for calculating the number of neurons that should be kept in the hidden layer so that the neural network training time can be reduced and also accuracy in determining target output can be increased. An "structured trial and error" method is used by the maximum developer for selecting a neural network's number of hidden neuron approximation. Y. Liu *et al.* [73] in their approach to optimize the number of neurons in the hidden layer using benchmark datasets and estimation of the signal-to-noise-ratio figure. F. Fnaiech *et al.* in [13] make an attempt to prune the hidden nodes of the feed forward architecture by initially creating a non linear activation function of hidden nodes as Taylor's expansion and then NARX (nonlinear auto regressive with exogenous input) model is used. To keep relatively small errors on software fault prediction, it is extremely important to train the neural network. The minimal error reflects better stability, and higher error reflects worst stability. We experiment with different numbers of hidden neurons for a fixed number of input neurons for all datasets. The excessive hidden neurons cause the so-called over fitting problem and tend to overestimate the complexity of the target problem. To choose the optimal number of hidden neurons we follow the well-known “ rules of thumb ” for choosing a suitable architecture. We determine the number of hidden nodes by (2/3)(number of inputs+outputs) [56].

Table 2.4 presents the prediction performance based on AE for eight datasets with and without data transformation, where the values in “ Architecture” denote the number of input neurons, the number of hidden neurons and the output neuron in this order. According to rules of thumb, we fix input neuron numbers and change the hidden neurons. From this result, FT provides the better prediction performance in all cases, except in DS#6, in here the other transformation gives the relatively small output values. In addition, we observe that the architecture, 5:3:1, 5:4:1, 10:7:1, 15:10:1 and 20:14:1, lead to less error than the other architectures. From the results we find that the rule of thumb is rather accurate to obtain the nearly optimal network architecture.

2.4.3 Effect of number of input nodes

In most situations, there is no way to determine the best number of input nodes without training several networks and estimating the generalization error. If we have too many input neurons, we may get higher training error and higher generalization error, due to under-fitting and large statistical bias. Once the best number of hidden neurons is known, we can change the input neuron numbers and can make a different architecture to find out the optimal number of input neurons.

Table 2.5 presents the prediction performance based on AE for all datasets with and without data transformation. It is shown that when the number of input neurons increases, the error rate also increases, but some cases e.g., DS#3$\sim$DS#5 without transform provide the better results for 11:7:1, 12:10:1 and 12:7:1 than FT, because AE is independent of the Gaussian distribution property. It should be noted that the neural network has one major drawback for application in software reliability assessment, i.e., in the common neural network computing the initial weight is randomly selected, so it acts as a trial-and-error heuristics.

Table 2.4: Average Relative error in point prediction.

Average Relative Error					
		Effect of hidden neurons			
Dataset #	Architecture	AT1	FT	BT	Normal
1	5:3:1	0.3370	0.1041	0.4139	0.1433
	5:4:1	0.5148	0.9771	1.3048	2.1271
	5:5:1	0.5236	0.1253	0.4253	0.3215
2	5:3:1	0.2417	0.5412	1.1774	1.1478
	5:4:1	0.1639	0.3685	0.8222	0.7348
	5:5:1	1.5236	1.0235	3.1253	1.9356
3	10:5:1	1.7969	0.5747	0.9225	0.9162
	10:7:1	0.9043	0.2685	0.7069	0.6712
	10:8:1	2.5369	1.2536	3.1235	3.3698
4	15:5:1	1.8627	1.2515	1.9189	1.3079
	15:10:1	0.6845	0.6122	0.8647	0.7851
	15:15:1	1.1627	1.0488	2.1147	1.2387
5	10:5:1	1.3895	0.9713	1.1568	0.6985
	10:7:1	0.8704	0.7001	0.9882	0.9257
	10:8:1	1.7845	1.0134	1.1575	1.6134
6	20:5:1	2.1763	0.986	0.5412	1.1398
	20:10:1	1.3895	0.9713	1.1568	2.7634
	20:14:1	0.9852	1.1207	0.7613	1.2078
7	10:5:1	1.4713	0.9231	1.4801	0.9474
	10:7:1	0.9958	0.4847	1.2128	0.7487
	10:8:1	0.7856	0.8945	0.9645	1.2356
8	5:3:1	1.9457	1.3531	1.8537	0.9147
	5:4:1	0.8387	0.6914	1.3731	0.9537
	5:5:1	2.0931	1.0537	1.4526	1.0535

Table 2.5: Average Relative error in point prediction.

Average Relative Error					
		Effect of input neurons			
Dataset #	Architecture	AT1	FT	BT	Normal
1	4:3:1	0.7856	0.2796	0.9225	0.8262
	5:3:1	0.3045	0.0685	0.5706	0.3671
	6:3:1	2.3075	0.7943	2.4508	3.3484
2	4:4:1	1.2591	0.7447	1.2135	0.9344
	5:4:1	0.5196	0.0481	0.8222	0.3687
	6:4:1	2.1235	0.9536	3.1253	1.4589
3	10:7:1	0.7469	0.1548	0.2386	0.3489
	11:7:1	0.9632	0.8456	1.2536	0.2356
	12:7:1	1.2536	1.1025	4.1253	1.8523
4	12:10:1	2.4562	1.4523	1.9825	0.9456
	13:10:1	1.7458	1.1023	2.0123	1.0425
	14:10:1	0.9256	0.8596	1.1235	1.2536
5	10:7:1	1.2456	0.9819	1.1523	3.1245
	11:7:1	1.4253	1.4856	1.4586	2.8569
	12:7:1	2.8563	1.8156	1.8963	1.4523
6	12:10:1	2.4528	1.8456	0.9827	2.8452
	13:10:1	1.7852	0.9856	1.7459	1.4756
	14:10:1	0.4587	0.4523	0.9336	1.1245
7	10:7:1	1.8745	0.8965	1.3785	0.9859
	11:7:1	2.2851	1.6285	1.1789	1.1287
	12:7:1	1.3570	0.8745	2.4859	2.5897
8	4:4:1	1.8956	1.4523	0.9856	1.6456
	5:4:1	1.2451	0.5423	1.4523	1.1235
	6:4:1	2.4589	1.4253	1.8567	1.8596

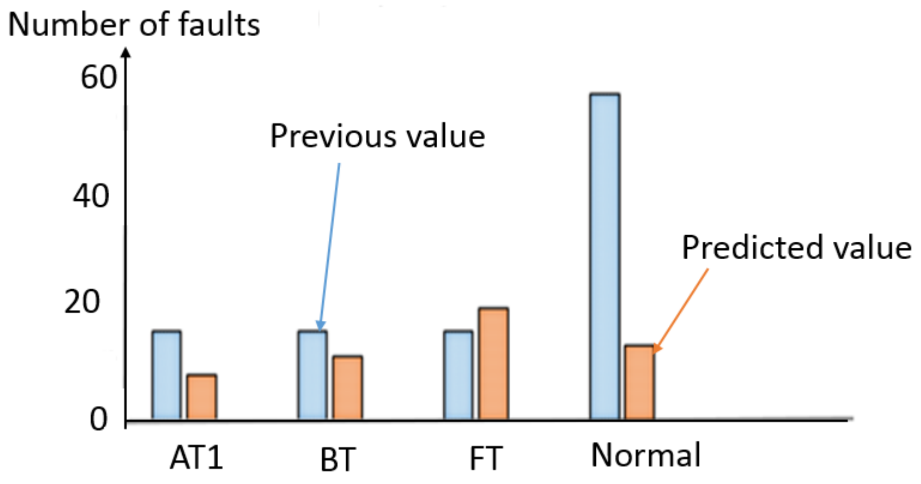


Figure 2.4: s+1 testing day predicated value (DS4 22:15:1).

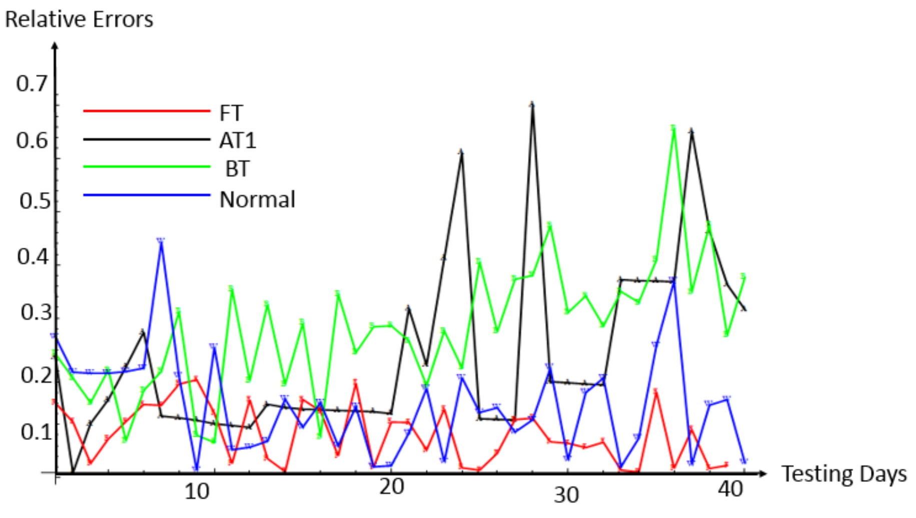


Figure 2.5: Time-dependent behavior of relative error (DS7 10:7:1).

2.4.4 One-stage look ahead point prediction

To get the one-stage look ahead point prediction value, we use DS # 4. Noting that our dataset contains 22 days, we wish to know 23th point prediction value of the cumulative number of software faults for all transform methods. From the result of Tables 2.4 and 2.5, it is seen that FT can give less error rate and more reliable prediction results than the others. Figure 2.4 shows that result based on the best neural network architecture. From this figure we can say that FT gives little larger value than the others.

Since software is written by humans, errors will be always involved in the product. From this well-known fact, it can be recognized that the PIs can predict the number of software fault counts under uncertainty, which will experience in the future operational phase, and can be useful for the probabilistic inference with subjective significance level controlled by the software test manager. We have investigated the effect of a number of hidden and input nodes on prediction accuracy according to the rules of thumb. The experimental results have shown that the proposed approach gave the acceptable results for prediction using the different neural network architectures.

Chapter 3

Multi-stage look-ahead prediction : Refined Neural Network Architecture

In this chapter, we consider the long-term prediction of the number of software faults, and propose a refined NN approach with the grouped data, where the multi-stage look-ahead prediction is carried out with a simple MLP neural network with multiple outputs. Under the assumption that the software fault count data follow a Poisson process with an unknown mean value function, we transform the underlying Poisson count data to the Gaussian data via five data transformation methods. Next, we predict the long-term behavior of software fault counts by the neural network. In numerical examples with eight actual software fault data sets, we compare our neural network approach with the existing software reliability growth models based on an nonhomogeneous Poisson process, in terms of predictive performance with average relative error. It is shown that our neural network approach affords a more appropriate prediction device and tends to have an enhanced performance of the viewpoint of predictability in the early phase of software testing.

3.1 Model Description

3.1.1 NHPP-based Software Reliability Modeling

Here we summarize the software reliability growth modeling. Suppose that a software system test starts at time $t = 0$. Let $X(t)$ be the cumulative number of software faults detected by time t, where $\{X(t), t \geq 0\}$ denotes a stochastic (non-decreasing) counting process in continuous time. In particular, it is said that $X(t)$ is a NHPP if the following conditions hold:

- $X(0) = 0$,
- $X(t)$ has independent increments,
- $\Pr\{X(t+h) - X(t) \geq 2\}{=}o(h)$,
- $\Pr\{X(t+h) - X(t) = 1\}{=}\ \lambda(t;\boldsymbol{\theta})h + o(h)$,

where $o(h)$ is the higher term of infinitesimal time h, and $\lambda(t;\boldsymbol{\theta})$ is the intensity function of an NHPP which denotes the instantaneous fault detection rate per each fault. In the above definition, $\boldsymbol{\theta}$ is the model parameter (vector) included in the intensity function. Then, the probability that the cumulative number of software faults detected by time t equals x is given by

$$\Pr\{X(t) = x\} = \frac{\{\Lambda(t;\boldsymbol{\theta})\}^x}{x!} \exp\{-\Lambda(t;\boldsymbol{\theta})\}, \tag{3.1}$$

where

$$\Lambda(t;\boldsymbol{\theta}) = \int_0^t \lambda(x;\boldsymbol{\theta})dx \tag{3.2}$$

is called the mean value function and indicates the expected cumulative number of software faults up to time t, say, $\Lambda(t;\boldsymbol{\theta})$ =E$[X(t)]$.

If the mean value function $\Lambda(t;\boldsymbol{\theta})$ or the intensity function $\lambda(t;\boldsymbol{\theta})$ is specified, then the identification problem of the NHPP is reduced to a statistical estimation problem of unknown model parameter $\boldsymbol{\theta}$. In this way, when the parametric form of the mean value function or the intensity function is given, the resulting NHPP-based SRGMs is called parametric NHPP-based SRGMs. Table 3.1 contains the representative NHPP-based SRGMs and their mean value functions. Okamura and Dohi [41] summarized these eleven parametric NHPP-based SRGMs and developed a parameter estimation tool, SRATS (Software Reliability Assessment Tool with Spread Sheet), based on the maximum likelihood method, and the EM (Expectation and Maximization) algorithm. In SRATS, the best SRGM with the smallest AIC (Akaike information criterion) is automatically selected, so the resulting best SRGM can fit best the past data on software fault counts among the eleven models.

Suppose that n realizations of $X(t_i)$, x_i $(i = 1, 2, \ldots, n)$, are observed up to the observation point t $(\geq t_n)$. We estimate the model parameter $\boldsymbol{\theta}$ by means of the maximum likelihood method. Then, the log likelihood function for the grouped data (t_i, x_i) $(i = 1, 2, \ldots, n)$ is given by

$$\begin{aligned} LLF(\boldsymbol{\theta}) &= \textstyle\sum_{i=1}^{n} \Big((x_i - x_{i-1}) \log\big\{\Lambda(t_i; \boldsymbol{\theta}) - \Lambda(t_{i-1}; \boldsymbol{\theta})\big\} \\ &\qquad - \log\big\{(x_i - x_{i-1})!\big\}\Big) - \Lambda(t_n; \boldsymbol{\theta}), \end{aligned} \tag{3.3}$$

where $\Lambda(0; \boldsymbol{\theta}) = 0$, $x_0 = 0$ and $t = t_n$ for simplification. The maximum likelihood estimate of the model parameter, $\hat{\boldsymbol{\theta}}$, can be obtained by maximizing Eq.(5.3) with respect to the model parameter $\boldsymbol{\theta}$. Once the model parameter is estimated, our next concern is to predict the future value of the intensity function or the mean value function at an arbitrary time t_{n+l} $(l = 1, 2, \ldots)$, where l denotes the prediction length. In parametric modeling, the prediction at time t_{n+l} is easily done by substituting the estimated model parameter $\hat{\boldsymbol{\theta}}$ into the time evolution $\Lambda(t; \boldsymbol{\theta})$, where the unconditional and conditional mean value functions as an arbitrary future time t_{n+l} are given by

$$\Lambda(t_{n+l}; \hat{\boldsymbol{\theta}}) = \int_0^{t_{n+l}} \lambda(x; \hat{\boldsymbol{\theta}}) dx, \tag{3.4}$$

$$\begin{aligned} \Lambda(t_{n+l} | X(t_n) = x_n; \hat{\boldsymbol{\theta}}) &= x_n + \int_{t_n}^{t_{n+l}} \lambda(x; \hat{\boldsymbol{\theta}}) dx \\ &= x_n + \Lambda(t_{n+l}; \hat{\boldsymbol{\theta}}) - \Lambda(t_n; \hat{\boldsymbol{\theta}}). \end{aligned} \tag{3.5}$$

When the mean value function is unknown, a few nonparametric approaches have been developed [62],[47]. However, it should be noted that those approaches can deal with the fault -detection time data, but do not work for future prediction. The wavelet-based method in [54] can treat the grouped data, but fails to make the long-term prediction in nature. In the following section, we use an elementary ANN for the purpose of long-term software fault prediction.

3.1.2 A Refined Neural Network Approach

Artificial neural network (ANN) is a computational metaphor inspired by the

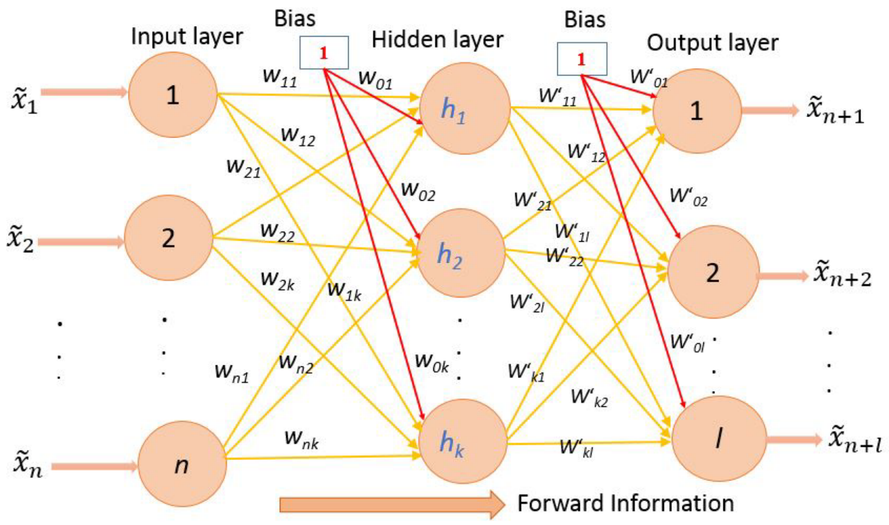


Figure 3.1: Architecture of back propagation type MIMO.

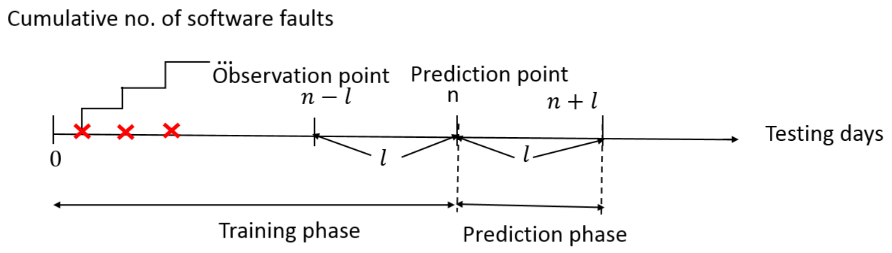


Figure 3.2: Configuration of prediction scheme via MIMO.

brain and nervous system study, and consists of an input layer with some inputs, multiple hidden layers with hidden neurons and one output layer. The input layer of neurons can be used to capture the inputs from the outside world. Since the hidden layer of neurons has no communication with the external world, the output layer of neurons sends the final output to the external world. Hence, determining an appropriate number of hidden neurons is an important design issue in neural computation. In here we consider a multiple-inputs multiple-outputs (MIMO) neural network with only one hidden layer. Similar to Section 3.1, suppose that n software fault count data (t_i, x_i) $(i = 1, 2, \ldots, n)$ are observed at the observation point t $(= t_n)$. Our concern is about the future prediction of the cumulative number of software faults at time t_{n+l} $(l = 1, 2, \ldots)$.

3.1.2.1 *Preliminary Set-up:*

In the common neural computation, it is noted that the ANN including the simplest MLP with only one output neuron is regarded as a nonlinear regression model, where the explanatory variables are randomized by the Gaussian white noise. In other words, the output data in the MLP is implicitly assumed to be a realization of a nonlinear Gaussian model. On the other hand, since one handles the Poisson count data as integer values in the software fault prediction, the underlying data shall be transformed to the Gaussian data in advance. Such a pre-data processing is common in the wavelet shrinkage estimation [54], but has not been considered in the software fault prediction via the ANN. According to the idea by Xiao and Dohi [54], we apply five data transform techniques from the Poisson data to the Gaussian data. Table 3.2 and Section 1.4 summarizes the data transform techniques used and their inverse transform formulae. In the table, x_i denotes the cumulative number of software faults detected at i ($i = 1,\ 2, \ldots, n$)-th testing day. Then, we have the transformed data $\tilde{x}_i$ by means of any data transformation method. Let $\tilde{x}_i$ ($i = 1, 2, \ldots, n$) and $\tilde{x}_{n+l}$ ($l = 1, 2, \ldots$) be the input and output for the MIMO neural network, respectively. Then, the prediction of the cumulative number of software faults is given by the inversion of the data transform. Figure 3.1 depicts the architecture of back propagation type MIMO, where n is the number of software fault count data experienced before the observation point t_n and l is the prediction length. We suppose that there is only one hidden layer with k ($= 1, 2, \ldots$) hidden neurons in our MIMO neural network.

3.1.2.2 *Training Phase:*

Suppose that all the connection weights (nk weights from input to hidden layer, kl weights from hidden to output layer in Fig. 3.1) are first given by the uniformly distributed pseudo random varietes. In the MIMO, if these weights are completely known, then it is possible to calculate $(\tilde{x}_{n+1}, \ldots, \tilde{x}_{n+l})$ from the input $(\tilde{x}_1, \ldots, \tilde{x}_n)$ directly. However, since it is impossible to train all the weights including $k(n+l)$ unknown patterns in principle via the common BP algorithm, it is needed to develop a new long-term prediction scheme for the MIMO. Suppose that $n > l$ without any loss of generality. In Fig. 3.2, we illustrate the configuration of our prediction scheme. In order to predict the number of software faults for l testing days from the observation point t_n, the prediction has to be made at the point t_{n-l}. This implies that only $(n-l)k + kl = nl$ weights can be estimated with the training data experienced for the period $(t_{n-l}, t_n]$ and that the remaining $k(n+l) - nl$ weights are not trained at time t_n. We call these $k(n+l) - nl$ weights the *non-estimable* weights in this paper. As the prediction length is longer, the number of non-estimable weights becomes greater and the prediction uncertainty also increases more. In this scheme, the transformed data $(\tilde{x}_1, \ldots, \tilde{x}_{n-l})$ are used for the input in the MIMO, and the remaining data $(\tilde{x}_{n-l+1}, \ldots, \tilde{x}_n)$ are used for the teaching signals in the training phase. The BP algorithm is the well-known gradient descent method used to update the connection weights, so as to minimize the squared error between the network output values and the teaching signals. For the value coming out of an input neuron, $\tilde{x}_i$ $(i = 1,\ 2,\ \ldots,\ n-l)$, it is common to add two special inputs; bias units which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. Let $w_{ij}\ \in [-1, 1]$ be the connection weight from i-th input neuron to j-th hidden neuron, where w_{0j} and w'_{0s} denote the bias weights for j-th hidden neuron and s-th output neuron, respectively for the training phase with $i = 0, 1, \ldots, n-l$, $j = 0, 1, \ldots, k$ and $s = n-l+1, n-l+2, \ldots, n$. Each hidden neuron calculates the weighted sum of the input neuron, h_j, in the following equation:

$$h_j = \sum_{i=1}^{n-l} \tilde{x}_{ij} w_{ij} + w_{0j}. \tag{3.6}$$

Since there is no universal method to determine the number of hidden neurons, we change k in the pre-experiments and choose an appropriate value. After calculating h_j for each j , we apply a sigmoid function $f(h_j) = 1/\exp(-h_j)$ as a threshold function in the MIMO. Since h_j are summative and weighted inputs from respective hidden neuron, the s-th output $s = n-l+1, n-l+2, \ldots, n$ in the output layer is given by

$$\tilde{x}_s = \sum_{j=1}^{k} f(h_j) w'_{js} + w'_{0s}. \tag{3.7}$$

Because $\tilde{x}_s$ are also summative and weighted inputs from respective hidden neuron in the output layer, the weight w'_{js} is connected from j-th hidden neuron to s-th output neuron. The output value of the network in the training phase, $\tilde{x}_s$, is calculated by $f(\tilde{x}_s) = 1/\exp(-\tilde{x}_s)$. In the BP algorithm, the error is propagated from an output layer to a successive hidden layer by updating the weights, where the error function is defined by

$$\mathrm{SSE} = \frac{\sum_{s=n-l+1}^{n}(\tilde{x}_s^o - \tilde{x}_s)^2}{(l-1)} \tag{3.8}$$

with the prediction value $\tilde{x}_s$ and the teaching signal $\tilde{x}_s^o$ observed for the period $(t_{n-l+1}, t_n]$.

The BP algorithm updates the weight parameters so as to minimize SSE between the network output values $\tilde{x}_s$ $(s = n-l+1, n-l+2, \ldots, n)$ and the teaching signals $\tilde{x}_s^o$, where each connection weight is adjusted using the gradient descent according to the contribution to SSE in Eq. (3.8). The momentum, α, and the learning rate, η, are controlled to adapt the weights and the convergence speed in the BP algorithm, respectively. Since these are the main tuning parameters in the BP algorithm, we carefully examine these parameters in pre-experiments. We set α and η like as previous in Section 2.1.2. Then, the connection weights are updated in the following:

$$\begin{aligned} w_{ij(new)} &= w_{ij} + \alpha w_{ij} + \eta \delta h_j \tilde{x}_i \\ &\qquad (i = 1, 2, \ldots, n-l, j = 1, \ldots, k), \qquad (3.9) \\ w'_{js(new)} &= w'_{js} + \alpha w'_{js} + \eta \delta \tilde{x}_s \tilde{x}_s \\ &\qquad (j = 1, 2, \ldots, k, s = n-l+1, \ldots, n), \qquad (3.10) \end{aligned}$$

where δh_j and $\delta \tilde{x}_s$ are the output gradient of j-th hidden neuron and the output gradient in the output layer, and are defined by

$$\begin{aligned} \delta h_j &= f(h_j)(1 - f(h_j)), \qquad (3.11) \\ \delta \tilde{x}_s &= \tilde{x}_s(1 - \tilde{x}_s)(\tilde{x}_s^o - \tilde{x}_s), \qquad (3.12) \end{aligned}$$

respectively. Also, the updated bias weights for hidden and output neurons are respectively given by

$$w_{0j(new)} = w_{0j} + \alpha w_{0j} + \eta\delta h_j, \tag{3.13}$$

$$w'_{0s(new)} = w'_{0s} + \alpha w'_{0s} + \eta\delta\tilde{x}_s. \tag{3.14}$$

The above procedure is repeated until the desired output is achieved.

3.1.2.3 *Prediction Phase:*

Once the nl weights are estimated with the training data experienced for the period $(t_{n-l}, t_n]$ through the BP algorithm, we need to obtain the remaining $k(n+l) - nl$ non-estimable weights for prediction. Unfortunately, since these cannot be trained with the information at time t_n, we need to give these values by the uniform pseudo random varieties ranged in $[-1, 1]$. By giving the random connection weights, the output as the prediction of the cumulative number of software faults, $(\tilde{x}_{n+1}, \ldots, \tilde{x}_{n+l})$, are calculated by replacing Eqs. (3.6) and (3.7) by

$$h_{j(new)} = \sum_{i=1}^{n} \tilde{x}_{ij} w_{ij(new)} + w_{0j(new)}, \tag{3.15}$$

$$\tilde{x}_{n+s} = \sum_{j=1}^{k} f(h_{j(new)}) w'_{js(new)} + w'_{0s(new)}, \tag{3.16}$$

respectively, for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, k$ and $s = 1, 2, \ldots, l$. Note that the resulting output is based on one sample by generating the uniform pseudo random variates only once. In order to obtain the prediction of the expected cumulative number of software faults, we generate m sets of random variates and take the arithmetic mean of the m predictions of $(\tilde{x}_{n+1}, \ldots, \tilde{x}_{n+l})$, where $m = 1,000$ is confirmed to be enough in our preliminary experiments. In other words, the prediction in the MIMO neural network is reduced to a combination of the BL learning and a Monte Carlo simulation on the connection weights.

3.2 Numerical Experiments

We use eight real project data sets cited in the reference [29]; DS1$\sim$ DS8, which consist of the software fault count (grouped) data . In Table 3.3 summarizes the data sets used for analysis, where “Testing days” means the total testing days,

n denotes the total number of software faults detected in testing, and “Project type” implies the project name where each dataset are used.

To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent, where the initial guess of weights, w_{ij}, w'_{js}, w_{0j} and w'_{0s}, are given by the uniform random varieties ranged from -1 to +1, the number of total iterations in the BP algorithm and the convergence criteria on the minimum error issame as previous. In Figures 3.5 and 3.6, we give two examples on how to determine the optimal transformation parameter λ^* for BoxCox power transformation. In our experiments, it is shown that the search range of λ should be $[-3, +2]$.

Suppose that the observation point is given by the n-th testing day, t_n. In this case, $(n - l)$ software fault counts data are used for training the MIMO neural network. The capability of the prediction model is measured by the average relative error (AE),

$$AE_l = \frac{\sum_{s=1}^{l} RE_s}{l}, \tag{3.17}$$

where RE_s is called the relative error for the future time $t = n + s$ and is given by

$$RE_s = \left| \frac{(\tilde{x}^o_{n+s} - \tilde{x}_{n+s})}{\tilde{x}^o_{n+s}} \right| \quad (s = 1, 2, \ldots, l). \tag{3.18}$$

So we regard the prediction model with smaller AE as a better prediction model.

3.2.1 Discussion

In appendix A and B summarize the results on AE for the underlying data set DS1∼ DS8 at 50% ∼ 90% observation points of the whole data for the prediction length l=5, 10, 15, and 20 days. For instance, we divided AE by two ways, according to observation point and prediction length. Table 3.4 ∼ 3.5 shows the result for AE according to observation point for all datasets. In here, l means that prediction length, “ ’Middle Testing” represents 50% ∼ 70% observation point, and “Late Testing” implies the observation point 80%– 90%. From these results, it can be seen that (i) Our MIMO (BT) neural network could work well to predict the cumulative number of software faults in the early testing phase, (ii) Late testing phase SRGM (txvmax) provides the better result than MIMO.

Table 3.6–3.7 shows the result of AE according to prediction length for all datasets. Where, *O_Point* means observation point, "Short Term" implies that 5–10 testing days and "Long Term" indicate 15–20 testing days. For short-term point prediction, most of the case MIMO method provides a better result for relatively small testing time except DS2 at 70% $\sim$ 90% observation point. Surprisingly, this is true even if the testing time longer our refined NN approach provides a better result. Summary of the two criteria represent by Table 3.8 for all datasets.

Figures 3.3 $\sim$ 3.4 we illustrate the time-dependent behavior of RE and compare SRGMs with DS1 and DS6 at 60% observation point for 20 output length respectively. From this result, it can be observed that from early testing to late testing AT2 provide less error than others. Without transform (Normal) deals worse result. SRGM (llogist) without early testing phase it sounds good (see Figure 3.3). On the other hand , in Figure 3.4 BT gives better result than SRGM. Though we omit to show all the results for brevity, it can be seen that the SRGM (gamma) leads very close results to the BT.

From Figures 3.5 and 3.6, it can be recognized that the adjustment of λ is quite sensitive to the predictive performance and has to be done through the try-and-error heuristics. However, for an arbitrary λ, we can know that the multi-stage look-ahead prediction of software fault count is possible with the MIMO type of MLP.

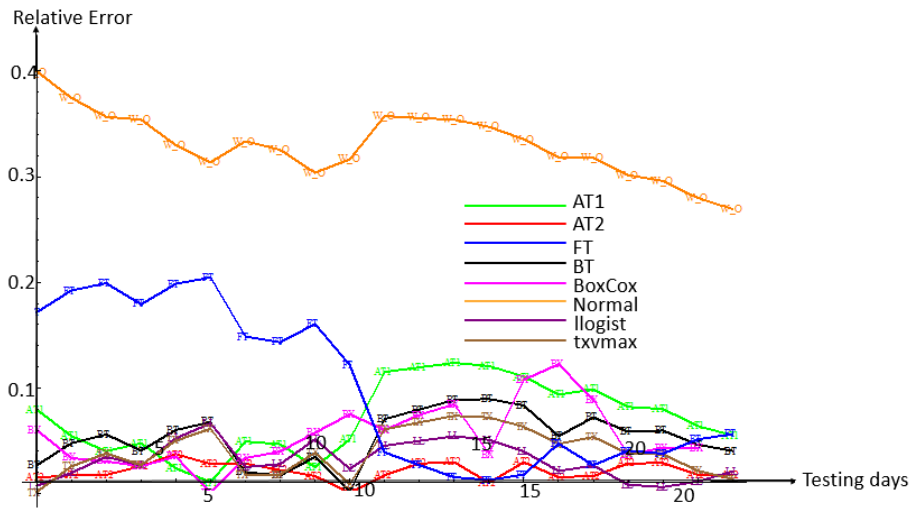


Figure 3.3: Time-dependent behavior of relative error with DS1 at 60% observation point.

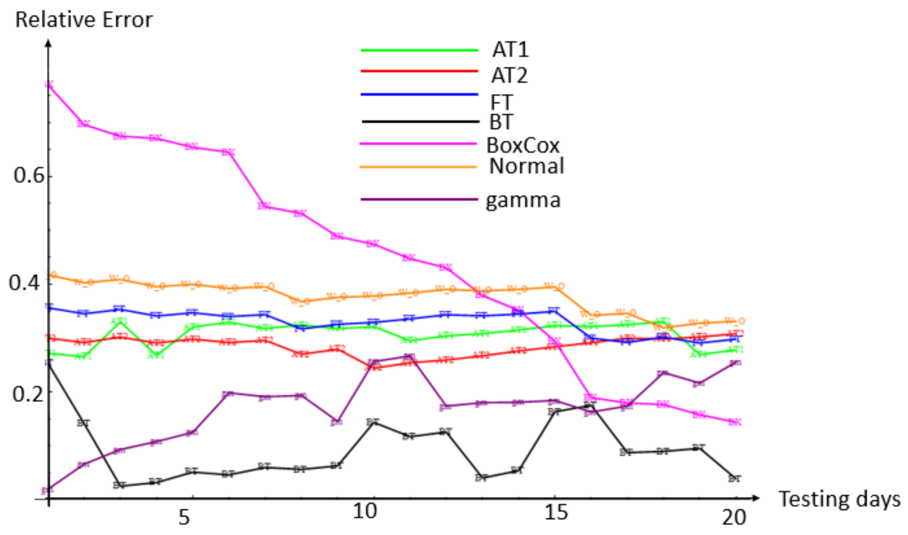


Figure 3.4: Time-dependent behavior of relative error with DS6 at 60% observation point.

Table 3.1: NHPP-based SRGMs.

Model (Abbr.)	Mean value function
Exponential (exp) [14]	$\Lambda(t) = aF(t)$ $F(t) = a\{1 - \exp(-bt)\}$
Gamma (gamma) [55]	$\Lambda(t) = aF(t)$ $F(t) = \int_0^t \frac{c^b s^{b-1} \exp(-cs)}{\Gamma(b)} ds$
Pareto (pareto) [1, 28]	$\Lambda(t) = aF(t)$ $F(t) = 1 - (\frac{c}{t+c})^b$
Truncated normal (tnorm)[40]	$\Lambda(t) = a\frac{F(t)-F(0)}{1-F(0)}$ $F(t) = \frac{1}{\sqrt{2\pi}b} \int_{-\infty}^t \exp(-\frac{(s-c)^2}{2b^2}) ds$
Log normal (lnorm) [3, 40]	$\Lambda(t) = aF(\log t)$ $F(t) = \frac{1}{\sqrt{2\pi}b} \int_{-\infty}^t \exp(-\frac{(s-c)^2}{2b^2}) ds$
Truncared logistic (tlogist) [38]	$\Lambda(t) = a\frac{F(t)-F(0)}{1-F(0)}$ $F(t) = \frac{1}{1+\exp(-\frac{t-c}{b})}$
Log logistic (llogist) [16]	$\Lambda(t) = aF(\log t)$ $F(t) = \frac{1}{1+\exp(-\frac{t-c}{b})}$
Truncated extreme value maximum (txvmax) [39]	$\Lambda(t) = a\frac{F(t)-F(0)}{1-F(0)}$ $F(t) = \exp(-\exp\{(-\frac{t-c}{b})\})$
Log extreme value maximum (lxvmax) [39]	$\Lambda(t) = aF(\log t)$ $F(t) = \exp(-\exp\{(-\frac{t-c}{b})\})$
Truncated extreme value minimum (txvmin) [39]	$\Lambda(t) = a\frac{F(0)-F(-t)}{F(0)}$ $F(t) = \exp(-\exp\{(-\frac{t-c}{b})\})$
Log extreme value minimim (lxvmin) [15, 39]	$\Lambda(t) = a(1 - F(-\log t))$ $F(t) = \exp(-\exp\{(-\frac{t-c}{b})\})$

Table 3.2: Data transform formulae.

	Formulae	
Method	Data transform	Inversion transform
AT1 [4]	$\tilde{x}_i = 2\sqrt{x_i + 3/8}$	$\frac{\tilde{x}_{n+l}^2 - 3/2}{4}$
AT2 [32]	$\tilde{x}_i = 2\sqrt{x_i + 1/8}$	$\frac{2 * \tilde{x}_{n+l}^2 - 1}{8}$
BT [5]	$\tilde{x}_i = 2\sqrt{x_i + 1/2}$	$\frac{\tilde{x}_{n+l}^2 - 2}{4}$
FT [12]	$\tilde{x}_i = \sqrt{x_i + 1} + \sqrt{x_i}$	$\frac{\tilde{x}_{n+l}^2 + \tilde{x}_{m+l}^{-2} - 2}{4}$
BoxCox [8]	$when \quad \lambda = 0, \quad \tilde{x}_i = log(x_i)$ $when \quad \lambda \neq 0, \quad \tilde{x}_i = \frac{x_i^\lambda - 1}{\lambda}$	$exp(\tilde{x}_{n+l})$ $(\lambda * \tilde{x}_{n+l} + 1)^{1/\lambda}$

Table 3.3: Data sets observed for multi-stage look ahead prediction.

Dataset	Testing days	n	Project type
1	62	133	Command and Control subsystem
2	41	266	Flight Data subsystem
3	46	144	Control application system
4	109	553	Real-time control application & Control
5	111	481	Monitoring and real-time control system
6	73	367	Commercial Subsystem
7	81	461	Brazilian Electronic Switching System
8	114	144	Telecommunications switch software

Table 3.4: AE according to observation point for four (DS1 ∼ DS4) datasets

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO BoxCox	MIMO FT	MIMO BoxCox	SRGM txvmin	SRGM txvmax
10	MIMO AT2	SRGM tlogist	MIMO AT2	SRGM lxvmax	
15	MIMO AT2	MIMO AT2	SRGM txvmax		
20	MIMO BoxCox	MIMO AT2			

DS1

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO FT	MIMO FT	SRGM lxvmin	SRGM lxvmin	SRGM txvmax
10	MIMO AT2	MIMO AT2	SRGM lxvmin		
15	MIMO FT	MIMO BoxCox			
20	MIMO AT1				

DS2

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO AT2	MIMO BT	MIMO BT	MIMO FT	SRGM txvmax
10	MIMO BT	SRGM txvmax	SRGM txvmax		
15	MIMO AT1	SRGM txvmax			
20	MIMO FT				

DS3

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO FT	SRGM txvmax	SRGM txvmax	SRGM lxvmax	SRGM lxvmax
10	MIMO BoxCox	MIMO BT	SRGM lxvmax	MIMO AT2	
15	SRGM txvmax	MIMO AT1	SRGM txvmax		
20	MIMO BT	MIMO AT1			

DS4

Table 3.5: AE according to observation point for four (DS5 ∼ DS8) datasets

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	SRGM gamma	SRGM txvmax	SRGM txvmax	SRGM lxvmax	SRGM txvmax
10	MIMO FT	SRGM lxvmax	SRGM txvmax	SRGM lxvmax	SRGM txvmax
15	MIMO BT	SRGM txvmax	SRGM txvmax	SRGM txvmax	
20	MIMO BT	SRGM txvmax	MIMO BT		

DS5

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO BT	MIMO FT	SRGM txvmax	MIMO FT	SRGM lxvmax
10	MIMO FT	MIMO BT	MIMO FT	SRGM txvmax	
15	MIMO BT	MIMO AT2	MIMO FT	SRGM txvmax	
20	MIMO AT1	MIMO BT	MIMO AT1		

DS6

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO FT	SRGM txvmax	SRGM txvmax	MIMO BT	SRGM txvmax
10	MIMO BT	MIMO AT1	MIMO FT	MIMO FT	
15	MIMO BT	SRGM txvmax	MIMO AT1	SRGM txvmax	
20	MIMO BT	MIMO FT	MIMO BT		

DS7

l	Middle Testing			Late Testing	
	50%	60%	70%	80%	90%
5	MIMO FT	SRGM lxvmax	SRGM lxvmax	MIMO BT	MIMO BT
10	MIMO BoxCox	MIMO BT	MIMO FT	SRGM lxvmax	SRGM exp
15	MIMO FT	MIMO AT2	MIMO BT	SRGM txvmax	
20	MIMO AT2	MIMO BT	MIMO AT2	MIMO BT	

DS8

Table 3.6: AE according to prediction length for four (DS1 ~ DS4) datasets

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO BoxCox	MIMO AT2	MIMO AT2	MIMO BoxCox
60%	MIMO FT	SRGM tlogist	MIMO AT2	MIMO AT2
70%	MIMO BoxCox	MIMO AT2	SRGM txvmax	
80%	SRGM txvmin	SRGM lxvmax		
90%	SRGM txvmax			

DS1

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO FT	MIMO AT2	MIMO FT	MIMO AT1
60%	MIMO FT	MIMO AT2	MIMO BoxCox	
70%	SRGM lxvmin	SRGM lxvmin		
80%	SRGM lxvmin			
90%	SRGM txvmax			

DS2

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO AT2	MIMO BT	MIMO AT1	MIMO FT
60%	MIMO BT	SRGM txvmax	SRGM txvmax	
70%	MIMO BT	SRGM txvmax		
80%	MIMO FT			
90%	SRGM txvmax			

DS3

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO FT	MIMO BoxCox	SRGM txvmax	MIMO BT
60%	SRGM txvmax	MIMO BT	MIMO AT1	MIMO AT1
70%	SRGM txvmax	SRGM lxvmax	SRGM txvmax	
80%	SRGM lxvmax	MIMO AT2		
90%	SRGM lxvmax			

DS4

Table 3.7: AE according to prediction length for four (DS5 ~ DS8) datasets

O_Point	Short Term		Long Term	
	5	10	15	20
50%	SRGM gamma	MIMO FT	MIMO BT	MIMO BT
60%	SRGM txvmax	SRGM lxvmax	SRGM txvmax	SRGM txvmax
70%	SRGM txvmax	SRGM txvmax	SRGM txvmax	MIMO BT
80%	SRGM lxvmax	SRGM lxvmax	SRGM txvmax	
90%	SRGM txvmax	SRGM txvmax		

DS5

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO BT	MIMO FT	MIMO BT	MIMO AT1
60%	MIMO FT	MIMO BT	MIMO AT2	MIMO BT
70%	SRGM txvmax	MIMO FT	MIMO FT	MIMO AT1
80%	MIMO FT	SRGM txvmax	SRGM txvmax	
90%	SRGM lxvmax			

DS6

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO FT	MIMO BT	MIMO BT	MIMO BT
60%	SRGM txvmax	MIMO AT1	SRGM txvmax	MIMO FT
70%	SRGM txvmax	MIMO FT	MIMO AT1	MIMO BT
80%	MIMO BT	MIMO FT	SRGM txvmax	
90%	SRGM txvmax			

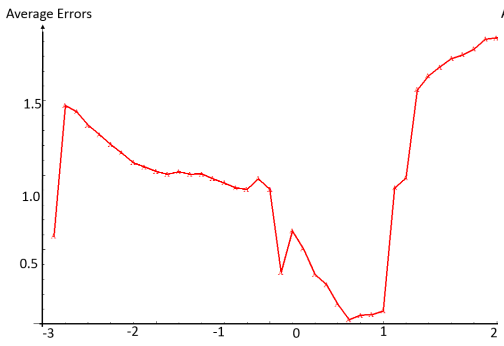
DS7

O_Point	Short Term		Long Term	
	5	10	15	20
50%	MIMO FT	MIMO BoxCox	MIMO FT	MIMO AT2
60%	SRGM lxvmax	MIMO BT	MIMO AT2	MIMO BT
70%	SRGM lxvmax	MIMO FT	MIMO BT	MIMO AT2
80%	MIMO BT	SRGM lxvmax	SRGM txvmax	MIMO BT
90%	MIMO BT	SRGM exp		

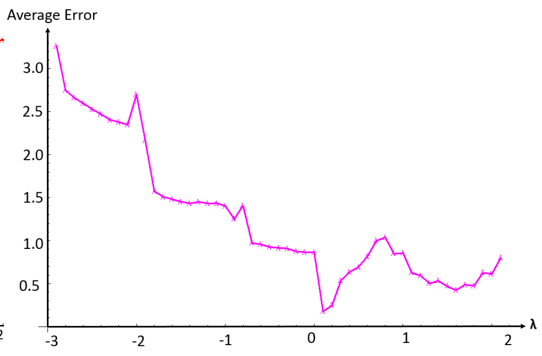
DS8

Table 3.8: Comparison table for both cases with all datasets

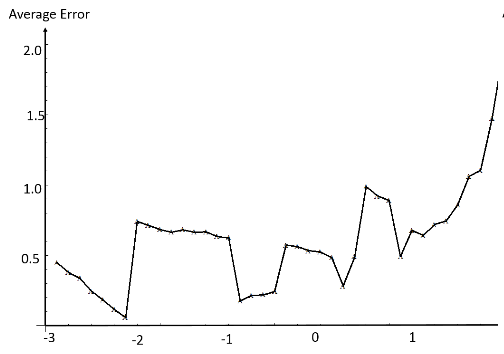
Dataset	Observation point		Prediction length	
	Middle Testing	Late Testing	Short Term	Long Term
DS1	MIMO(AT2)	SRGM(txvmax)	SRGM(txvmax)	MIMO(AT2)
DS2	MIMO(FT)	SRGM(lxvmin)	SRGM(lxvmin)	MIMO(AT1)
DS3	MIMO(BT)	SRGM(txvmax)	MIMO(BT)	SRGM(txvmax)
DS4	MIMO(AT1)	SRGM(lxvmax)	SRGM(lxvmax)	MIMO(AT1)
DS5	SRGM(txvmax)	SRGM(txvmax)	SRGM(txvmax)	SRGM(txvmax)
DS6	MIMO(BT)	SRGM(txvmax)	MIMO(FT)	MIMO(AT1)
DS7	MIMO(BT)	SRGM(txvmax)	MIMO(FT)	MIMO(BT)
DS8	MIMO(AT2)	MIMO(BT)	MIMO(BT)	MIMO(BT)



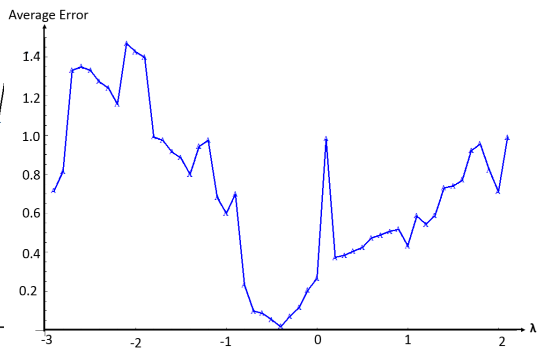
(a) 5 output neurons with 30 hidden neurons.



(b) 10 output neurons with 20 hidden neurons.

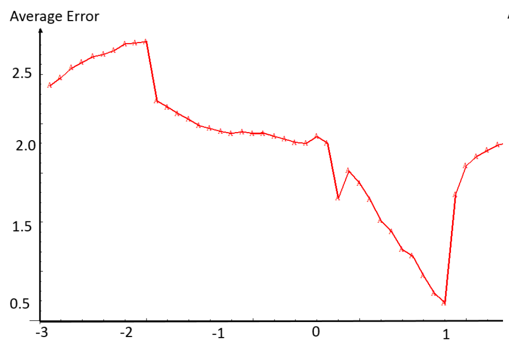


(c) 15 output neurons with 10 hidden neurons.

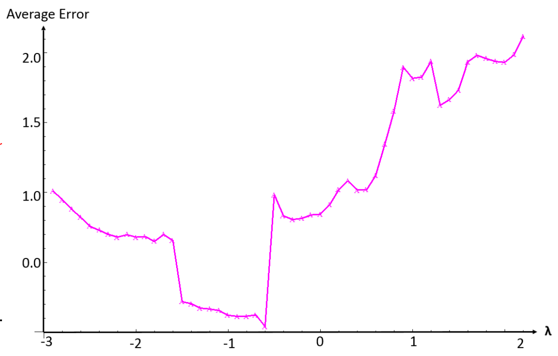


(d) 20 output neurons with 50 hidden neurons.

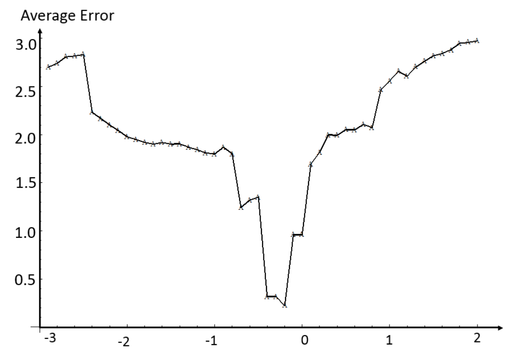
Figure 3.5: Determination of the transformation parameter λ (DS1 with 50% observation point).



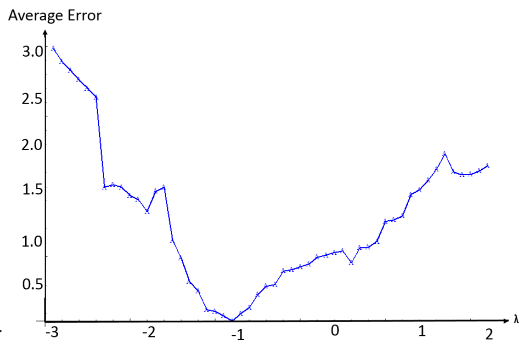
(a) 5 output neurons with 50 hidden neurons.



(b) 10 output neurons with 20 hidden neurons.



(c) 15 output neurons with 10 hidden neurons.



(d) 20 output neurons with 40 hidden neurons.

Figure 3.6: Determination of the transformation parameter λ (DS2 with 50% observation point).

Finally, we cannot obtain the strong conclusion on how to design the MIMO for the purpose of software fault prediction. However, the lesson learned from the numerical experiments suggests that the multi-stage look-ahead prediction of software fault count is possible with the MIMO neural network, and that the data transform from the Poisson data to the Gaussian data works better to predict the number of software faults accurately.

3.3 Simulation Experiments

To investigate the accuracy of our methods, Monte Carlo simulation is carried out. Here, we also focus on the single time data. Suppose that the (unknown) time process follows an exponential NHPP model, $\lambda(t) = a\{1 - \exp(-bt)\}$, with model parameters $(a, b) = (413.2050, 0.0461)$. We generate the original failure time data $X^*_{k,i}$ for $i = 1,\ 2,\ \ldots,\ n$, at k-th simulation as the pseudo random variables, are given by $0 < T_1 \leq T_2 \leq \cdots \leq T_n$ with realizations $0 < t_1 \leq t_2 \leq \cdots \leq t_n$ by the *thinning algorithm* [30]. Consider the case where the failure time distribution $F(t)$ and the intensity function $\lambda(t)$ are completely unknown. Then, it can be shown that $X^*_{k,1},\ X^*_{k,2},\ \cdots\ (k = 1, 2, \ldots, m)$ follows an NHPP with intensity function $\lambda(t)$, where $(m = 1000)$, after that we make grouped data from time data. That is, it is assumed that n failures occur by time t and the realizations of T_i $(i = 1, 2, \ldots, n)$, say, t_i are observed, where $t_n \leq t$. Without any loss of generality, we can make count data from the random variable $X_i = T_i \leq T_n \in [i = 1, 2, \ldots, n]$ with realizations $x_i = t_i \leq t_n \in [i = 1, 2, \ldots, n]$ for $i = 1,\ 2,\ \ldots,\ n$, less than or equal to the maximum length. We regard the pair (t_i, x_i) $(i = 1, 2, \ldots, n)$ as a software count data of the underlying NHPP. From 1000 samples we select 4 types of cases (1) Case1, Fitted which is fit with real mean value function, (2) Caes2, Underfitted is not exact fit with real mean value function, (3) Case3, over fit to the real value is called Overfitted dataset and (4) Case4, S-shaped which behavior like s. Figure 3.7 and 3.8 show the structure of two datasets Case1 and Case4 respectively. In here, we recognized the dataset as like as Case1, Case2, Case3 and Case4 respectively.

3.3.1 Point Estimation

For n software fault data, we estimate point prediction for long-term. In that case, we observed 50% past data as our input to the NN for next 15 days. In previous Section 3.1.2 we described our refined NN approach. After preprocessing our simulation 4 datasets we estimated point prediction for long-term by five data transformation method with the Normal method. Table 3.2 and Section 1.4 summarizes the data transform techniques used and their inverse transform formulae. Suppose that the observation point is given by the n-th testing day, t_n. In this case, (n-l) software fault counts data are used for training the MIMO type of MLP. The capability of the prediction model is measured by the average error (AE) cited in Eq.(3.17). where RE_s is called the relative error for the future time $t = n + s$ and is given by in Eq. (3.18). So we regard the prediction model with smaller AE as a better prediction model.

Table 3.9 summarize the results on AE for the underlying cases Case1, Case2, Case3 and Case4 at 50% observation points of the whole data for the prediction length $l = 15$ days, where " Best λ" denotes the optimal transformation parameter in the sense of minimum AE, and the bold number implies the best prediction model in the same category. For instance, Table 3.9 gives the prediction results on the cumulative number of software faults for 15 days at respective observation points, when the number of hidden neurons changes from $k = 10 \sim 50$. In the MIMO type of MLP neural network, we compare five data transform with the non-transformed case (Normal). It is seen that our MIMO-based approaches provide smaller AEs than the Normal in almost all cases when the observation point is 50%. Incase of Case1, the best prediction model is the transformed MIMO (FT) with $k = 40$. On the other hand, for Case2 MIMO with FT offers less error than the Normal MIMO with $k = 20$. Focusing on the number of hidden neurons in the MIMO type of MLPs, we expected that the larger k may lead to the better predictive performance. In terms of predictive performance, Case3 and Case4 with BT and FT provides the best prediction result than Normal respectively. However, in the simulation experiment results, it is seen that the data transformation can work well to give more accurate prediction results in the MIMO type of MLPs. In the MIMO-based approach, it is essential to determine feasible k and λ values, because the number of hidden neurons produce more expensive computation cost with dierent prediction length l. Unfortunately, no universal method to determine the optimal λ and hidden neuron number.

In Figure 3.9, we illustrate the time-dependent behavior of RE and compare six data transform methods (Normal) with Case1. From this result, it can be observed that without middle testing phase FT delivers less error. The common approach Normal gives the worse result in the full testing phase, comparing with other. On the other hand, the most standard BT, AT2, and BoxCox provide the worse results and give the growing trend as the software testing. AT1 delivers average error rate from first to last testing phase.

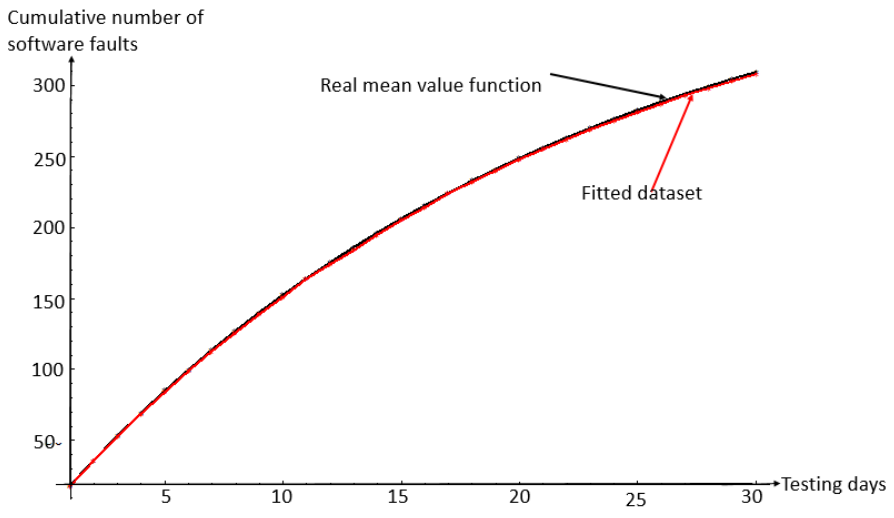


Figure 3.7: Case1 Fitted dataset

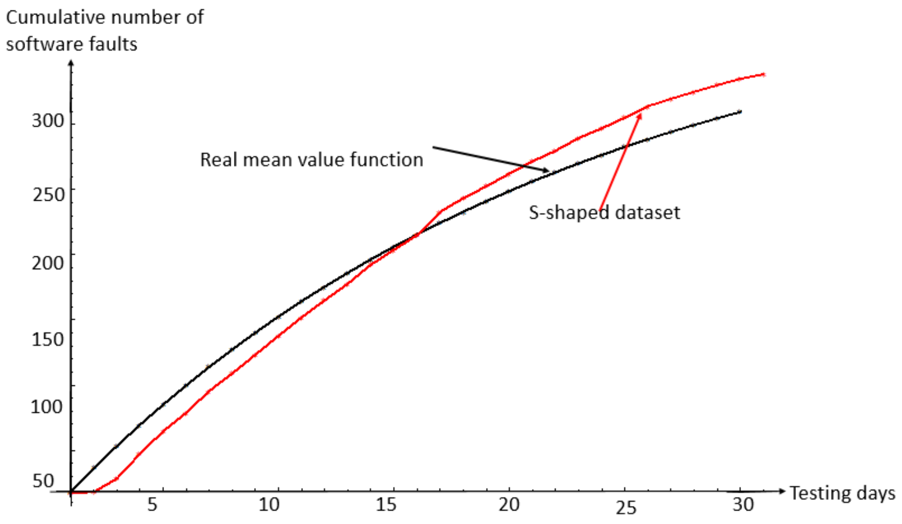


Figure 3.8: Case4 S-shaped dataset

Table 3.9: Comparison of average relative errors for fifteen days prediction with Case1 $\sim$ Case4 ($l = 15$).

50% observation ($t_n = 15$)						
Case2						
Average Error(AE)						
	MIMO					
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal
10	1.0563	1.0952	0.8952	0.7514	1.3698(1.1)	2.4589
20	0.8531	1.1361	0.0756	1.369	0.9458(1.0)	1.9412
30	0.1362	0.2492	0.0911	0.3448	0.2771(0.7)	0.5062
40	0.3589	0.8781	**0.0396**	0.6391	1.069(0.3)	1.3497
50	0.1015	0.2639	0.0963	0.3940	0.6852(0.1)	0.7423
Case2						
10	2.5896	3.4712	2.4820	1.6352	3.6094(1.1)	4.8956
20	2.1356	2.9841	**0.0641**	2.4810	1.0489(0.8)	3.6523
30	3.0564	1.8971	0.6749	0.9423	1.6357(1.4)	1.4758
40	0.9852	0.0918	0.0686	1.2301	0.6987(-0.3)	0.3497
50	0.0779	1.2891	0.9852	0.3289	0.4024(1.3)	0.504579
Case3						
10	2.1324	0.0654	0.6894	2.4120	2.3145(-1.9)	4.2536
20	1.4578	1.6932	0.0335	1.9482	0.2837(2.0)	2.5896
30	0.0229	1.3472	1.2035	**0.0151**	1.9823(0.0)	0.3811
40	0.6417	0.9856	1.4712	0.7519	1.8742(1.2)	1.9786
50	0.2471	0.8945	0.3691	0.2981	0.8974(1.0)	1.8421
Case4						
10	3.6451	1.6523	0.6417	0.8741	2.8963(1.3)	3.6012
20	1.6481	1.9631	1.2013	1.2475	1.9876 (-1.1)	3.1987
30	1.9685	0.3781	1.2489	0.3419	1.0947(0.9)	2.8475
40	0.2896	1.0143	1.0321	0.6389	1.6981(-0.8)	1.2589
50	1.0241	0.2781	**0.2389**	0.3141	0.8949(-0.3)	1.3698

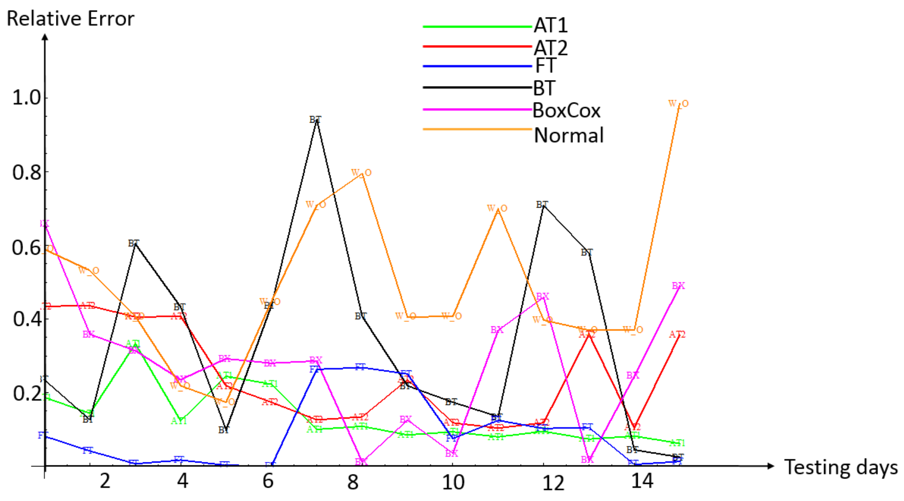


Figure 3.9: Time-dependent behavior of relative error with Case1 ($k = 40$).

Chapter 4

Predictive Interval: Refined Neural Network Approach

In this chapter, we consider predictive interval for software fault data by neural network approach. It is noted that the ANN including the simplest MLP with only one output neuron is regarded as a nonlinear regression model, where the explanatory variables are randomized by the Gaussian white noise. In other words, the output data in the MLP is implicitly assumed to be a realization of a nonlinear Gaussian model. On the other hand, since one handles the Poisson count data as integer values in the software fault prediction, the underlying data shall be transformed to the Gaussian data in advance. Two of these are based on the fault data, one is one stage look ahead prediction and an other is multi-stage look-ahead prediction, which are introduced in Chapter 2 and 3. Throughout examples with real software fault data and simulation experiment, it is shown that the proposed methods provide more accurate estimation results. In the traditional statistics, the interval estimation is useful to take account of the uncertainty of point estimate itself, though it is difficult to obtain analytically the statistical estimator distribution of the target output.

4.1 Delta Method

Delta method is known as an elementary method of propagation of errors. It is a commonly used approach which is easily implemented, not computer-intensive, and can be robustly applied to many situations such as an approximation of the variance for an arbitrary functional of random variables, based on Taylor series expansions. It is also used to obtain an asymptotic statistical estimator from the knowledge of the limiting variance [60], [27].

Let δ_r^T is the output gradient vector with respect to gradient values for all output and hidden neurons in the MPL neural network:

$$\delta_r^T = [\delta\tilde{x}_1, \delta\tilde{x}_2, \ldots, \delta\tilde{x}_s, \delta h_1, \delta h_2, \ldots, \delta h_j] \tag{4.1}$$

where δh_j and $\delta\tilde{x}_s$ are the output gradient of j-th hidden neuron and the output gradient in the output layer, and are defined at Eq. (2.8) In practice, the neural network parameters such as pervious weights have to be adjusted by minimizing the average SSE^2. Let Δw_r is the Jacobian matrix with respect to all updated weight parameters from an output neuron to hidden neurons. It is computed for all the training samples, where

$$\Delta w_r = \begin{bmatrix} w'_{1s(new)} & w_{11(new)} & w_{21(new)} & \cdots & w_{i1(new)} \\ w'_{2s(new)} & w_{12(new)} & w_{22(new)} & \cdots & w_{i2(new)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w'_{js(new)} & w_{1j(new)} & w_{2j(new)} & \cdots & w_{ij(new)} \end{bmatrix} \tag{4.2}$$

In the above equation, $w'_{js(new)} (s = 1)$ and $(j = 1, \ldots, k)$ are the new weights of hidden neurons connected to the output neuron, and $w_{ij(new)}$ are the new weights of n input neurons which are connected to j-th hidden neuron in the hidden layer. These weights, $w'_{js(new)}$ and $w_{ij(new)}$, are given at Eq. (2.8).

Define the PIs of the cumulative number of software faults by $[PI_{low}, PI^{up}]$ in the MLP neural network computing. Then, the lower limit and upper limit of PI are given by

$$PI_{low} = \tilde{x}_s - t_{n-l}^{1-\alpha/2}\sqrt{1 + \delta_r^T(\Delta w_r^T \Delta w_r)^{-1}\delta_r} \tag{4.3}$$

$$PI^{up} = \tilde{x}_s + t_{n-l}^{1-\alpha/2}\sqrt{1 + \delta_r^T(\Delta w_r^T \Delta w_r)^{-1}\delta_r} \tag{4.4}$$

By adding (10) and (11), the PI can be expressed as

$$PI_{low}^{up} = \tilde{x}_s \pm t_{n-l}^{1-\alpha/2}\sqrt{1 + \delta_r^T(\Delta w_r^T \Delta w_r)^{-1}\delta_r} \tag{4.5}$$

In the above equations $\tilde{x}_s$ denotes the point eastimation value for s-th testing days cited in Eq. 2.2, $t_{n-l}^{1-\alpha/2}$is the $\alpha/2$ - quantile of the student t-distribution function with $(n-l)$ degree of freedom, n is the number of inputs in the neural network , and l is predicition period [27].

The Jacobian matrix $(\Delta w_r)$and its gradient value (δ_r^T) are quite hard to obtain with all input data, so delta method contains a somewhat puzzling issue to construct PIs. However the other calculations are comparatively modest. Here, the Jacobian matrix and the gradient value are calculated and estimated at off-line, although they can be potential sources of computational error for constructing PIs. In addition, the quality of PIs and their optimal values of gradient and Jacobian matrix must be carefully checked to satisfy the convergence condition that the minimum error is achieved at a tolerance level.

4.2 One-stage look ahead prediction

- **Without effect of hidden and input neuron:**

  In Chapter 2, we apply the NN methods to obtain point estimation for one-stage look-ahead prediction. In that time we pre-processing the datasets like as common in the wavelet shrinkage estimation [54], but has not been considered in the software fault prediction via the ANN. According to the idea by Xiao and Dohi [54], we apply three data transform techniques from the Poisson data to the Gaussian data. More specifically, we use BT [5], AT [4], FT [12] as the most major normalizing and variance-stabilizing transforms. Table 2.1 summarizes the data transform techniques and their inverse transform formulae. In the table, x_i denotes the cumulative number of software faults detected at i $(i = 1,\ 2, \ldots, n)$-th testing day. Then, we have the transformed data $\tilde{x}_i$ by means of any data transformation method. Let $\tilde{x}_i$ $(i = 1, 2, \ldots, n)$ and $\tilde{x}_l$ $(l = 1)$ be the input and output for the MIMO neural network, respectively. Then, the prediction of the cumulative number of software faults is given by the inversion of the data transform. We suppose that there is only one hidden layer with k $(= 1, 2, \ldots)$ hidden neurons in our MIMO neural network. After complete other equation in the Section 2.1.2 we got point estimation value for 4 datasets. We derive the two-sided 95% prediction intervals of the software fault data for one-stage look-head prediction with delta method.

We use four real project data sets cited in the reference [29]; DS1∼DS4, which consist of the software fault count (grouped) data. In these data sets, the length of software testing and the total number of detected software faults are given by (62, 133), (22, 54), (41, 351) and (114, 188) respectively.

- **PI Assesment**

In order to measure the quality of PIs on the predictive software fault counts, we need define two predictive measures called the PI coverage rate (PICP) and the mean predictive interval width (MPIW) [27]. Let set up the significance level as 95%. PCIP is the portion of the number of the software fault count covered by the PIs, and is defined by

$$PICP = \frac{\sum_{s=1}^{N} CP_s}{N}, \tag{4.6}$$

where

$$CP_s = \begin{cases} 1 & \tilde{x}_s \in [PI_{low}, PI^{up}] \\ 0 & \tilde{x}_s \ni [PI_{low}, PI^{up}] \end{cases} \tag{4.7}$$

PI_{low} and PI^{up} are the lower and upper predictive limits, $\tilde{x}_s$ is the actual number of fault count at i-th testing day ($i = 1, 2, \ldots, k-1$).

On the other hand, MPIW evaluates the width of PIs, and is defined by

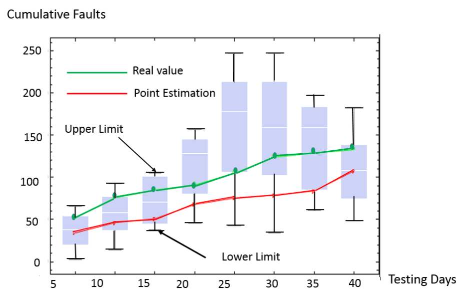
$$MPIW = \frac{\sum_{s=1}^{N} PI^{up} - PI_{low}}{N}, \tag{4.8}$$

In Table 4.1 presents the predictive PI measures for four datasets with and without data transformation, where the values in round brackets denote the number of input layer neurons and the number of hidden layer neurons. It can be seen that the data transformation does work to cover PICP with the corresponding narrow MPIWs. For practical usage of PIs, one may understand the data transformation methods are needed because their associated coverage rates are cover 95% significance level with wider lengths between the lower and upper predictive limits. However, our purpose here is to get the theoretically reasonable PIs which are consistent to

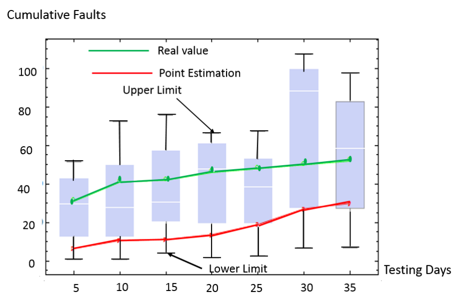
Table 4.1: Prediction PI measures.

DataSet	Prediction PI measures		
	Data transformation	PICP	MPIW
DS1(10,8)	Normal	0.9652	5056.98
	AT1	0.9505	4556.94
	FT	0.9798	3996.81
	BT	0.9545	3031.80
DS2(5,7)	Normal	0.9584	4061.82
	AT1	0.9647	3849.68
	FT	0.9695	5391.29
	BT	0.9574	5294.29
DS3(5,9)	Normal	0.9689	4998.96
	AT1	0.9798	4597.04
	FT	0.9798	5599.89
	BT	0.9691	4132.62
DS4(15,10)	Normal	0.9659	5469.40
	AT1	0.9579	5684.64
	FT	0.9654	6469.40
	BT	0.9653	5226.08

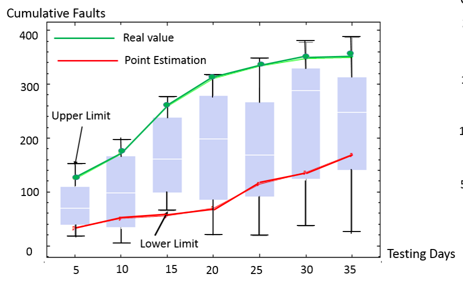
the approximate PIs based on the delta methods. This result enables us to know that the PIs used to predict the number of software fault counts which will experience in the future, and be useful for the probabilistic inference with subjective significance level controlled by the software test manager.



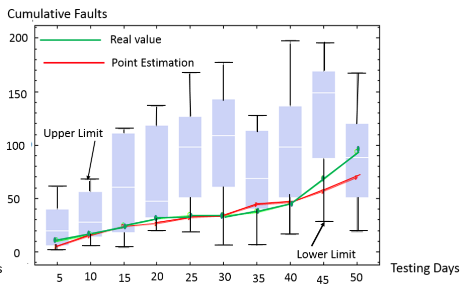
(a) AT with DS1



(b) BT with DS2



(c) Normal with DS3



(d) FT with DS4

Figure 4.1: Prediction Interval

In Figure 4.1 , we depict the sequential prediction results of software fault counts and its 95% prediction intervals with four data transformation methods with all datasets where the length of each box plot denotes the two-sided 50% prediction intervals. It can be found that all the methods cover the one-stage look-ahead prediction and the data itself within PIs except DS3 with Normal. The PI construction delta method computational load is higher than others method but also the operational planners and schedulers can enjoy the excellent quality of PI for software reliability engineering.

- **The effect of number of hidden nodes and input nodes:**

We use eight real project data sets cited in [29]; DS1∼DS8, which consist of the software fault count data. Chapter 2, Table 2.3 summarizes the data sets and their cumulative numbers of software faults detected in testing. The prediction performance is evaluated in sequential software testing, so that we make the one-stage look-ahead prediction based on the past observation and sequentially evaluate the prediction performance. In order to measure the quality of PIs on the prediction of software fault counts, we need to define three prediction measures called the PI coverage probability (PICP), the mean prediction interval width (MPIW) and PI-normalized averaged width (PINAW) [27] same as previous. PI-normalized averaged width (PINAW) quantifies the wide constructed PIs;

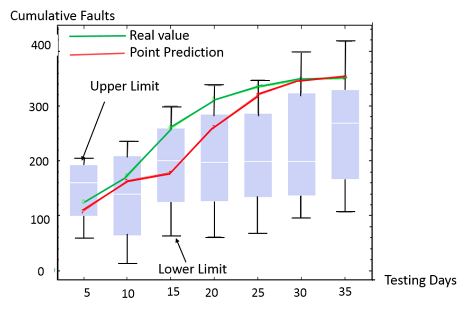
$$PINAW = MPIW/R, \tag{4.9}$$

where R is the range of the underlying target, and is used to compare PIs. In Table 4.2 we give the prediction PI measures for eight datasets. It can be seen that the data transformation works to increase PICP because the corresponding MPIWs become wider. In practical usage of PIs, the associated coverage rate is large enough if the width is narrow. From this table we can say that in most of the cases FT provides narrow width with higher coverage rate excluding DS# 6. On the other hand, BT can give the better result than FT. Since the sharper PIs are theoretically more informative and practically more useful than the wider PIs if it is contains the high coverage probability, it is noted that PINAWs indicate the sharpness of PIs, so that smaller PINAW is regarded as better PIs. In overall discussion we can observe that NN with transform provide high coverage probability with narrow width.

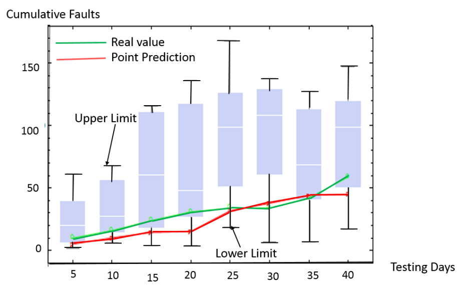
In Figure 4.14, we depict the sequential prediction results of software fault counts and their PIs with four data transformation methods (including the case with no transform) for the different datasets, where the best architecture in each dataset is applied. Figure show the two-sided 95% prediction intervals of cumulative number of software faults via delta method, where the length of each box plot denotes the two-sided 50% prediction intervals. It can be found that the PIs with FT can cover both of the one-stage

Table 4.2: Prediction PI measures.

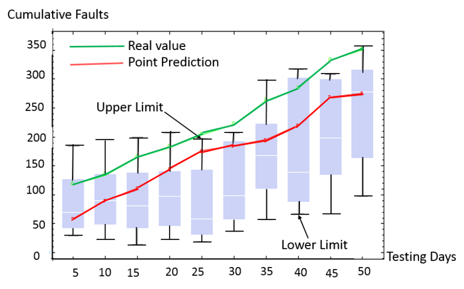
	Prediction PI measures			
Dataset #	Data transform	PICP	PINAW	MPIW
DS1(5,3)	Normal	0.9232	30.4904	152.452
	AT1	0.9762	48.4956	162.478
	FT	0.9598	42.5052	112.526
	BT	0.9532	48.557	242.785
DS2(5,4)	Normal	0.9398	88.4904	442.452
	AT1	0.9586	69.1568	345.784
	FT	0.9642	54.825	274.125
	BT	0.9786	90.4696	452.348
DS3(10,7)	Normal	0.9478	15.4754	154.754
	AT1	0.9614	54.8127	548.127
	FT	0.9585	52.4784	524.784
	BT	0.9589	45.4796	454.796
DS4(5,4)	Normal	0.9504	38.6446	192.223
	AT1	0.9654	40.3386	201.693
	FT	0.9585	38.448	192.240
	BT	0.9494	49.6048	248.024
DS5(14,10)	Normal	0.9617	15.9846	223.785
	AT1	0.9758	18.6584	261.218
	FT	0.9897	11.0492	154.689
	BT	0.9582	21.0611	294.856
DS6(10,7)	Normal	0.9398	105.2142	1052.142
	AT1	0.9584	45.6989	456.989
	FT	0.9554	66.2489	662.489
	BT	0.9855	43.2689	432.689
DS7(5,4)	Normal	0.9281	53.8294	269.147
	AT1	0.9588	56.9528	284.764
	FT	0.9515	33.896	169.480
	BT	0.9445	71.3624	356.812
DS8(5,4)	Normal	0.9476	100.4246	502.123
	AT1	0.9678	129.169	645.845
	FT	0.9585	50.425	252.125
	BT	0.9589	90.429	452.145



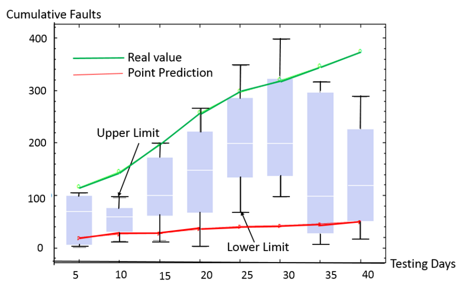
(a) FT with DS2



(b) FT with DS3



(c) BT with DS5



(d) Normal transformation with DS7

Figure 4.2: Predictive Interval with optimal hidden and input neurons.

look-ahead point prediction and the actual data itself with DS3 (See (b)). On the other hand, (c) shows that BT with DS5 does not cover the real value in PIs. In addition, (d) illustrates that the case with no transform for DS7 does not cover point estimation and real value.

4.3 PI for multi-stage look-ahead prediction

In biological organisms, neural networks are computational metaphor inspired by the brain and nervous system study. They are greatly evaluated by some mathematical models to understand these nervous mechanisms, which consist of many simple processing units, called neurons. Neurons have interconnections with weights to encode the knowledge of the whole network, which has a learning algorithm so as to automatically develop internal representations. The most widely used processing-unit models are the logistic function and the sigmoid function. The MLP feed forward neural network consists of three types of layers; an input layer, hidden layer and output layer. The input layer of neuron can be used to capture the inputs from the outside world. The hidden layer of neuron has no communication with the external world, but the output layer of neuron sends the final output to the external world. The main task of hidden layer neurons is to receive the inputs and weights from the previous layer and to transfer the aggregated information to the output layer by any transfer function. For multi-stage look-ahead prediction we consider a multiple-inputs multiple-outputs (MIMO) neural network with only one hidden layer. Similar to Chapter 3 Section 3.1.2, suppose that n software fault count data (t_i, x_i) $(i = 1, 2, \ldots, n)$ are observed at the observation point t $(= t_n)$. Our concern is about the future prediction of the cumulative number of software faults at time t_{n+l} $(l = 1, 2, \ldots)$. After that we calculated the point estimation for long term prediction as follows as previous Section 3.1.2, subsection 3.1.2.1, 3.1.2.2 and 3.1.2.3.

4.4 Numerical Experiments

We use eight real project data sets cited in the reference [29]; DS1$\sim$ DS8, which consist of software fault count (grouped) data in 3.3. To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent. We set all values like as previously.

4.4.1 Interval Estimation

4.4.1.1 Real Data Anlysis

We derive the two-sided 95% prediction intervals of the software fault data with delta and simulation based predictive interval methods. For interval estimation we used eight datasets at the 60% observation point only 20 output neuron except DS2 and DS3 at 50% observation point.

- **Delta Method**

  In the previous Section 4.2, we discuss delta approach for construction of prediction region with software cumulative number of fault data, since the main purpose of this section is to analyze the eight real project data sets of the software fault count data.

- **Simulation based predictive interval method**

```
Set $w_{ij}(new)$ and $output_m$ to empty;
Set $n$ to the total number ofinput node in the NN;
Set $k$ to the hidden neuron number;
Set $l$ to the output neuron number;
For $(i = 1;\ i \leq n;\ i++)$ {
For $(j = 1;\ j \leq k;\ j++)$ {
Randomly generate $w_{ij}(new)$ $(w_{ij}(new) \in [-1,\ +1))$;
Calculate $(h_j)$ by:
Chapter 3 Subsection 3.1.2.3  Eq. (3.15)
For $(s = 1;\ s \leq l;\ s++)$ {
Calculate $(\tilde{x}_{n+s})$ by:
Chapter 3 Subsection 3.1.2.3 Eq. (3.16)
For $output_m$ $(m \in [0,\ 1000])$;
Add $output_m$ to $\tilde{x}_{n+s}$ ;}}}
Finally, we calculate the two-sided $100(1-\alpha)$ %
prediction intervals with significance level $\alpha \in (0,1)$.
[$output_(m(\alpha/2)),\ output_(m(1-\alpha/2))$],   [$PI_{low}$,$PI^{up}$]
/*$PI_{low}$ and $PI^{up}$  is the lower and upper bound of prediction interval*/
```

Figure 4.3: A algorithm for simulation based predictive interval method.

In the previous Chapter 3, we discuss about refined NN approach for long term point estimation. In subsection prediction phase 3.1.2.3 we estimated with the training data experienced for the period $(t_{n-l}, t_n]$ through the BP algorithm, we need to obtain the remaining $k(n+l)-nl$ non-estimable weights for prediction. Unfortunately, since these cannot be trained with the information at time t_n, we need to give these values by the uniform pseudo random varieties ranged in $[-1, 1]$. By giving the random connection weights, the output as the prediction of the cumulative number of software faults, $(\tilde{x}_{n+1}, \ldots, \tilde{x}_{n+l})$, are calculated by Eqs. (3.15) and (3.16) respectively in Chapter 3 Subsection 3.1.2.3, for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, k$ and $s = 1, 2, \ldots, l$. Note that the resulting output is based on one sample of generating the uniform pseudo random variates only once. In order to obtain the prediction interval of the expected cumulative number of software faults, we generate m sets of random variates where $m = 1,000$ is confirmed to be enough in our preliminary experiments. The Figure 4.3 is a simple algorithm to calculate the simulation based predictive interval method, where $\alpha = 0.05$.

Figures 4.4–4.5 show the two-sided 95% predictive intervals of software fault data in the case of $l = 20$ via two prediction interval methods, where the length of each box plot denotes the two-sided 50% prediction intervals with DS1. First we can find that the prediction intervals for delta method are longer than those for simulation based method. This is because the delta method is subject to much more uncertainty. Among delta approaches, the non transform MIMO approach (Normal) gives tighter prediction regions with low coverage probability, so Normal does not cover point estimation and real value. Second, it is seen that the transformation for MIMO with AT1, AT2, BT, FT and BoxCox is included point estimation and real value within the box range.

In order to measure the quality of PIs on the predictive software fault counts, we need define predictive measures called the PI coverage rate (PICP), the mean predictive interval width (MPIW) and PI-normalized averaged width (PINAW) [27] same as previous. Let set up the significance level as 95%. Tables 4.3 and 4.7 present the comparison results of two prediction interval methods for DS1$\sim$ DS4 and DS5$\sim$ DS8 data sets where $l = 20$. In the table, “Data_T” means the data transformation, “Delta” means delta method and “Simulation” implies that the simulation based method for prediction interval. From this result, it is seen that the delta approach wider than simulation based method. On the other hand, the simulation based method even narrow but its coverage probability is over 95% significance level. In addition Normal for both cases, it does not cover the coverage probability. Even MIMO with BoxCox also shows the worse result for coverage probability in case of DS4 and DS5.

- **Real predictive interval method**

  To validate the simulation based and delta method we need to calculate real predictive interval for our four simulation data sets such as Fitted, Underfitted, Overfitted and S-shaped denoted as Case1, Case2, Case3 and Case4. In previous Chapter 3 in Section 3.3 we discussed about simulation experiment data sets, where we generate $m = 1,000$ random samples by *thinining algorithm.* As like as previous section 4.4.1.2 we regard the pair (t_i, x_i) as a realization of the underlying NHPP. For real predictive interval, the time process follows an exponential NHPP model, $\Lambda(t) = aF(t)$, $F_i(t) = a\{1 - \exp(-bt)\}$ with model parameters $(a, b) = (68764.95, 0.00177), (299.59, 0.00262), (183.87, 0.000416)$, and $(389.93, 0.000182)$ for Case1, Case2, Caes3, and Case4 respectively. Then Conditional probability and thinning algorithm is used to generate random variable time sequences 1000 time. Afterthat, we remake group data from a time sequence like as the previous case. Finally, we calculate the two-sided $100(1-\alpha)\%$ predictive intervals with significance level $\alpha \in (0, 1)$, where $\alpha = 0.05$.

- **Point estimation of real predictive interval**

  Based on the past software bug counting experiences, we predict the future value of the intensity function or the mean value function from the observation time t. In parametric modeling, the prediction is easily done by substituting estimated model parameter $\hat{\boldsymbol{\theta}}$ into the time evolution in $\Lambda(t; \hat{\boldsymbol{\theta}})$. By using SRATS tool [41], given x_{50} software bug data at time t_{50}, we estimate the best model parameter value $\hat{\boldsymbol{\theta}}$ at time t in the sense of maximum likelihood and the (unconditional and conditional) mean value function at an arbitrary future time t_{100}:

$$\Lambda(t_{100}; \hat{\boldsymbol{\theta}}) = \int_0^{t_{100}} \lambda(x; \hat{\boldsymbol{\theta}}) dx, \tag{4.10}$$

$$\begin{aligned} \Lambda(t_{100} | X(t_{50}) = x_{50}; \hat{\boldsymbol{\theta}}) &= x_{50} + \int_{t_{50}}^{t_{100}} \lambda(x; \hat{\boldsymbol{\theta}}) dx \\ &= x_{50} + \Lambda(t_{100}; \hat{\boldsymbol{\theta}}) - \Lambda(t_{50}; \hat{\boldsymbol{\theta}}). \end{aligned} \tag{4.11}$$

Figures 4.6–4.13 show the two-sided 95% predictive intervals of software fault data in all cases with delta and simulation based method respectively. Observing the whole data, we can easily realize the same result like as privious. For Case1 and Case4, MIMO with AT1, FT, and BT provides real value and point prediction within box plot (see Figures 4.7 and 4.8, 4.13 and 4.14) with delta and simulation based method respectively. In addition, the results in the remaining cases are same as ones in the neural network model including with AT2 and BoxCox. Table 4.6 shows the result of real predictive interval at 50% observation point with all cases. In that case, real predictive interval width is narrow but its coverage probability is over 95% significance level.

4.4.1.2 Simulation Experiment Data Analysis

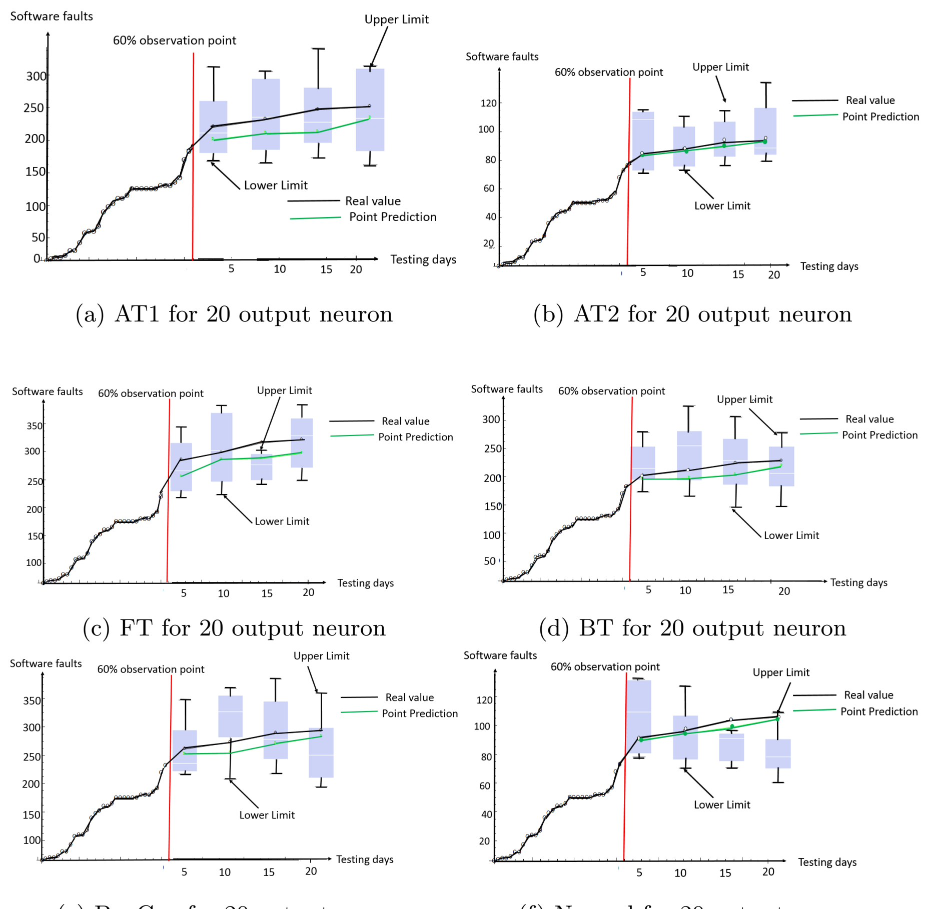
In previous Chapter 3, in Section 3.3 we discussed about simulation experimental data sets. Same as previous, we assume that the form of intensity function is

Table 4.3: DS1∼ DS4.

60% observation point						
DS1						
	Delta			Simulation		
Data _T	PICP	MPIW	PINAW	PICP	MPIW	PINAW
AT1	0.9589	2174.98	108.75	0.9785	2152.74	107.63
AT2	0.9525	2112.45	105.62	0.9512	1618.45	100.66
FT	0.9458	1049.78	52.48	0.9569	2013.25	98.23
BT	0.9545	2098.25	109.91	0.9584	1964.78	49.92
BoxCox	0.9558	1075.25	53.75	0.9348	998.48	80.92
Normal	0.9289	1017.12	49.85	0.9308	1025.12	51.96
50% observation point						
DS2						
AT1	0.9585	2945.85	147.29	0.9683	2143.57	107.19
AT2	0.9472	2515.56	125.78	0.9545	1945.41	97.27
FT	0.9580	3258.86	162.94	0.9540	2145.78	107.29
BT	0.9654	3982.41	199.12	0.9635	3120.12	156.01
BoxCox	0.9523	2134.57	106.73	0.9571	2051.23	102.56
Normal	0.9445	1325.85	66.29	0.9347	1643.27	82.16
50% observation point						
DS3						
AT1	0.9558	5015.47	250.77	0.9595	4019.92	200.99
AT2	0.9549	4045.89	200.64	0.9681	3647.72	182.38
FT	0.9581	4012.12	200.61	0.9533	3782.25	189.11
BT	0.9625	4317.82	215.89	0.9543	3758.87	187.94
BoxCox	0.9455	2915.85	145.79	0.9565	3247.29	162.36
Normal	0.9248	2654.35	132.72	0.9351	2471.19	123.56
DS4						
AT1	0.9532	5263.56	263.19	0.9683	6458.51	322.93
AT2	0.9582	5796.59	289.83	0.9523	5894.73	294.74
FT	0.9685	6125.49	306.27	0.9519	4785.84	239.29
BT	0.9651	5872.20	293.61	0.9453	4506.89	225.34
BoxCox	0.9451	4969.58	248.48	0.9351	4891.47	244.57
Normal	0.9263	2920.23	146.01	0.9289	1279.91	63.99

Table 4.4: DS5∼ DS8.

60% observation point						
DS5						
	Delta			Simulation		
Data_T	PICP	MPIW	PINAW	PICP	MPIW	PINAW
AT1	0.9545	6589.91	299.54	0.9735	6989.87	349.49
AT2	0.9659	7156.59	357.83	0.9686	6789.46	339.47
FT	0.9586	7258.51	362.92	0.9541	6243.31	312.16
BT	0.9647	7564.43	378.22	0.9547	5678.71	283.93
BoxCox	0.9453	5489.94	274.50	0.9458	5472.82	273.64
Normal	0.9382	4234.37	211.72	0.9453	4989.59	249.48
DS6						
AT1	0.9582	4712.24	235.61	0.9535	4576.67	228.83
AT2	0.9512	4523.33	226.17	0.9579	4971.18	248.56
FT	0.9612	4871.17	243.56	0.9589	4283.23	214.16
BT	0.9510	4279.29	213.96	0.9519	3986.58	199.33
BoxCox	0.9543	4585.83	229.29	0.9534	4575.51	228.77
Normal	0.9245	2987.49	149.37	0.9452	3698.45	184.92
DS7						
AT1	0.9586	5698.45	284.92	0.9615	6125.89	306.29
AT2	0.9512	4953.23	247.66	0.9536	4989.91	249.46
FT	0.9586	5941.49	297.07	0.9585	6124.47	306.22
BT	0.9596	5210.31	260.52	0.9589	5426.63	271.33
BoxCox	0.9556	4653.28	232.66	0.9495	4523.21	226.16
Normal	0.9289	3961.45	198.07	0.9453	3212.24	160.61
DS8						
AT1	0.9541	745.41	37.27	0.9555	643.23	32.16
AT2	0.9652	659.86	32.99	0.9656	712.20	35.61
FT	0.9553	598.89	29.94	0.9562	562.12	28.11
BT	0.9659	963.12	48.16	0.9585	496.56	24.83
BoxCox	0.9532	489.87	24.49	0.9594	589.89	29.49
Normal	0.9251	233.53	11.68	0.9343	375.86	18.79



(a) AT1 for 20 output neuron

(b) AT2 for 20 output neuron

(c) FT for 20 output neuron

(d) BT for 20 output neuron

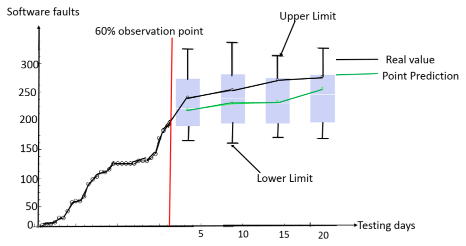
(e) BoxCox for 20 output neuron

(f) Normal for 20 output neuron

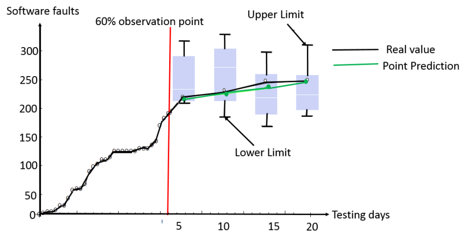
Figure 4.4: Prediction Interval for Delta method with DS1.

completely known. Suppose that the failure-occurrence time data under minimal fault, which are the random variables, are given by $0 < T_1 \leq T_2 \leq \cdots \leq T_n$. That is, it is assumed that n failures (minimal faults) occur by time t and the realizations of T_i $(i = 1, 2, \ldots, n)$, say, t_i are observed, where $t_n \leq t$. We regard the pair (t_i, x_i) as a realization of the underlying NHPP. In here, we used point estimation of our four simulation data sets such as Fitted, Overfitted, Underfitted and S-shaped. After that we applied delta method described in Section 4.2 and simulation based prediction interval method described in Section 4.4.1.1.

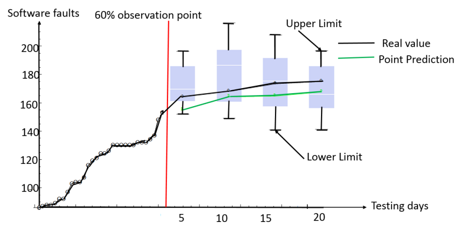
Let set up the significance level as 95%. We estimated predictive interval for 50% observation point with l=15 for all data sets. Table 4.5 shows the result of predictive measures with all cases such as Fitted, Underfitted, Overfitted and



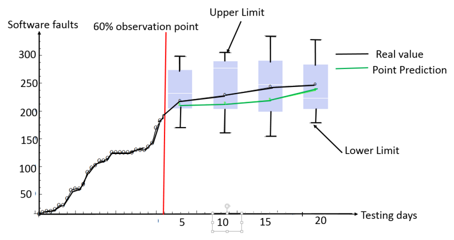
(a) AT1 for 20 output neuron



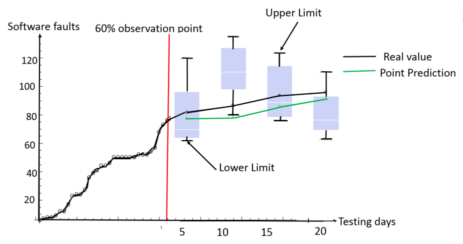
(b) AT2 for 20 output neuron



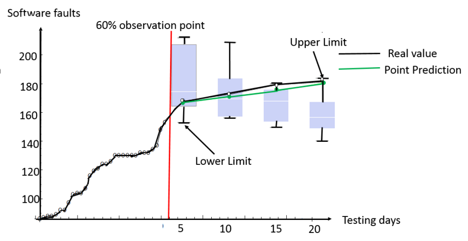
(c) FT for 20 output neuron



(d) BT for 20 output neuron



(e) BoxCox for 20 output neuron



(f) Normal for 20 output neuron

Figure 4.5: Predictive Interval for Simulation based method with DS1.

S-shaped dataset respectively. First we can find that the measures for the delta method is wider than those for simulation based model. In addition, all almost all cases transformation with MIMO wider and coverage rate over 95%. On the other hand, Normal is narrow and does not include the coverage probability.

Figures 4.6-4.13 presents the prediction interval of two methods, delta and simulation based on predictive interval method respectively with all cases. All of the cases of AT1, FT and BT contain the point estimation and real value in their prediction region. On the other hand, Normal with simulation based method coverage rate sounds good than delta method. AT2 and BoxCox show the average result in both of cases.

4.4.2 Statistical Properties of Estimators

Based on *thinining algorithm* [30] for random samples, we generate $m = 1,000$ random samples. However, since the resulting estimate is calculated from a fixed sample of failure time data $t_1, t_2, \cdots, t_n$, we cannot correspond to unknown failure patterns in the future, and cannot consider the uncertainty of the estimator as a random variable. For such a problem, it is well known that the interval estimation may work better to make the valid decision under uncertainty. Let $\hat{\tau}^*_{(k)}$ be each estimate of the software fault during time τ^* which is calculated by using thinining algorithim samples, where $k = 1, 2, \cdots, m$. Also, we label $\hat{\tau}^*_{(m/2)}$ and $\bar{\tau}^* = (\sum_{k=1}^{m} \hat{\tau}^*_{(k)})/m$ as the median and the mean, respectively. The variance V, skewness S and kurtosis K of estimators of the cumulative number of software faults are defined by the following equations:

$$V_{\tau^*} = \frac{\sum_{k=1}^{m-1} (\hat{\tau}^*_{(k)} - \bar{\tau}^*)^2}{m-1}, \tag{4.12}$$

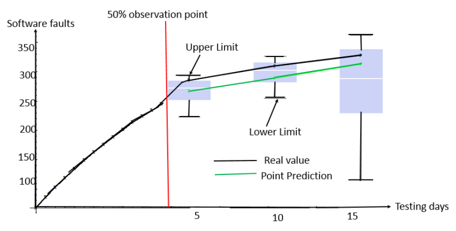
$$S_{\tau^*} = \frac{\sum_{k=1}^{m} (\hat{\tau}^*_{(k)} - \bar{\tau}^*)^3}{m V_{\tau^*}^{\frac{3}{2}}}, \tag{4.13}$$

$$K_{\tau^*} = \frac{\sum_{k=1}^{m} (\hat{\tau}^*_{(k)} - \bar{\tau}^*)^4}{m V_{\tau^*}^{2}}. \tag{4.14}$$

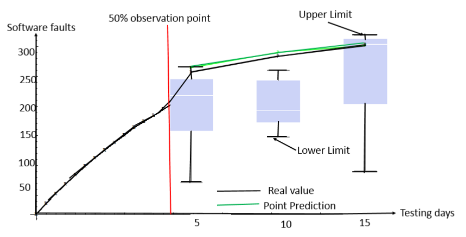
Based on the estimator distributions, we derive variance, skewness and kurtosis of the cumulative number of software faults. It is well known that if the skewness is closed to 0.0 and the kurtosis is closed to 3.0, then we can regard the estimator distribution as normal distribution approximately. Tables 4.7 present the statistics of estimators of the cumulative number of software fault, when the number of minimal testing days is 30.

Table 4.5: Case1∼ Case4.

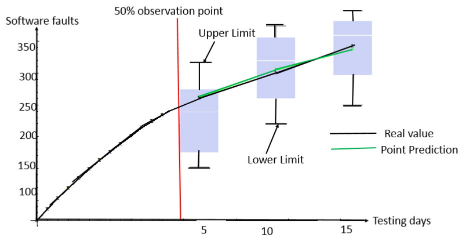
50% observation point						
Case1						
	Delta			Simulation		
Data_T	PICP	MPIW	PINAW	PICP	MPIW	PINAW
AT1	0.9632	157.95	10.51	0.9536	170.71	11.38
AT2	0.9482	229.02	15.27	0.9621	137.13	9.14
FT	0.9549	107.91	7.19	0.9429	106.47	7.09
BT	0.9521	113.71	7.59	0.9587	52.08	3.47
BoxCox	0.9479	183.75	12.25	0.9425	96.96	6.46
Normal	0.9189	46.30	3.08	0.9349	43.13	2.88
Case2						
AT1	0.9592	197.62	13.17	0.9693	192.51	12.83
AT2	0.9653	196.38	13.09	0.9589	167.48	11.16
FT	0.9617	141.03	9.40	0.9623	124.48	8.29
BT	0.9551	157.57	10.51	0.9593	125.90	8.39
BoxCox	0.9546	177.88	11.85	0.9459	167.62	11.17
Normal	0.9263	111.95	7.46	0.9129	135.25	9.017
Case3						
AT1	0.9512	142.94	9.53	0.9521	129.42	8.63
AT2	0.9641	198.22	13.21	0.9523	151.75	10.12
FT	0.9581	204.45	13.63	0.9557	188.67	12.58
BT	0.9547	75.34	5.02	0.9562	101.44	6.76
BoxCox	0.9596	143.83	9.58	0.9459	76.87	5.12
Normal	0.9154	51.62	3.44	0.9591	92.24	6.15
Case4						
AT1	0.9589	125.07	8.34	0.9683	115.78	7.72
AT2	0.9492	156.76	10.45	0.9614	121.47	8.09
FT	0.9859	193.94	12.93	0.9519	158.89	10.59
BT	0.9459	179.42	11.96	0.9598	129.89	8.65
BoxCox	0.9587	157.64	10.51	0.9241	111.47	7.43
Normal	0.9459	125.45	8.36	0.9137	89.78	5.99



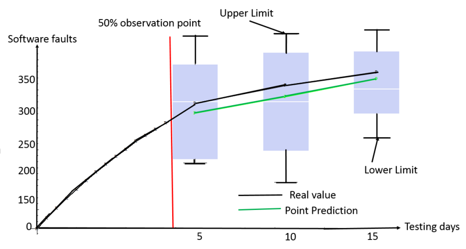
(a) AT1 for 15 output neuron



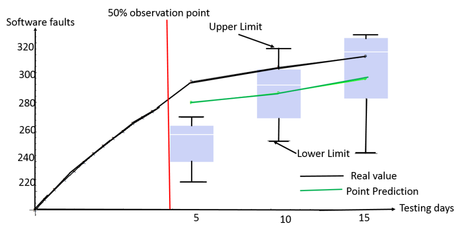
(b) AT2 for15 output neuron



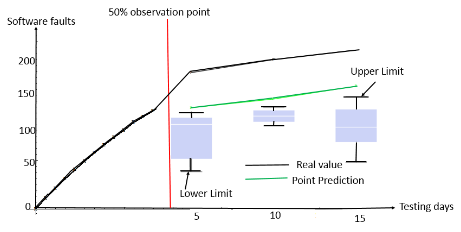
(c) FT for 15 output neuron



(d) BT for 15 output neuron



(e) BoxCox for 15 output neuron

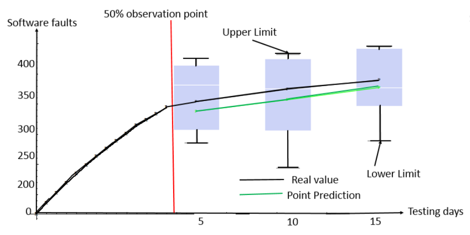


(f) Normal for 15 output neuron

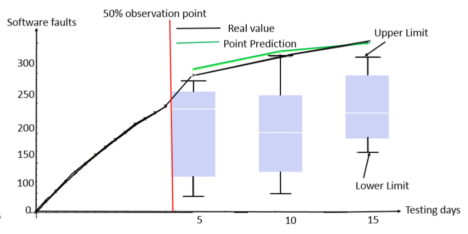
Figure 4.6: Prediction Interval for delta method with Case1.

Table 4.6: Real Predictive Interval.

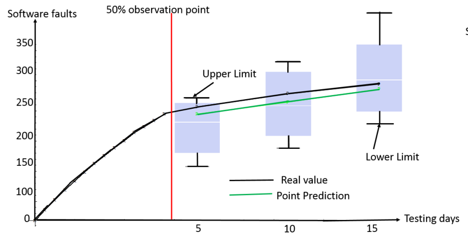
50% observation point			
Real PI			
Dataset	PICP	MPIW	PINAW
Case1	0.9685	96.6	6.41
Case2	0.9496	85.63	5.71
Case3	0.9563	121.45	8.09
Case4	0.9512	123.51	8.23



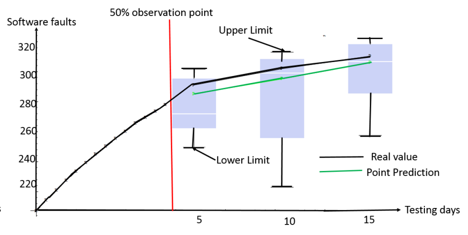
(a) AT1 for 15 output neuron



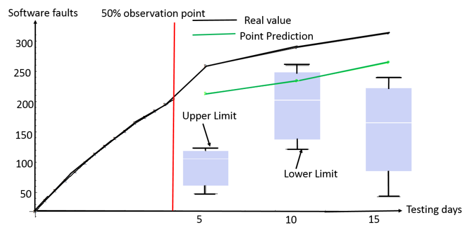
(b) AT2 for 15 output neuron



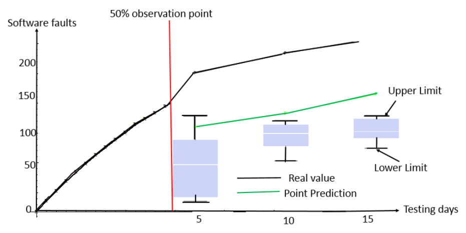
(c) FT for 15 output neuron



(d) BT for 15 output neuron

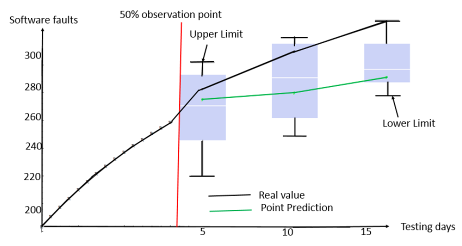


(e) BoxCox for 15 output neuron

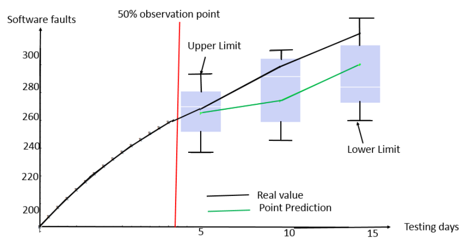


(f) Normal for 15 output neuron

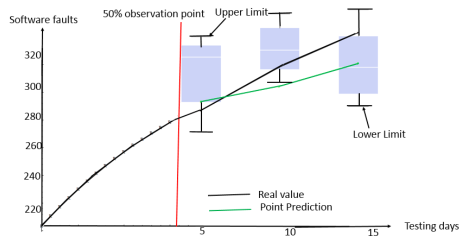
Figure 4.7: Prediction Interval for Simulation based method with Case1.



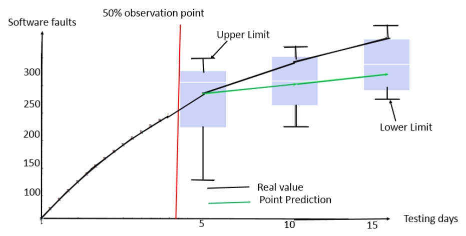
(a) AT1 for 15 output neuron



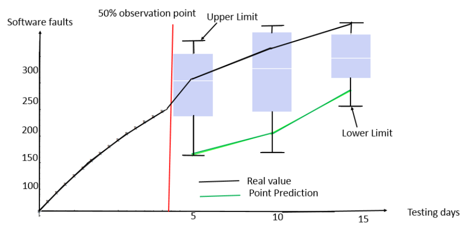
(b) AT2 for15 output neuron



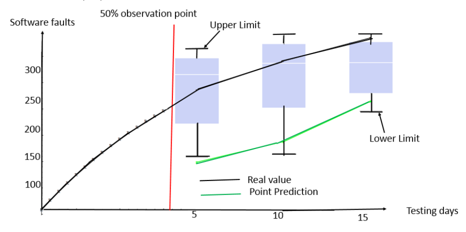
(c) FT for 15 output neuron



(d) BT for 15 output neuron

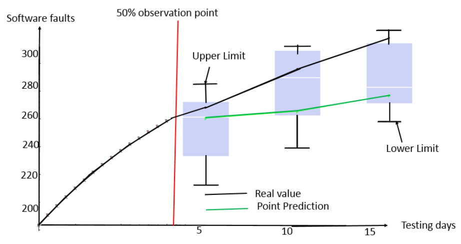


(e) BoxCox for 15 output neuron

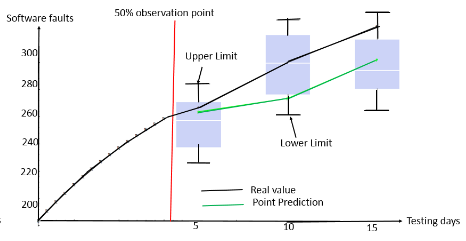


(f) Normal for 15 output neuron

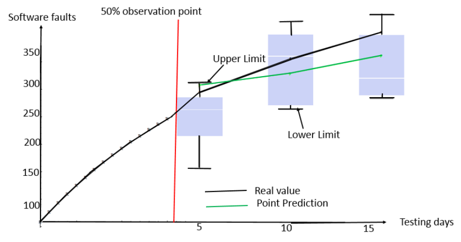
Figure 4.8: Prediction Interval for delta method with Case2.



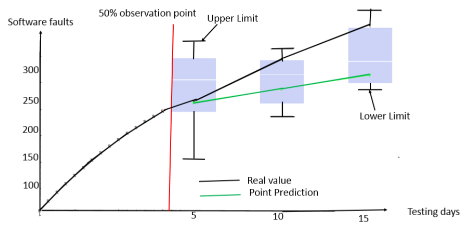
(a) AT1 for 15 output neuron



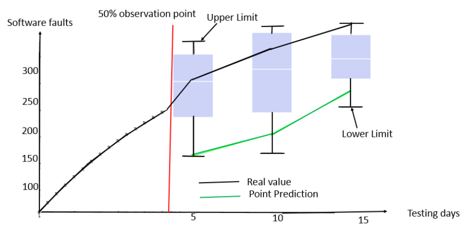
(b) AT2 for15 output neuron



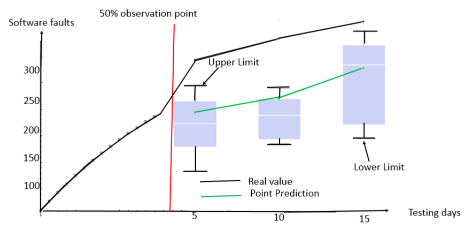
(c) FT for 15 output neuron



(d) BT for 15 output neuron

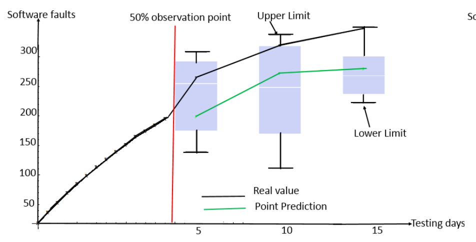


(e) BoxCox for 15 output neuron

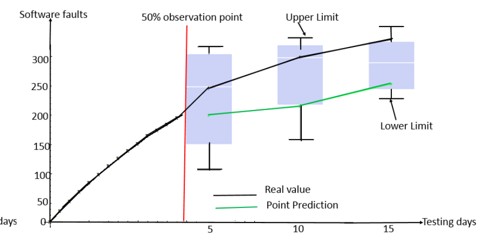


(f) Normal for 15 output neuron

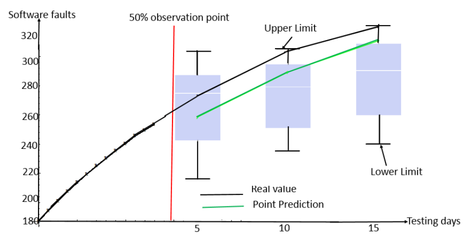
Figure 4.9: Prediction Interval for simulation based method with Case2.



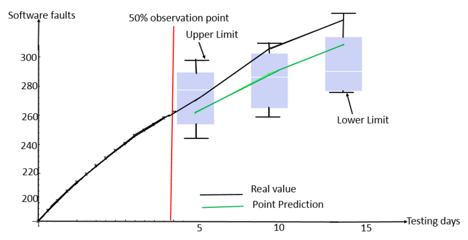
(a) AT1 for 15 output neuron



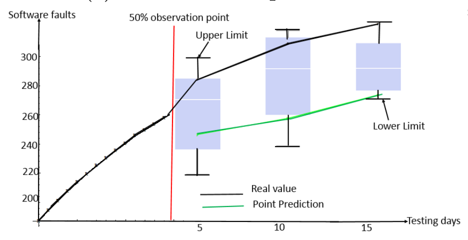
(b) AT2 for15 output neuron



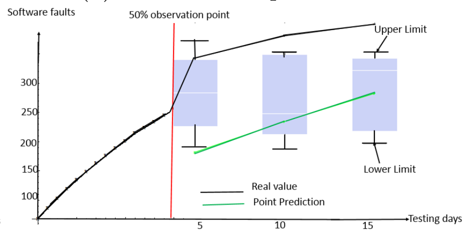
(c) FT for 15 output neuron



(d) BT for 15 output neuron

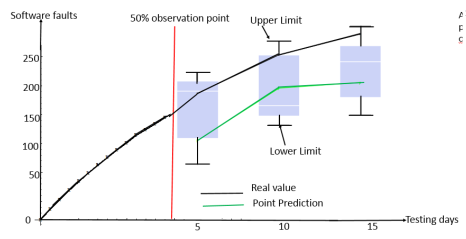


(e) BoxCox for 15 output neuron

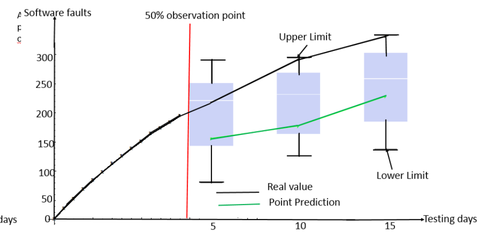


(f) Normal for 15 output neuron

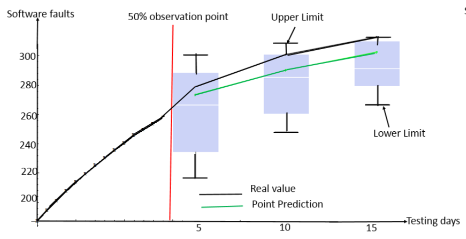
Figure 4.10: Prediction Interval for delta method with Case3.



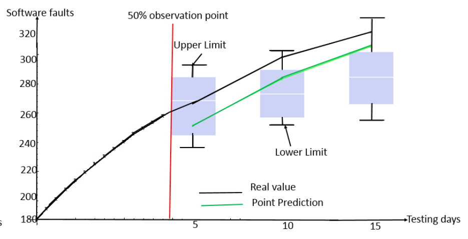
(a) AT1 for 15 output neuron



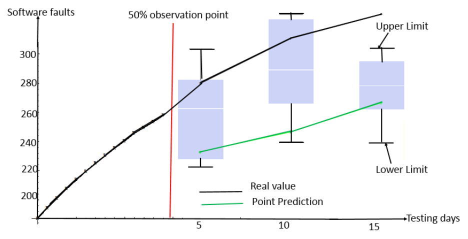
(b) AT2 for15 output neuron



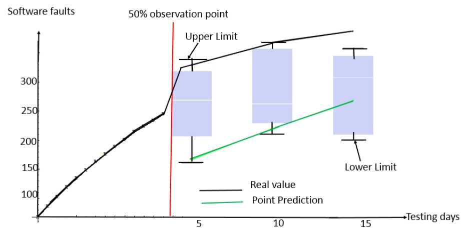
(c) FT for 15 output neuron



(d) BT for 15 output neuron

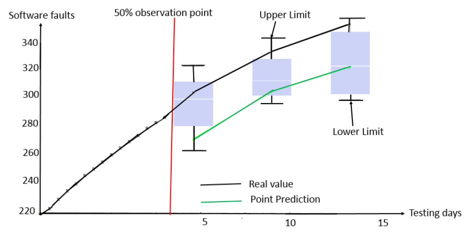


(e) BoxCox for 15 output neuron

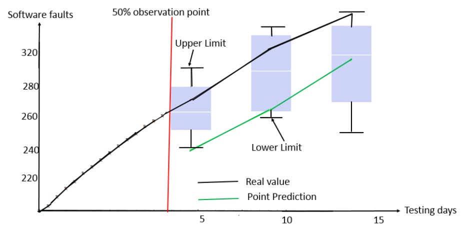


(f) Normal for 15 output neuron

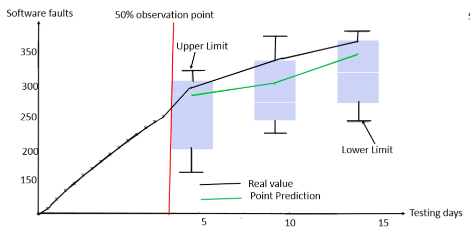
Figure 4.11: Prediction Interval for simulation based method with Case3.



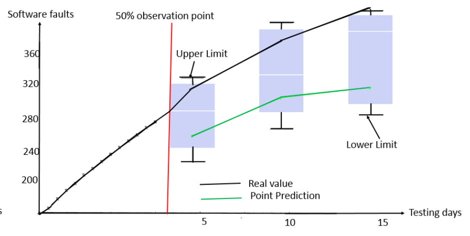
(a) AT1 for 15 output neuron



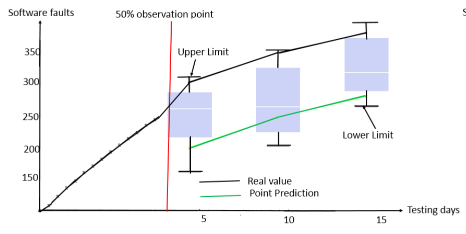
(b) AT2 for15 output neuron



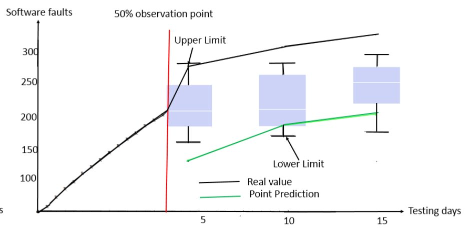
(c) FT for 15 output neuron



(d) BT for 15 output neuron

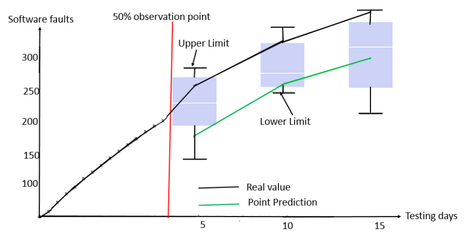


(e) BoxCox for 15 output neuron

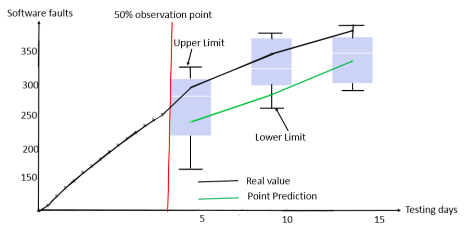


(f) Normal for 15 output neuron

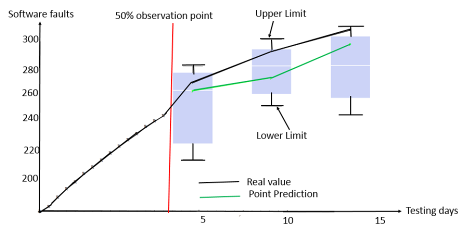
Figure 4.12: Prediction Interval for delta method with Case4.



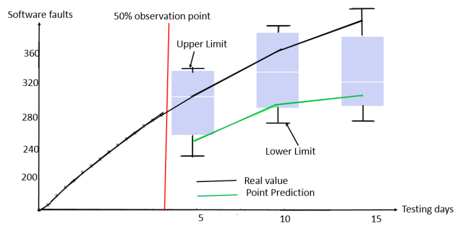
(a) AT1 for 15 output neuron



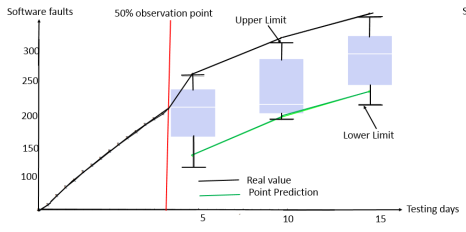
(b) AT2 for15 output neuron



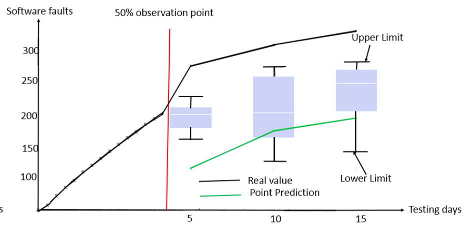
(c) FT for 15 output neuron



(d) BT for 15 output neuron

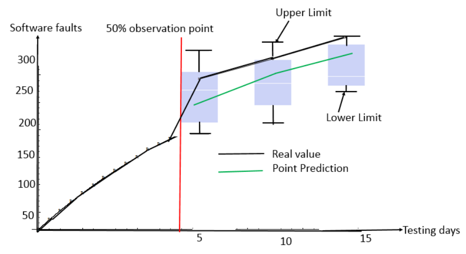


(e) BoxCox for 15 output neuron

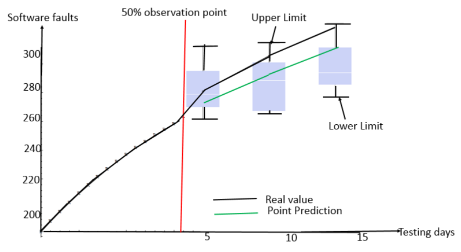


(f) Normal for 15 output neuron

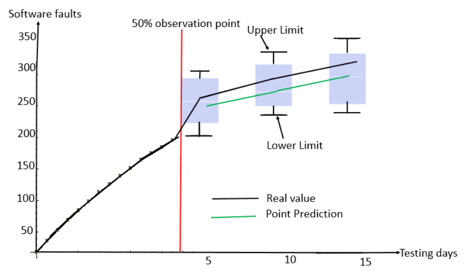
Figure 4.13: Prediction Interval for simulation based method with Case4.



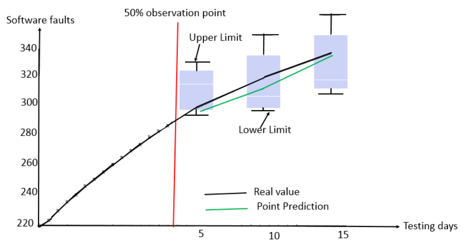
(a) Case1



(b) Case2



(c) Case3



(d) Case4

Figure 4.14: Real predictive interval with Case1 $\sim$ Case4

Table 4.7: Statistics of estimators of the software cumulative number of fault data.

Testing_D	Mean	Variance	Skewness	Kurtosis
1	18.30	1.18	0.28	4.13
2	36.06	1.34	-0.38	5.123
3	53.02	1.57	-1.53	15.28
4	69.22	1.81	-2.63	22.21
5	84.69	2.07	-3.47	25.09
6	99.47	2.17	-4.00	33.59
7	113.58	2.50	-5.37	33.71
8	127.05	3.17	-8.82	33.83
9	139.92	3.23	-9.48	38.83
10	152.18	4.53	-9.41	62.46
11	163.87	7.07	-10.44	65.71
12	175.09	6.62	-10.35	75.72
13	185.81	6.06	-10.23	86.52
14	195.98	9.08	-10.89	103.58
15	205.72	10.81	-11.23	115.97
16	215.07	9.99	-11.09	119.60
17	223.97	9.08	-10.91	123.57
18	232.49	8.11	-10.66	131.48
19	240.63	7.11	-10.34	133.33
20	253.92	5.66	-10.38	134.30
21	255.82	5.11	-9.32	134.62
22	262.52	4.19	-8.51	135.27
23	269.68	3.37	-7.37	136.36
24	276.15	2.66	-5.78	140.03
25	282.33	2.10	-3.59	142.85
26	288.24	1.72	-086	145.01
27	293.88	1.54	2.04	148.44
28	299.27	1.60	4.61	169.71
29	304.42	1.91	6.65	175.01
30	313.00	9.54	2.27	186.28

Chapter 5

Optimal Software Release Decision : Refined NN Approach

In this chapter, we present a novel method to estimate the optimal software testing time which minimizes the relevant expected software cost via a refined neural network approach with the grouped data, where the multi-stage look-ahead prediction is carried out with a simple three-layer perceptron neural network with multiple outputs. To analyze the software fault count data which follow a Poisson process with unknown mean value function, we transform the underlying Poisson count data to the Gaussian data by means of five data transformation methods, and predict the cost-optimal software release time via a neural network. In numerical examples with eight actual software fault count data, we compare our neural network approach with the conventional NHPP -based SRGMs. It is shown that our proposed method could provide a more accurate and more flexible decision making than the conventional stochastic modeling approach.

5.1 Point Estimation

5.1.1 NHPP-Based Software Reliability Modeling

Here we summarize the software reliability growth modeling. Suppose that software system test starts at time $t = 0$. Let $X(t)$ be the cumulative number of software faults detected by time t, where $\{X(t), t \geq 0\}$ denotes a stochastic (non-decreasing) point process in continuous time. In particular, it is said that

$X(t)$ is an NHPP if the following conditions hold:

- $X(0) = 0$,
- $X(t)$ has independent increments,
- $\Pr\{X(t+h) - X(t) \geq 2\}=o(h)$,
- $\Pr\{X(t+h) - X(t) = 1\}= \lambda(t;\boldsymbol{\theta})h + o(h)$,

where $o(h)$ is the higher term of infinitesimal time h, and $\lambda(t;\boldsymbol{\theta})$ is the intensity function of an NHPP which denotes the instantaneous fault detection rate per each fault. In the above definition, $\boldsymbol{\theta}$ is the model parameter (vector) included in the intensity function. Then, the probability that the cumulative number of software faults detected by time t equals x is given by

$$\Pr\{X(t) = x\} = \frac{\{\Lambda(t;\boldsymbol{\theta})\}^x}{x!} \exp\{-\Lambda(t;\boldsymbol{\theta})\}, \tag{5.1}$$

where

$$\Lambda(t;\boldsymbol{\theta}) = \int_0^t \lambda(x;\boldsymbol{\theta})dx \tag{5.2}$$

is called the mean value function and indicates the expected cumulative number of software faults up to time t, say, $\Lambda(t;\boldsymbol{\theta})$ =E$[X(t)]$.

If the mean value function $\Lambda(t;\boldsymbol{\theta})$ or the intensity function $\lambda(t;\boldsymbol{\theta})$ is specified, then the identification problem of the NHPP is reduced to a statistical estimation problem of unknown model parameter $\boldsymbol{\theta}$. In this way, when the parametric form of the mean value function or the intensity function is given, the resulting NHPP-based SRGMs are called parametric NHPP-based SRGMs. Table 1 contains the representative NHPP-based SRGMs and their mean value functions. Okamura and Dohi [41] summarized these eleven parametric NHPP-based SRGMs (See Chapter 3 Table 3.1) and developed a parameter estimation tool, SRATS , based on the maximum likelihood method, and the EM algorithm. In SRATS, the best SRGM with the smallest AIC is automatically selected, so the resulting best SRGM can fit best the past data on software fault counts among the eleven models.

Suppose that n realizations of $X(t_i)$, x_i $(i = 1, 2, \ldots, n)$, are observed up to the observation point t $(\geq t_n)$. We estimate the model parameter $\boldsymbol{\theta}$ by means of the maximum likelihood method. Then, the log likelihood function for the grouped data (t_i, x_i) $(i = 1, 2, \ldots, n)$ is given by

$$\begin{aligned} LLF(\boldsymbol{\theta}) &= \sum_{i=1}^{n} \Big((x_i - x_{i-1}) \log\big\{\Lambda(t_i; \boldsymbol{\theta}) - \Lambda(t_{i-1}; \boldsymbol{\theta})\big\} \\ &\qquad - \log\big\{(x_i - x_{i-1})!\big\}\Big) - \Lambda(t_n; \boldsymbol{\theta}), \end{aligned} \tag{5.3}$$

where $\Lambda(0; \boldsymbol{\theta}) = 0$, $x_0 = 0$ and $t = t_n$ for simplification. The maximum likelihood estimate of model parameter, $\hat{\boldsymbol{\theta}}$, can be obtained by maximizing Eq. (5.3) with respect to the model parameter $\boldsymbol{\theta}$. Once the model parameter is estimated, our next concern is to predict the future value of the intensity function or the mean value function at an arbitrary time t_{n+l} $(l = 1, 2, \ldots)$, where l denotes the prediction length. In parametric modeling, the prediction at time t_{n+l} is easily done by substituting estimated model parameter $\hat{\boldsymbol{\theta}}$ into the time evolution $\Lambda(t; \boldsymbol{\theta})$, where the unconditional and conditional mean value functions at an arbitrary future time t_{n+l} are given by

$$\Lambda(t_{n+l}; \hat{\boldsymbol{\theta}}) = \int_0^{t_{n+l}} \lambda(x; \hat{\boldsymbol{\theta}}) dx, \tag{5.4}$$

$$\begin{aligned} \Lambda(t_{n+l} | X(t_n) = x_n; \hat{\boldsymbol{\theta}}) &= x_n + \int_{t_n}^{t_{n+l}} \lambda(x; \hat{\boldsymbol{\theta}}) dx \\ &= x_n + \Lambda(t_{n+l}; \hat{\boldsymbol{\theta}}) - \Lambda(t_n; \hat{\boldsymbol{\theta}}). \end{aligned} \tag{5.5}$$

When the mean value function is unknown, on the other hand, a few non-parametric approaches have been developed by Kaneishi and Dohi [62] and Saito and Dohi [47]. However, it should be noted that those approaches can deal with the fault -detection time data, but do not work for prediction in future. The wavelet-based method by Xiao and Dohi [54] can treat the grouped data, but fails to make the long-term prediction in nature. In the following section, we use an elementary MIMO type of MLP for the purpose of the long-term software fault prediction.

5.1.2 Refined NN approach

ANN is a computational model inspired by the structure and functions of biological neural networks. An ANN has several advantages but one of the most significant ones is that it can train from past observation. The simplest ANN has three layers that are interconnected. The first layer consists of input neurons. Those neurons send data to the second layer i.e. hidden layer, which sends the outputs to the third layer. Subsequently, the hidden neurons have no communication with the external world, so that the output layer of neurons sends the final output to the external world. The problem is how to get an appropriate number of hidden neurons in the neural computation. In here we consider a MIMO neural network with only one hidden layer. Similar to Section 5.1.1, suppose that n software fault count data (t_i, x_i) $(i = 1, 2, \ldots, n)$ are observed at the observation point t $(= t_n)$. Our concern is about the future prediction of the cumulative number of software faults at time t_{n+l} $(l = 1, 2, \ldots)$.

5.1.2.1 *First phase: Data transformation*

The simplest MLP with only one output neuron is considered as a nonlinear regression model, where the explanatory variables are randomized by the Gaussian white noise. In particular, the output data in the MLP are indirectly assumed to be realizations of a nonlinear Gaussian model. By contrast, the fault count data are integer values. Hence, the original data shall be transformed to the Gaussian data in advance. Such a pre-data processing is common in the wavelet shrinkage estimation [54], but has not been considered in the software fault prediction via the ANN. According to the idea by Xiao and Dohi [54], used a pre-data processing which is common in the wavelet shrinkage estimation, where the underlying data follows the Poisson data. In the same way, we apply BT [5], AT1 [4], AT2[59], FT [12], and BoxCox power transformation [8] as the most major normalizing and variance-stabilizing transforms. Table 3.2 and and Section 1.4 summarizes the data transform techniques used and their inverse transform formulae. In the table, x_i denotes the cumulative number of software faults detected at i $(i = 1,\ 2, \ldots, n)$-th testing day. Then, we have the transformed data $\tilde{x}_i$ by means of any data transformation method. Let $\tilde{x}_i$ $(i = 1, 2, \ldots, n)$ and $\tilde{x}_{n+l}$ $(l = 1, 2, \ldots)$ be the input and output for the MIMO neural network, respectively. Then, the prediction of the cumulative number of software faults is given by the inversion of the data transform. Figure 5.1 depicts the architecture of back propagation type MIMO, where n is the number of software fault count data experienced before the observation point t_n

and l is the prediction length. We suppose that there is only one hidden layer with k $(= 1, 2, \ldots)$ hidden neurons in our MIMO neural network.

5.1.2.2 *Second Phase: Training to the neural network*

In Figure 5.1, n denotes input neuron number in the input layer, k is the hidden neuron and l indicates output neuron so we assume that all the connection weights (nk weights from input to hidden layer, kl weights from hidden to output layer) are first given by the uniformly distributed pseudo random varietes. In our MIMO type of MLP, output neuron can be calculated $(\tilde{x}_{n+1}, \ldots, \tilde{x}_{n+l})$ from the previous transformed input $(\tilde{x}_1, \ldots, \tilde{x}_n)$ if these weights are completely known. But, in principle of the common BP algorithm, it is impossible to train all the weights including $k(n+l)$ unknown patterns, as a result, it is required to improve the common BP algorithm for long-term prediction scheme.

- **Long-term prediction scheme:** In Figure 5.2, we show the structure of our prediction scheme. For the long-term prediction, we assume that $n > l$ without any loss of generality. In order to predict the cumulative number of software faults for l testing days from the observation point t_n, the prediction has to be made at the point t_{n-l}. This implies that only $(n - l)k + kl = nl$ weights can be estimated with the training data experienced for the period $(t_{n-l}, t_n]$ and that the remaining $k(n + l) - nl$ weights are not trained at time t_n. We call these $k(n + l) - nl$ weights the *non-estimable* weights. As the prediction length is longer, the number of non-estimable weights becomes greater and the prediction uncertainty also increases more. In this scheme, the transformed data $(\tilde{x}_1, \ldots, \tilde{x}_{n-l})$ are used for the input in the MIMO, and the remaining data $(\tilde{x}_{n-l+1}, \ldots, \tilde{x}_n)$ are used for the teaching signals in the training phase.

- **Common BP algorithm:** Back propagation (BP) is a common method for training a neural network. The BP algorithm is the well-known gradient descent method to update the connection weights, to minimize the squared error between the network output values and the teaching signals. There are two special inputs; bias units which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. For the value coming out of an input neuron, $\tilde{x}_i$ ($i = 1,\ 2,\ \ldots,\ n-l$), it is common to add two special inputs; bias units which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. Let $w_{ij} \in [-1,1]$ be the connection weight from i-th input neuron to j-th hidden neuron, where w_{0j} and w'_{0s} denote the bias weights for j-th hidden neuron and s-th output neuron, respectively for the training phase with $i = 0,1,\ldots,n-l$, $j = 0,1,\ldots,k$ and $s = n-l+1, n-l+2, \ldots, n$. Each hidden neuron calculates the weighted sum of the input neuron, h_j, in the following equation:

$$h_j = \sum_{i=1}^{n-l} \tilde{x}_{ij} w_{ij} + w_{0j}. \tag{5.6}$$

Since there is no universal method to determine the number of hidden neurons, we change k in the pre-experiments and choose an appropriate value. After calculating h_j for each j , we apply a sigmoid function $f(h_j) = 1/\exp(-h_j)$ as a threshold function in the MIMO. Since h_j are summative and weighted inputs from respective hidden neuron, the s-th output $s = n-l+1, n-l+2, \ldots, n$ in the output layer is given by

$$\tilde{x}_s = \sum_{j=1}^{k} f(h_j) w'_{js} + w'_{0s}. \tag{5.7}$$

Because $\tilde{x}_s$ are also summative and weighted inputs from respective hidden neuron in the output layer, the weight w'_{js} is connected from j-th hidden neuron to s-th output neuron. The output value of the network in the training phase, $\tilde{x}_s$, is calculated by $f(\tilde{x}_s) = 1/\exp(-\tilde{x}_s)$. In the BP algorithm, the error is propagated from an output layer to a successive hidden layer by updating the weights, where the error function is defined by

$$\mathrm{SSE} = \frac{\sum_{s=n-l+1}^{n} (\tilde{x}_s^o - \tilde{x}_s)^2}{(l-1)} \tag{5.8}$$

with the prediction value $\tilde{x}_s$ and the teaching signal $\tilde{x}_s^o$ observed for the period $(t_{n-l+1}, t_n]$.

Next we quickly overview the BP algorithm. It updates the weight parameters so as to minimize SSE between the network output values $\tilde{x}_s$ ($s =$

$n-l+1, n-l+2, \ldots, n$) and the teaching signals $\tilde{x}_s^o$, where each connection weight is adjusted using the gradient descent according to the contribution to SSE in Eq. 5.8. The momentum, α, and the learning rate, η, are controlled to adjust the weights and the convergence speed in the BP algorithm, respectively. Since these are the most important turning parameters in the BP algorithm, we carefully examine these parameters in pre-experiments. Then, the connection weights are updated in the following:

$$\begin{aligned} w_{ij(new)} &= w_{ij} + \alpha w_{ij} + \eta \delta h_j \tilde{x}_i \\ &\qquad (i = 1, 2, \ldots, n-l, j = 1, \ldots, k), \qquad (5.9) \\ w'_{js(new)} &= w'_{js} + \alpha w'_{js} + \eta \delta \tilde{x}_s \tilde{x}_s \\ &\qquad (j = 1, 2, \ldots, k, s = n-l+1, \ldots, n), \qquad (5.10) \end{aligned}$$

where δh_j and $\delta \tilde{x}_s$ are the output gradient of j-th hidden neuron and the output gradient in the output layer, and are defined by

$$\begin{aligned} \delta h_j &= f(h_j)(1 - f(h_j)), \qquad (5.11) \\ \delta \tilde{x}_s &= \tilde{x}_s(1 - \tilde{x}_s)(\tilde{x}_s^o - \tilde{x}_s), \qquad (5.12) \end{aligned}$$

respectively. Also, the updated bias weights for hidden and output neurons are respectively given by

$$\begin{aligned} w_{0j(new)} &= w_{0j} + \alpha w_{0j} + \eta \delta h_j, \qquad (5.13) \\ w'_{0s(new)} &= w'_{0s} + \alpha w'_{0s} + \eta \delta \tilde{x}_s. \qquad (5.14) \end{aligned}$$

The above procedure is repeated until the desired output is achieved.

5.1.2.3 *Last Phase: Long-term prediction*

Once the nl weights are estimated with the training data experienced for the period $(t_{n-l}, t_n]$ through the BP algorithm, we need to obtain the remaining $k(n+l) - nl$ non-estimable weights for prediction. Unfortunately, since these

Table 5.1: Posteriori optimal software testing time for eight datasets.

	t_0^*							
c_2	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8
2	50	28	33	59	49	64	45	74
3	51	35	37	89	61	73	69	114
4	51	35	44	89	61	73	70	114
5	54	36	44	89	61	73	70	114
6	56	36	44	89	61	73	70	114
7	62	36	44	89	61	73	70	114
8	62	36	44	89	61	73	75	114
9	62	36	44	89	61	73	75	114
10	62	36	44	89	61	73	75	114

cannot be trained with the information at time t_n, we need to give these values by the uniform pseudo random varieties ranged in $[-1, 1]$. By giving the random connection weights, the output as the prediction of the cumulative number of software faults, $(\tilde{x}_{n+1}, \dots, \tilde{x}_{n+l})$, are calculated by replacing Eqs.5.6 and 5.7 by

$$h_{j(new)} = \sum_{i=1}^{n} \tilde{x}_{ij} w_{ij(new)} + w_{0j(new)}, \tag{5.15}$$

$$\tilde{x}_{n+s} = \sum_{j=1}^{k} f(h_{j(new)}) w'_{js(new)} + w'_{0s(new)}, \tag{5.16}$$

respectively, for $i = 1, 2, \dots, n$, $j = 1, 2, \dots, k$ and $s = 1, 2, \dots, l$. Note that the resulting output is based on one sample by generating the uniform pseudo random variates only once. In order to obtain the prediction of the expected cumulative number of software faults, we generate m sets of random variates and take the arithmetic mean of the m predictions of $(\tilde{x}_{n+1}, \dots, \tilde{x}_{n+l})$, where $m = 1,000$ is confirmed to be enough in our preliminary experiments. In other words, the prediction in the MIMO neural network is reduced to a combination of the BL learning and a Monte Carlo simulation on the connection weights.

5.2 Optimal Software Testing Policy

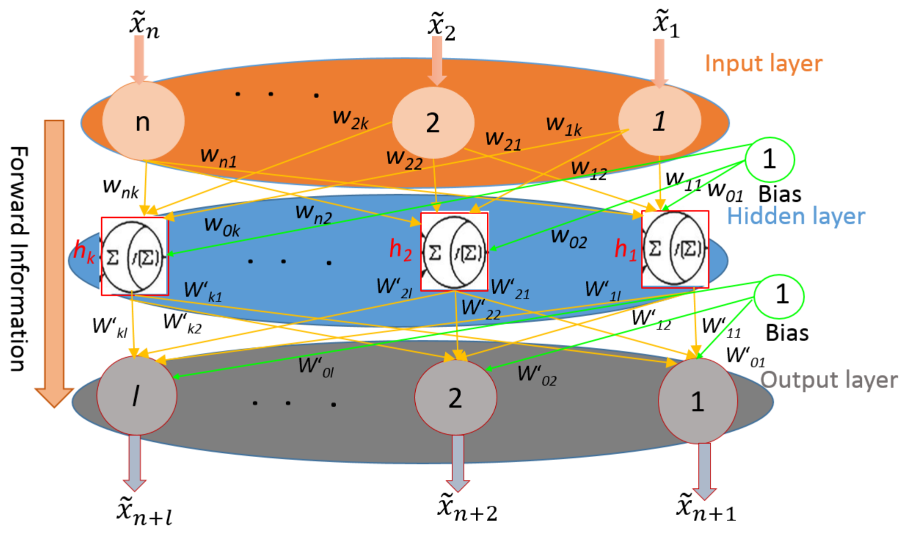


Figure 5.1: Architecture of back propagation type MIMO.

Suppose that the system test of a software product starts at $t = 0$ and terminates at $t = t_0$. Let T_L be the software lifetime or the upper limit of the software warranty period, where the time length $(t_0, T_L]$ denotes the operational period of software. When the software fault count data $\chi_n = \{x_1, \ldots, x_n\}$ which are the cumulative number of detected faults at time t_i $(i = 1, 2, \ldots, n)$ are observed at time t_n $(0 < t_n \le t_0)$, the model parameter $\hat{\boldsymbol{\theta}}$ is estimated with these data in parametric NHPP-based SRGMs. Define the following cost components:

- c_0 (> 0): testing cost per unit system testing time,
- c_1 (> 0): debugging cost per fault in system testing phase,
- c_2 $(> c_1)$: debugging cost per fault in operational phase.

Based on the above cost parameters, the expected total software cost is given by

$$C(t_0; t_n, \hat{\boldsymbol{\theta}}) = c_0 t_0 + c_1 \Big\{ x_n + \Lambda(t_0; \hat{\boldsymbol{\theta}}) - \Lambda(t_n; \hat{\boldsymbol{\theta}}) \Big\} + c_2 \Big\{ \Lambda(T_L; \hat{\boldsymbol{\theta}}) - \Lambda(t_0; \hat{\boldsymbol{\theta}}) \Big\}. \tag{5.17}$$

Note that $\Lambda(T_L; \hat{\boldsymbol{\theta}})$ is independent of t_0 and that $\Lambda(t_n; \hat{\boldsymbol{\theta}}) = x_n$ from Eq.(4). When $t_0 = t_{n+l}$, l corresponds to the remaining testing length. Hence, our

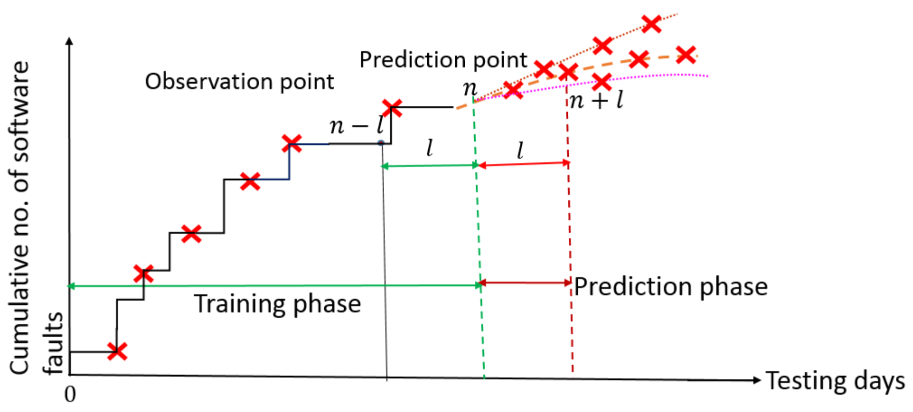


Figure 5.2: Configuration of prediction scheme via MIMO.

optimization problem is essentially reduced to

$$\begin{aligned}\min_{t_n \le t_0 \le T_L} C(t_0; T, \hat{\boldsymbol{\theta}}) &\Longleftrightarrow \min_{t_n \le t_0 \le T_L} \left\{ c_0 t_0 - (c_2 - c_1)\Lambda(t_0; \hat{\boldsymbol{\theta}}) \right\} \\ &\Longleftrightarrow \max_{t_n \le t_0 \le T_L} \left\{ \Lambda(t_0; \hat{\boldsymbol{\theta}}) - \frac{c_2 - c_1}{c_0} t_0 \right\}. \end{aligned} \qquad (5.18)$$

When SRGMs are assumed, the mean value function $\Lambda(t_0; \hat{\boldsymbol{\theta}})$ can be estimated from the underlying data. On the other hand, in the context of neural computation, the output of MLP in Eq. 5.16 is regarded as an estimate of $\Lambda(t_0; \hat{\boldsymbol{\theta}})$. Hence, output sequence $(\tilde{x}_{n+1}, \tilde{x}_{n+2}, \ldots, \tilde{x}_{n+l})$ denotes ($\Lambda(t_{n+1}; \hat{\boldsymbol{\theta}})$, $\Lambda(t_{n+2}; \hat{\boldsymbol{\theta}})$, $\ldots$, $\Lambda(t_{n+l}; \hat{\boldsymbol{\theta}})$) in the MIMO type MLP. The essentially similar but somewhat different dual problem was given by Dohi et al. [11]. Consequently, the optimal testing time $\hat{t}_0$ is equivalent to the point with the maximum vertical distance between the predicted curve $\Lambda(t_0; \hat{\boldsymbol{\theta}})$ and a straight line $(c_2 - c_1)t_0/c_0$ in the two-dimensional plane. Then, there exists a finite optimal software release time. In fact, when $\hat{t}_0$ is equal to the observation point t_n, then it is optimal not to continue testing the software product any more. In other words, the optimization problem considered here is a prediction problem of the function $\Lambda(t_0; \hat{\boldsymbol{\theta}})$.

5.3 Numerical Illustrations

We give numerical examples to predict the optimal software release time based on the nonparametric inference approach, where eight data sets, DS1$\sim$ DS8, are used for analysis [29]. Table 3.3 in Chapter 3 shows the structure of the datasets observed in this study. To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent, where the initial guess of weights, w_{ij}, w'_{js}, w_{0j} and w'_{0s}, are given by the uniform random variates range $[-1, +1]$, the number of total iterations in the BP algorithm run is 1,000 and the convergence criteria on the minimum error is 0.001. In our experiments, it is shown that the search range of the transformation parameter λ should be $[-3, +2]$.

5.3.1 Predictive Performance

To evaluate the prediction performance of the optimal software testing time via MIMO type of MLP or SRGM, we define the error criterion in the following

$$\text{Prediction Error} = \left| \frac{\hat{t}_0 - t_0^*}{t_{0^*}} \right| \times 100, \tag{5.19}$$

where t_0^* is a posteriori optimal testing time and is calculated by finding the point l $(= 0, 1, 2, \ldots, q)$ which maximizes $\Lambda(t_{n+l}; \hat{\boldsymbol{\theta}}) - (c_2 - c_1)t_{n+l}/c_0$, and q is satisfies $t_{n+q} = T_L$. This error criterion is used for the predictive performance given all the data $x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{n+q}$. In Table 5.1 with the whole data (*i.e.*, 100%), we present posteriori optimal testing times for DS1 $\sim$ DS7, when $c_0 = 4$, $c_1 = 1$ and c_2 varies in the range of $2 \sim 10$ and for DS8 we change c_0 parameter, i.e $c_0 = 1$ other parameter is same as previous.

5.3.2 Results

Since the cost parameter c_2 is not sensitive, so we focus on only two cases; $c_2 = 4$ and $c_2 = 9$ hereafter. In Tables 5.2 $\sim$ 5.17 we give the prediction results on the optimal software testing time for DS1$\sim$ DS8 with $c_2 = 4$ and $c_2 = 9$, respectively, where the prediction of optimal testing time and its associated prediction error are calculated at each observation point (50% $\sim$ 90% point of the whole data). In these tables, the bold number implies the best prediction model than the best SRGM among eleven models and BoxCox shows the best λ. In the MIMO type of MLP, we compare five data transform methods with the non-transformed case (Normal) and the best SRGM. In the column of SRGM, we denote the best SRGMs in terms of predictive performance (in the sense of minimum average relative error; AE) and estimation performance (in the sense of minimum Akaike information criterion; AIC) by P and E, respectively, where AE denotes the capability of the prediction model and is defined by

$$AE_l = \frac{\sum_{s=1}^{l} RE_s}{l}, \tag{5.20}$$

where RE_s is called the relative error for the future time $t = n + s$ and is given by

$$RE_s = \left| \frac{\tilde{x}^o_{n+s} - \tilde{x}_{n+s}}{\tilde{x}^o_{n+s}} \right| \quad (s = 1, 2, \ldots, l). \tag{5.21}$$

So we regard the prediction model with smaller AE as a better prediction model. Figures 5.3 and 5.4 illustrate the behavior of cumulative number of software faults detected in the testing phase in DS1 and DS2, respectively. In Table 5.2 $\sim$ 5.17, it is seen that our MIMO-based approaches in almost all cases provide smaller prediction error than the common SRGM when the observation point is 50% $\sim$ 90% point. On the other hand, in the latter phase of software testing, *i.e.*, 80% $\sim$ 90% observation points, SRGMs, such as txvmax and txvmin, offer less error than the MIMO type of MLP (See Tables 5.4 $\sim$ 5.7). Even in these cases, it should be noted the best SRGM with the minimum AIC is not always equivalent to the best SRGM with the minimum AE. This fact tells that one cannot know exactly the best SRGM in terms of judgment on when to stop software testing. Similar to Tables 5.8$\sim$ 5.12, MIMO type of MLP gives the best timing of software system test in the all (middle and late) testing phase. In this experiment, it can be seen that the data transform does function well to predict the optimal software testing time than the non-transform case. In DS6 and DS8 all most all testing phase SRGM offer better result. Furthermore, MIMO with the non-transform (Normal) gives the best timing of software system test in that tables (see Tables 5.12 and 5.13, 5.16 and 5.17 respectively). In Table 5.1 we can see when we the cost parameter c_2 increases, the resulting software testing time

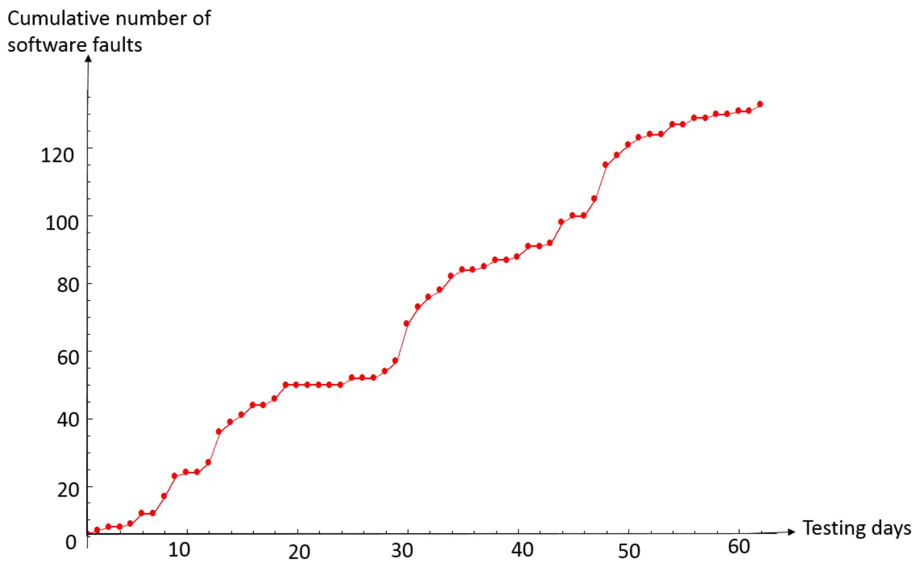


Figure 5.3: DS1.

also increases. Furthermore, it can be seen that relative error gets smaller as the data increases in the case of 50% ∼90% observation points. This means that the predictive optimal software release time can be updated by adding software fault data. Since 80% and 90% observation points have already exceeded the real optimal release time, the predictive optimal software release time of these two cases is equivalent to the observation point (see Table 5.16). Therefore, it can be said that our NN with BoxCox power transformation method works well to judge when to stop the software testing and release to the software.

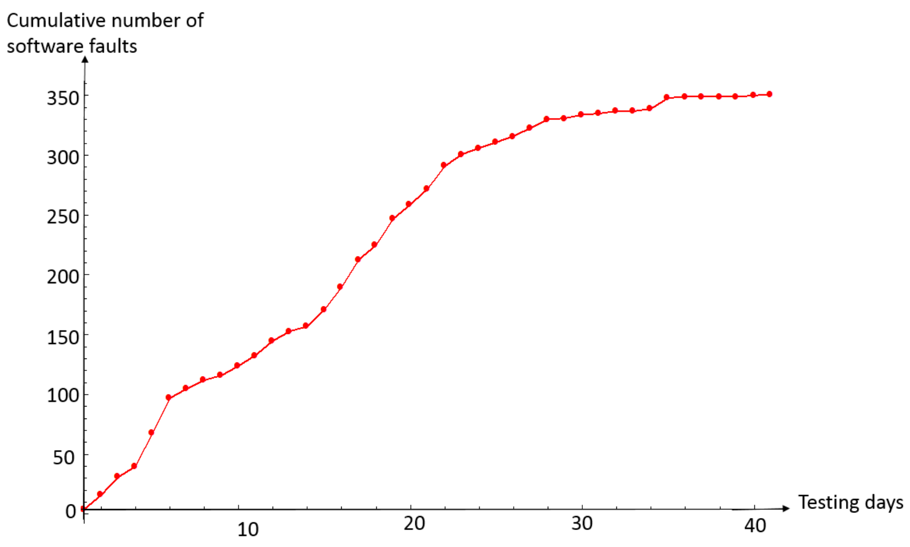


Figure 5.4: DS2.

Table 5.2: Prediction results of optimal software testing time with DS1 with $c_2 = 4$.

50% observation ($t_n = 31$)						
MIMO ($\hat{t}_0$)						Best SRGM ($\hat{t}_0$)
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	
49 (3.92%)	49 (3.92%)	49 (3.92%)	50 (1.96%)	51(0.7) **(0%)**	40 (21.56%)	P: pareto, 40 (21.56%) E: txvmax, 35 (31.37%)
60% observation ($t_n = 37$)						
52 (1.96%)	52 (1.96%)	51 **(0%)**	52 (1.96%)	51(1.5) **(0%)**	49 (3.92%)	P:lnorm, 47 (7.84%) E: txvmax, 38 (25.49%)
70% observation ($t_n = 43$)						
52 (1.96%)	52 (1.96%)	52 (1.96%)	52 (1.96%)	52(0.7) (1.96%)	52 (1.96%)	P: txvmax, 51 **(0%)**) E: txvmax, 51 **(0%)**)
80% observation ($t_n = 50$)						
52 (1.96%)	54 (5.88%)	56 (9.80%)	57 (11.76%)	54(0.6) (5.88%)	54 (5.88%)	P: txvmin, 51 **(0%)**) E: lxvmin, 52 (1.96%)
90% observation ($t_n = 56$)						
58 (13.72%)	57 **(11.76%)**	57 **(11.76%)**	57 **(11.76%)**	57(1.0) **(11.76%)**	57 **(11.76%)**	P: txvmax, 61 (19.6%) E: lxvmin, 61 (19.6%)

Table 5.3: Prediction results of optimal software testing time for DS1 with $c_2 = 9$.

50% observation ($t_n = 31$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
51 (**17.74%**))	51 (**17.74%**)	51 (**17.74)%**	51 (**17.74%**))	51 (0.2) (**17.74%**))	51 (**17.74)%**)	P: pareto, 40 (35.48%) E: txvmax, 35 (43.83%)
60% observation ($t_n = 37$)						
57 (**8.06%**))	56 (9.67%)	57 (**8.06%**))	57 (**8.06%**))	57(1.0) (**8.06%**))	56 (9.67%)	P: lnorm, 47 (24.19%) E: txvmax, 38 (38.71%)
70% observation ($t_n = 43$)						
58 (6.45%)	56 (9.67%)	58 (6.45%)	58 (6.45%)	61(0.4) (**1.61%**)	58 (6.45%)	P: txvmax, 51 (17.74%) E: txvmax, 51 (17.74%)
80% observation ($t_n = 50$)						
60 (3.22%)	56 (9.67%)	60 (3.22%)	60 (3.22%)	61(1.3) (**1.61%**))	60 (3.22%)	P:txvmin, 61 (**1.61%**)) E: lxvmin, 52 (16.13%)
90% observation ($t_n = 56$)						
61 (**1.61%**))	61 (**1.61%**))	61 (**1.61%**)	61 (**1.61%**))	61(0.7) (**1.61%**))	61 ((**1.61%**))	P: txvmax, 61 (**1.61%**) E: lxvmin, 57 (8.06%)

Table 5.4: Prediction of optimal software release time for DS2 with $c_2 = 4$.

50% observation ($t_n = 21$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
32 (8.57%)	26 (25.71%)	35 (**0%**)	32 (8.57%)	32(0.5) (8.57%)	30 (14.28%)	P: lxvmax, 26 (25.71%) E: lxvmin, 22 (37.14%)
60% observation ($t_n = 25$)						
34 (2.85%)	35 (**0%**))	35 (**0%**))	35 (**0%**))	35(1.7) (**0%**))	30 (14.28)	P:tlogist 30 (14.28%) E: lxvmin, 27 (22.85%)
70% observation ($t_n = 29$)						
38 (8.57%)	38 (8.57%)	38 (8.57%)	38 (8.57%)	35(1.0) (**0%**)	34 (2.85%)	P:lxvmin, 35 (**0%**)) E: lxvmin, 35 (**0%**)
80% observation ($t_n = 33$)						
39 (11.42%)	39 (11.42%)	38 (8.57%)	38 (8.57%)	35(0.8) (**0%**)	38 (8.57%)	P:lxvmin, 35 (**0%**) E: lxvmin, 35 (**0%**)
90% observation ($t_n = 37$)						
41 (17.14%)	40 (14.28%)	40 (14.28%)	39 (11.42%)	37(1.2) (**5.71%**)	41 (17.14%)	P: txvmax, 37 (**5.71%**) E: lxvmin, 39 (11.42%)

Table 5.5: Prediction of optimal software release time for DS2 with $c_2 = 9$.

50% observation ($t_n = 21$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
34	34	35	35	26(1.2)	35	P: lxvmax, 26 (27.87%)
(5.55%)	(5.55%)	**(2.77)%**	**(2.77)%**	(27.87%)	**(2.77)%**	E: lxvmin, 22 (38.89%)
60% observation ($t_n = 25$)						
35	35	35	35	36(0.3)	38	P: tlogist, 30 (16.67%)
(2.77%)	(2.77%)	(2.77%)	(2.77%)	**(0%)**	(5.55%)	E: lxvmin, 27 (25%)
70% observation ($t_n = 29$)						
36	36	38	38	38(1.0)	41	P: lxvmin, 35 (2.77%)
(**0%**)	(**0%**)	(5.55%)	(5.55%)	(5.55%)	(13.89%)	E: lxvmin, 35 (2.77%)
80% observation ($t_n = 33$)						
41	41	41	41	38(1.0)	41	P: lxvmin, 35 **(2.77)%**)
(13.89%)	(13.89%)	(13.89%)	(13.89%)	(5.55%)	(13.89%)	E: lxvmin, 35 **(2.77%)**)
90% observation ($t_n = 37$)						
41	41	41	41	41(0.1)	41	P: txvmax, 37 **(2.77)%**)
(13.89%)	(13.89%)	(13.89%)	(13.89%)	(13.89%)	(13.89%)	E: lxvmin, 39 (8.33%)

Table 5.6: Prediction results of optimal software testing time with DS3 with $c_2 = 4$.

50% observation ($t_n = 23$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
39	40	43	42	43(1.0)	40	P: exp, 42 (4.54%)
(11.36%)	(9.09%)	(**2.27%**)	(4.54%)	**(2.27%)**	(9.09%)	E: lnorm, 40 (9.09%)
60% observation ($t_n = 28$)						
39	40	44	42	43(2.0)	40	P: pareto, 46 (4.54%)
(11.36%)	(9.09%)	**(0%)**	(4.54%)	(2.27%)	(9.09%)	E: lnorm, 46 (4.54%)
70% observation ($t_n = 32$)						
39	40	44	42	43(-2.7)	40	P: txvmax, 44 **(0%)**
(11.36%)	(9.09%)	**(0%)**	(4.54%)	(2.27%)	(9.09%)	E: gamma, 39 (11.36%)
80% observation ($t_n = 37$)						
39	40	44	43	43(1.0)	40	P: exp, 44 **(0%)**
(11.36%)	(9.09%)	(**0%**)	(2.27%)	(2.27%)	(9.09%)	E: lxvmin, 42 (5.54%)
90% observation ($t_n = 41$)						
46	45	45	45	44(1.3)	42	P: txvmax, 44 **(0%)**
(4.54%)	(2.27%)	(2.27%)	(2.27%)	**(0%)**	(4.54%)	E: lxvmin, 44**(0%)**

Table 5.7: Prediction results of optimal software testing time with DS3 with $c_2 = 9$.

50% observation ($t_n = 23$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
42 (4.54%)	42 (4.54%)	44 (**0%**)	44 (**0%**)	44(1.8) **(0%)**	40 (9.09%)	P: exp, 44 **(0%)** E: lnorm, 40 (9.09%)
60% observation ($t_n = 28$)						
42 (4.54%)	39 (11.36%)	40 (9.09%)	41 (6.82%)	42(1.0) (4.54%)	39 (11.36%)	P: pareto, 46 (4.54%) E: lnorm, 46 (4.54%)
70% observation ($t_n = 32$)						
39 (11.36%)	40 (9.09%)	44 (**0%**)	42 (4.54%)	43(0.9) (2.27%)	40 (9.09%)	P: txvmax, 44 **(0%)** E: gamma, 39 (11.36%)
80% observation ($t_n = 37$)						
39 (11.36%)	40 (9.09%)	44 (**0%**)	43 (2.27%)	43(1.0) (2.27%)	40 (9.09%)	P: exp, 44 **(0%)** E: lxvmin, 42 (5.54%)
90% observation ($t_n = 41$)						
46 (4.54%)	45 (2.27%)	45 (2.27%)	45 (2.27%)	44(2.0) **(0%)**	42 (4.54%)	P: txvmax, 44 **(0%)** E: lxvmin, 44 **(0%)**

Table 5.8: Prediction results of optimal software testing time with DS4 with $c_2 = 4$.

50% observation ($t_n = 55$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
77 (13.48%)	67 (24.72%)	84 (5.62%)	89 **(0%)**	87(1.4) (2.24%)	64 (28.09%)	P: exp, 77 (13.48%) E: lxvmin, 109 (22.47%)
60% observation ($t_n = 64$)						
87 (2.24%)	87 (2.24%)	84 (9.09%)	89 **(0%)**	86(1.6) (4.54%)	69 (11.36%)	P: txvmax, 65 (4.54%) E: lxvmin, 109 (22.47%)
70% observation ($t_n = 76$)						
87 (2.24%)	87 (2.24%)	89 (**0%**)	89 **(0%)**	87(0.9) (2.24%)	69 (11.36%)	P: txvmax, 77 (13.48%) E: lxvmin, 109 (22.47%)
80% observation ($t_n = 87$)						
89 **(0%)**	87 (2.24%)	87 (2.24%)	87 (2.24%)	87(0.9) (2.24%)	109 (2.47%)	P: lxvmax, 88(1.12%) E: exp,109 (22.47%)
90% observation ($t_n = 98$)						
109 (22.47%)	109 (22.47%)	109 (22.47%)	98 **(10.11%)**	98(1.5) **(10.11%)**	109 (22.47%)	P: lxvmax, 99(11.23%) E: lxvmin, 109 (22.47%)

Table 5.9: Prediction results of optimal software testing time with DS4 with $c_2 = 9$.

MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
50% observation ($t_n = 55$)						
87 (13.48%)	80 (24.72%)	85 (5.62%)	89 **(0%)**	89(0.8) **(0%)**	70 (28.09%)	P: exp, 77 (13.48%) E: lxvmin, 109 (13.48%)
60% observation ($t_n = 64$)						
87 (2.24%)	86 (3.37%)	85 (9.09%)	89 **(0%)**	88(0.7) (1.12%)	109 (11.36%)	P: txvmax, 65 (26.96.%) E: lxvmin, 109 (13.48%)
70% observation ($t_n = 76$)						
87 (2.24%)	87 (2.24%)	89 **(0%)**	89 **(0%)**	87(0.9) (2.24%)	109 (13.48%)	P: lxvmax, 77 (13.48%) E: lxvmin, 109 (13.48%)
80% observation ($t_n = 87$)						
109 (22.47%)	88 (1.12%)	89 **(0%)**	87 (2.24%)	89(1.0) (2.24%)	109 (13.48%))	P: lxvmax, 88 (1.12%) E: exp, 109 (22.47%)
90% observation ($t_n = 98$)						
109 (22.47%)	109 (22.47%)	109 (22.47%)	98 **(10.11%)**	98(1.5) **(10.11%)**	109 (22.47%)	P: lxvmax, 99 (11.23%) E: lxvmin, 109 (13.48%)

Table 5.10: Prediction results of optimal software testing time with DS5 with $c_2 = 4$.

MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
50% observation ($t_n = 56$)						
56 (8.20%)	56 (8.20%)	57 **(6.55%)**	57 **(6.55%)**	56(1.8) (8.20%)	56 (8.20%)	P: gamma, 111 (81.96%) E: gamma, 111 (81.96%)
60% observation ($t_n = 67$)						
68 (11.47%)	68 (11.47%)	67 **(9.83%)**	67 **(9.83%)**	69(0.3) (13.11%)	111 (81.96%)	P: txvmax, 68 (11.47%) E: gamma, 111 (81.96%)
70% observation ($t_n = 78$)						
84 (37.70%)	84 (37.70%)	80 (31.14%)	78 **(27.87%)**	79(0.9) (29.51%)	111 (81.96%)	P: lxvmax, 79 (29.51%) E: gamma, 111 (81.96%)
80% observation ($t_n = 89$)						
111 (81.96%)	111 (81.96%)	89 **(45.90%)**	111 (81.96%)	111(1.6) (81.96%)	111 (81.96%)	P: txvmax, 90 (47.54%) E: gamma, 111 (81.96%)
90% observation ($t_n = 100$)						
111 (81.96%)	111 (81.96%)	111 (81.96%)	111 (81.96%)	111(1.5) (81.96%)	111 (81.96%)	P: txvmax, 101 **(65.57%)** E: gamma, 111 (81.96%)

Table 5.11: Prediction results of optimal software testing time with DS5 with $c_2 = 9$.

50% observation ($t_n = 56$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
61	68	60	67	65(1.2)	70	P: gamma, 111 (81.96%)
(8.20%)	(8.20%)	**(6.55%)**	**(6.55%)**	(8.20%)	(8.20%)	E: gamma, 111 (81.96%)
60% observation ($t_n = 67$)						
80	80	61	67	70(0.3)	111	P: txvmax, 68 (11.47%)
(11.47%)	(11.47%)	**(9.83%)**	**(9.83%)**	(13.11%)	(81.96%)	E: gamma, 111 (81.96%)
70% observation ($t_n = 78$)						
84	84	80	78	79(0.9)	111	P: txvmax, 79 (29.51%)
(37.70%)	(37.70%)	(31.14%)	**(27.87%)**	(29.51%)	(81.96%)	E: gamma, 111 (81.96%)
80% observation ($t_n = 89$)						
111	111	89	111	111(1.6)	111	P: txvmax, 90 (47.54%)
(81.96%)	(81.96%)	**(45.90%)**	(81.96%)	(81.96%)	(81.96%)	E: gamma, 111 (81.96%)
90% observation ($t_n = 100$)						
111	111	111	111	111(1.5)	111	P: txvmax, 101 **(65.57%)**
(81.96%)	(81.96%)	(81.96%)	(81.96%)	(81.96%)	(81.96%)	E: gamma, 111 (81.96%)

Table 5.12: Prediction results of optimal software testing time with DS6 with $c_2 = 4$.

50% observation ($t_n = 37$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
63	56	50	65	69(1.2)	40	P: gamma, 73 **(0%)**
(13.70%)	(23.28%)	(31.50%)	(10.96%)	(5.48%)	(45.20%)	E :gamma, 73 **(0%)**
60% observation ($t_n = 44$)						
65	60	70	69	70(1.7)	41	P: gamma, 73 **(0%)**
(31.50%)	(11.47%)	(4.11%)	(5.48%)	(13.11%)	(81.96%)	E: gamma, 73 **(0%)**
70% observation ($t_n = 51$)						
69	65	71	69	73(-2.9)	65	P: txvmax, 52 (29.51%)
(5.48%)	(31.50%)	(2.74%)	(5.48%)	**(0%)**	(31.50%)	E:gamma, 73 **(0%)**
80% observation ($t_n = 58$)						
73	65	71	73	73(1.9)	65	P: txvmax, 59 (47.54%)
(0%)	(31.50%)	(2.74%)	**(0%)**	**(0%)**	(31.50%)	E: gamma, 73 **(0%)**
90% observation ($t_n = 66$)						
73	65	71	73	73(-0.5)	73	P: lxvmax, 73 **(0%)**
(0%)	(31.50%)	(2.74%)	**(0%)**	**(0%)**	**(0%)**	E: gamma, 73 **(0%)**

Table 5.13: Prediction results of optimal software testing time with DS6 with $c_2 = 9$.

50% observation ($t_n = 37$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
69	69	69	73	69(0.3)	56	P: gamma, 73 **(0%)**
(5.48%)	(5.48%)	(5.48%)	**(0%)**	(5.48%)	(23.28%)	E: gamma, 73 **(0%)**
60% observation ($t_n = 44$)						
70	70	70	73	70(-1.1)	65	P: gamma, 73 **(0%)**
(4.11%)	(4.11%)	(4.11%)	**(0%)**	(13.11%)	(81.96%)	E: gamma, 73 **(0%)**
70% observation ($t_n = 51$)						
70	70	71	73	73(-2.9)	65	P: txvmax, 52 (29.51%)
(4.11%)	(4.11%)	(2.74%)	**(0%)**	**(0%)**	(31.50%)	E:gamma, 73 **(0%)**
80% observation ($t_n = 58$)						
73	70	71	73	73(1.9)	65	P: txvmax, 59 (47.54%)
(0%)	(4.11%)	(2.74%)	**(0%)**	**(0%)**	(31.50%)	E: gamma, 73 **(0%)**
90% observation ($t_n = 66$)						
73	71	71	73	73(-0.5)	73	P: lxvmax, 73 **(0%)**
(0%)	(2.74%)	(2.74%)	**(0%)**	**(0%)**	**(0%)**	E: gamma, 73 **(0%)**

Table 5.14: Prediction results of optimal software testing time with DS7 with $c_2 = 4$.

50% observation ($t_n = 41$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
42	46	49	50	46(1.9)	42	P: txvmax, 42 (40%)
(40%)	(34.28%)	(30%)	(28.57%)	(34.28%)	(40%)	E:lxvmin, 81 **(15.71%)**
60% observation ($t_n = 49$)						
56	56	65	55	65(0.3)	59	P: txvmax, 50 (28.57%)
(20%)	(20%)	**(7.14%)**	(21.43%)	**(7.14%)**	(15.71%)	E: lxvmin, 81 (15.71%)
70% observation ($t_n = 57$)						
69	58	71	59	59(0.9)	68	P: txvmax, 58 (17.14%)
(1.43%)	(4.11%)	**(1.43%)**	(15.71%)	(15.71%)	(2.85%)	E: lxvmin, 81 (15.71%)
80% observation ($t_n = 65$)						
69	69	71	71	73(1.9)	71	P: txvmax, 66 (6.06%)
(1.43%)	**(1.43%)**	**(1.43%)**	**((1.43%)**	(4.28%)	**(1.43%)**	E: llogist, 73 (4.23%)
90% observation ($t_n = 73$)						
73	73	73	73	73(-0.5)	73	P: txvmax, 73 **(4.23%)**
(4.23%)	**(4.23%)**	**(4.23%)**	**(4.23%)**	**(4.23%)**	**(4.23%)**	E: gamma, 73 **(4.23%)**

Table 5.15: Prediction results of optimal software testing time with DS7 with $c_2 = 9$.

50% observation ($t_n = 41$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
49 (34.66%)	49 (34.66%)	52 (30.66%)	65 (13.33%)	56(1.0) (25.33%)	81 **(8%)**	P: txvmax, 42 (44%) E:lxvmin, 81 **(8%)**
60% observation ($t_n = 49$)						
65 (13.33%)	65 (13.33%)	65 (13.33%)	70 **(3.75%)**	70(-0.9) **(3.75%)**	81 (8%)	P: txvmax, 50 (33.33%) E: lxvmin, 81 (8%)
70% observation ($t_n = 57$)						
73 **(2.67%)**	71 (5.33%)	71 (5.33%)	73 **(2.67%)**	71(1.6) (5.33%)	81 (8%)	P: txvmax, 58 (22.67%) E: lxvmin, 81 (8%)
80% observation ($t_n = 65$)						
73 (2.67%)	71 (5.33%)	71 (5.33%)	73 **(2.67%)**	71(0.8) (5.33%)	81 (8%)	P: txvmax, 66(12%) E: llogist, 73 (2.67%)
90% observation ($t_n = 73$)						
73 (2.67%)	73 (2.67%)	73 (2.67%)	75 **(0%)**	73(-0.5) (2.67%)	81 (8%)	P: txvmax, 73 (2.67%)) E: gamma, 73 (2.67%)

Table 5.16: Prediction results of optimal software testing time with DS8 with $c_2 = 4$.

50% observation ($t_n = 57$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
65 (42.98%)	65 (42.98%)	65 (42.98%)	65 (42.98%)	76(1.2) (33.33%)	114 **(0%)**	P: lxvmax, 114**(0%)** E: gamma, 114 **(0%)**
60% observation ($t_n = 68$)						
70 (38.60%)	74 (38.60%)	80 (29.82%)	68 (40.35%)	69(-0.9) (39.47%)	114 **(0%)**	P: lxvmax, 114 **(0%)** E: lxvmin, 114 **(0%)**
70% observation ($t_n = 80$)						
80 (29.82%)	80 (29.82%)	80 (29.82%)	80 (29.82%)	91(1.0) (20.17%)	114 **(0%)**	P: txvmax, 114 **(0%)** E: lxvmin, 114 **(0%)**
80% observation ($t_n = 91$)						
114 **(0%)**	114 **(0%)**	91 (20.17%)	91 (20.17%)	91(-0.6) (20.17%)	114 **(0%)**	P: lxvmax, 114**(0%)** E: lxvmin, 114 **(0%)**
90% observation ($t_n = 103$)						
114 **(0%)**	114 **(0%)**	114 **(0%)**	103 (9.65%)	103(-0.5) (9.65%)	114 **(0%)**	P: tlogist, 114 **(0%))** E: lxvmin, 114 **(0%)**

Table 5.17: Prediction results of optimal software testing time with DS8 with $c_2 = 9$.

50% observation ($t_n = 57$)						
MIMO ($\hat{t}_0$)						
AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best SRGM ($\hat{t}_0$)
80 (29.82%)	76 (33.33%)	65 (42.98%)	80 (29.82%)	91(1.2) (20.17%)	114 **(0%)**	P: lxvmax, 114 **(0%)** E: gamma, 114 **(0%)**
60% observation ($t_n = 68$)						
80 (29.82%)	68 (40.35%)	74 (38.60%)	91 (20.17%)	103(-0.9) (9.65%)	114 **(0%)**	P: lxvmax, 114 **(0%)** E: lxvmin, 114 **(0%)**
70% observation ($t_n = 80$)						
80 (29.82%)	80 (29.82%)	80 (29.82%)	91 (20.17%)	114(1.0) **(0%)**	114 **(0%)**	P: txvmax, 114 **(0%)** E: lxvmin, 114 **(0%)**
80% observation ($t_n = 91$)						
114 **(0%)**	114 **(0%)**	91 (20.17%)	91 (20.17%)	114(-0.6) **(0%)**	114 **(0%)**	P: lxvmax, 114**(0%)** E: lxvmin, 114 **(0%)**
90% observation ($t_n = 103$)						
114 **(0%)**	114 **(0%)**	114 **(0%)**	103 (9.65%)	114(-0.5) **(0%)**	114 **(0%)**	P: tlogist, 114 **(0%)** E: lxvmin, 114 **(0%)**

Chapter 6

Conclusions

6.1 Summary and Remarks

In Chapter 2, we consider on a prediction problem with the common multi-player perceptron neural network of the cumulative number of software faults in sequential software testing. We apply the well-known back propagation algorithm for feed forward neural network architectures. To predict the number of software faults, we impose a plausible assumption that the underlying fault-detection process obeys the Poisson law with unknown parameters. Since it is appropriate to input the training data as real number in the conventional MLP neural network, we propose to apply three data transformation methods from the Poisson count data to the Gaussian data: Bartlett transform (BT) [5], Anscombe transform (AT1) [4] and Fisz transform (FT) [12] of the cumulative number of software faults in one-stage look ahead prediction. To keep relatively small errors on software fault prediction, it is very important to train the neural network. The minimal error reflects better stability, and higher error reflects worst stability. We experiment with different numbers of hidden neurons and number of input neurons for all datasets.

In Chapter 3, we have considered the long-term prediction of the number of software faults, and propose a refined neural network approach with the grouped data, where the multi-stage look-ahead prediction is carried out with a simple MLP neural network with multiple outputs. Under the assumption that the software fault count data follows a Poisson process with an unknown mean value function, we transform the underlying Poisson count data to the Gaussian data via five data transformation methods. In here, we also conducted a Monte Carlo simulation for the single time data. We generate the original failure time data $X^*_{k,i}$ for $i = 1,\ 2,\ \ldots,\ n$, at k-th simulation as the pseudo random variables, by the *thinning algorithm* [30]. Consider the case where the failure time distribution $F(t)$ and the intensity function $\lambda(t)$ are completely unknown. Then, it can be shown that $X^*_{k,1},\ X^*_{k,2},\ \cdots\ (k = 1, 2, \ldots, m)$ follows an NHPP with intensity function $\lambda(t)$,Where $(m = 1000)$, after that we make grouped data from time data. That is, it is assumed that n failures occur by time t and the realizations of T_i $(i = 1, 2, \ldots, n)$, say, t_i are observed, where $t_n \leq t$. We regard the pair (t_i, x_i) $(i = 1, 2, \ldots, n)$ as a software count data of the underlying NHPP. From 1000 samples, we select 4 types of datasets (1) Fitted (2) Underfitted (3) Overfitted and (4) S-shaped.

Throughout numerical experiments with eight real software fault data and four simulation datasets, we compare our neural network approach with the existing software reliability growth models based on an nonhomogeneous Poisson process, in terms of predictive performance with an average relative error. It is shown that our neural network approach with BT affords a more appropriate point estimation and tends to have an enhanced performance from the viewpoint of predictability in the early phase of software testing.

In Chapter 4, we have proposed one prediction interval method, which were simulation-based approach. We have also applied the popular method for finding approximations based on Taylor series expansions delta method for two cases which were one-stage look-ahead and multi-stage look-ahead. Basically it is used the nonlinear regression. Furthermore, we have calculated the higher moments and two-sided prediction intervals, as well as the PICP, MPIW and PINAW. We have also investigated several statistical properties with respect to various estimators through simulation experiments and real data analysis. It could be confirmed that the point estimate was included between the 95% two-sided prediction intervals from the results of simulation experiment. On the other hand, we have shown that the user could utilize the useful information in order to determine the feature software fault by using the proposed neural network methods in real example. In Chapter 5, we have formulated an innovative technique to evaluate the optimal software release time using a neural network.

In our approach, a three-layer perceptron neural network with multiple outputs is used, where the underlying software fault count data are transformed into the Gaussian data by means of the well-known five data transformation method. Then the prediction of the optimal software testing time, which minimizes the expected software cost is carried out using the neural network. We have given illustrative examples with eight real software fault data, where we compare our approach with conventional NHPP -based SRGMs.

6.2 Future Works

For the same multi-stage look-ahead point estimation in Chapter 3, we will investigate the dependence of the hidden neuron and input neuron for refined NN architecture. For instance, any adaptive approach to estimate optimal hidden and input neuron will be useful to improve the estimation accuracy. We will also apply the present techniques to the other problems. For instance, probabilistic neural network (PNN) [68] is another challenging issue because the fast training process than back propagation and guaranteed to converge to an optimal classifier in the estimation framework.

For the prediction interval in Chapter 4, we will focus on the interval estimation of other software reliability measures (see [65],[64]). For this purpose, we intend to apply the bootstrap method, which is a representative statistical approach to replicate the original software fault data (e.g. see [62]).

Also, we will investigate various deep learning [66] architectures such as deep neural networks, convolutional deep neural networks, deep belief networks and recurrent neural networks for long-term prediction with the software fault count data.

Appendix A

AE result for four DS1 $\sim$ DS4 data sets

Suppose that the observation point is given by the n-th testing day, t_n. In this case, $(n-l)$ software fault counts data are used for training the MIMO neural network. The capability of the prediction model is measured by the average relative error (AE),

$$AE_l = \frac{\sum_{s=1}^{l} RE_s}{l}, \tag{A.1}$$

where RE_s is called the relative error for the future time $t = n + s$ and is given by

$$RE_s = \left| \frac{(\tilde{x}^o_{n+s} - \tilde{x}_{n+s})}{\tilde{x}^o_{n+s}} \right| \quad (s = 1, 2, \ldots, l). \tag{A.2}$$

So we regard the prediction model with smaller AE as a better prediction model.

In appendix A Tables A.1$\sim$A.16 summarize the results on AE for the underlying data set DS1$\sim$ DS4 at 50% $\sim$ 90% observation points of the whole data for the prediction length l=5, 10, 15, and 20 days, where the bold number implies the best prediction model in the same category. For instance, Table A.1 gives the prediction results on the cumulative number of software faults for 5 days prediction at respective observation points, when the number of hidden neurons changes from $k = 10$ to 50. In the MIMO neural network, we compare five data transform methods (AT1, AT2, FT, BT, and BoxCox) with the non-transformed case (Normal) and the best SRGM. In the column of SRGM, we denote the best SRGMs in terms of predictive performance (in the sense of minimum AE) and estimation performance (in the sense of minimum AIC) by P and E, respectively.

It is seen that our MIMO-based approaches in almost all k provide smaller AEs than the common SRGM when the observation point is 50% point. In both 60% and 70% observation points, the best prediction models are MIMO (FT and BoxCox) with $k = 40$, respectively. On the other hand, in the latter phase of software testing, *i.e.*, 80% $\sim$ 90% observation points, SRGMs, such as txvmin and txvmax and txvmax, offer less AEs than the MIMO neural networks. Even in these cases, it should be noted the best SRGM with the minimum AIC is not always equivalent to the best SRGM with the minimum AE. This fact tells that one cannot know exactly the best SRGM in terms of predictive performance in advance. However, in almost all cases, it is seen that the data transform can work well to give more accurate prediction results in the MIMO neural networks. Especially, MIMO with BoxCox and AT2 tends to give the better prediction result as the prediction length becomes longer for DS1.

Tables A.5$\sim$A.8 present the prediction results on AE for DS2. Similar to DS1, the MIMO neural networks can predict the cumulative number of software faults in the early testing phase, say, 50%–60% observation point, better than SRGMs. In this data, FT give the better prediction results. Focusing on the number of hidden neurons in the MIMO neural networks, we expected first that the larger k leads to the better predictive performance (See A.6–A.8). The prediction results on AE for DS3 and DS4 summarize in Tables A.9–A.16. In this case, comparing the data transform methods with SRGMs, we can find txvmax and lxvmax provide the best prediction result in 60%–90% in Table A.13, In the MIMO neural network approach, it is essential to determine a feasible k value because the number of hidden neurons results in the expensive computation cost with different prediction length l.

A.1 DS1

A.2 DS2

A.3 DS3

A.4 DS4

Table A.1: Comparison of average relative errors for five days prediction with DS1 ($l = 5$).

50% observation ($t_n = 31$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox (Best λ)	Normal	Best Model
10	0.118	0.096	0.161	0.126	0.157(-0.5)	0.224	
20	0.176	0.089	0.346	0.308	0.115(1.8)	0.137	P: pareto
30	0.121	0.071	0.096	0.462	**0.028**(0.7)	1.078	(1.280)
40	0.092	0.164	0.042	0.856	0.294(1.9)	0.136	E:txvmax
50	0.076	0.292	0.077	0.060	0.435(1.3)	0.078	(2.286)

60% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.035	0.485	0.028	0.045	0.052(0.2)	0.056	
20	0.026	0.384	0.073	0.021	0.116(1.6)	0.038	P:lnorm
30	0.183	0.021	0.127	0.131	0.072(1.1)	0.043	(0.023)
40	0.099	0.038	**0.016**	0.078	0.126(0.8)	0.029	E:txvmax
50	0.046	0.590	0.097	0.024	0.099(-0.3)	0.018	(0.764)

70% observation ($t_n = 43$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.136	0.053	0.257	1.094	0.245(0.245)	0.268	
20	0.166	0.479	0.078	0.119	0.151(0.3)	0.172	P:txvmax
30	0.416	0.138	0.429	0.083	1.105(1.3)	0.083	(0.017)
40	0.173	0.428	0.065	0.043	**0.016**(0.8)	0.098	E:txvmax
50	0.076	0.041	0.048	0.031	0.122(0.9)	0.056	(0.017)

80% observation ($t_n = 50$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.313	0.421	0.037	0.066	0.031(1.7)	0.017	
20	0.156	0.048	0.138	0.043	0.044(-0.4)	0.059	P:txvmin
30	0.167	0.035	0.164	0.042	0.019(0.2)	0.027	(**0.007**)
40	0.076	0.253	0.063	0.032	0.029(0.9)	0.018	E:lxvmin
50	0.013	0.058	0.029	0.048	0.119(0.8)	0.127	(0.022)

90% observation ($t_n = 56$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.237	0.157	0.032	0.319	0.145(0.4)	0.291	
20	0.205	0.037	0.147	0.196	0.068(1.0)	0.043	P:txvmax
30	0.457	0.056	0.028	0.152	0.184(0.6)	0.186	(**0.020**)
40	1.325	0.116	0.067	0.359	0.023(1.7)	0.034	E:lxvmin
50	0.076	0.287	0.052	0.026	0.134(1.3)	0.161	(0.030)

Table A.2: Comparison of average relative errors for ten days prediction with DS1 ($l = 10$).

60% observation ($t_n = 31$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.573	0.056	1.757	0.412	0.051(1.0)	0.958	
20	2.144	0.141	1.242	0.166	0.166(0.0)	0.479	P:lxvmin
30	0.767	**0.035**	0.862	1.241	0.341(0.6)	0.841	(1.120)
40	0.508	0.199	0.762	0.928	0.069(0.6)	0.057	E:txvmax
50	0.539	0.058	0.603	0.791	0.479(1.3)	1.023	(3.480)

60% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.938	0.118	1.645	1.098	0.043(0.7)	1.087	
20	1.130	0.032	0.946	1.048	0.946(-1.0)	1.106	P:tlogist
30	1.022	0.155	0.534	1.715	0.537(1.0)	1.746	(**0.025**)
40	1.537	0.138	0.043	0.947	0.035(1.6)	0.850	E:txvmax
50	0.035	0.029	0.785	0.047	0.623(1.1)	0.513	(1.287)

70% observation ($t_n = 43$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.069	0.177	2.062	1.043	0.088(2.0)	1.088	
20	0.137	0.064	1.723	1.066	0.219(1.0)	0.079	P:txvmax
30	1.062	0.107	1.647	0.549	0.265(1.4)	0.871	(0.077)
40	1.081	**0.046**	1.104	1.070	0.359(0.9)	1.058	E:txvmax
50	0.096	0.553	0.087	0.057	0.106(1.2)	1.099	(0.077)

80% observation ($t_n = 50$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.031	0.255	1.058	1.026	0.183(-2.0)	0.382	
20	0.897	0.457	0.658	1.074	0.058(1.0)	3.039	P:lxvmax
30	0.541	0.069	1.035	2.452	0.452(1.5)	1.052	(**0.018**)
40	0.100	0.547	0.126	0.135	0.198(1.9)	0.907	E:lxvmin
50	0.067	0.079	1.048	0.269	0.339(1.2)	0.839	(0.056)

Table A.3: Comparison of average relative errors for fifteen days prediction with DS1 ($l = 15$).

50% observation ($t_n = 31$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.341	0.281	1.039	1.319	0.056(-1.4)	0.348	
20	1.713	**0.039**	1.716	1.034	0.339(0.3)	1.728	P:lxvmin
30	0.296	0.346	0.846	0.198	0.087(1.2)	1.298	(1.550)
40	0.082	0.050	1.234	1.220	0.641(-1.1)	1.461	E:txvmax
50	1.062	0.860	1.329	0.775	0.331(0.2)	1.073	(4.981)

60% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.538	**0.037**	1.040	1.094	0.041(-1.0)	1.028	
20	0.480	0.141	1.036	1.355	0.279(1.0)	1.047	P:tlogist
30	1.079	0.191	1.049	0.989	0.198(0.7)	2.076	(0.042)
40	0.637	0.165	1.159	1.046	0.792(0.6)	1.037	E:txvmax
50	0.038	0.217	0.044	0.047	0.427(0.5)	0.076	(0.049)

70% observation ($t_n = 43$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.439	0.049	1.076	1.055	0.524(0.6)	1.024	
20	1.226	0.038	0.078	1.061	0.187(1.0)	0.087	P:txvmax
30	1.047	0.059	1.269	0.053	0.151(-1.8)	0.519	(**0.028**)
40	1.058	0.582	1.152	0.647	0.719(0.8)	1.129	E:txvmax
50	0.056	0.157	0.046	0.049	0.137(0.7)	0.072	(**0.028**)

Table A.4: Comparison of average relative errors for twenty days prediction with DS1 ($l = 20$).

50% observation ($t_n = 31$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.362	0.277	0.491	1.231	**0.018**(-0.6)	1.240	
20	0.319	0.146	1.318	0.909	0.169(0.0)	0.390	P:lxvmin
30	0.800	0.168	0.516	0.655	0.421(0.6)	0.634	(1.088)
40	1.124	0.267	1.384	1.216	0.293(0.2)	3.226	E:txvmax
50	2.300	0.120	0.526	0.864	0.591(-1.1)	0.451	(6.103)

60% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.071	0.119	2.210	0.136	0.071(-2.0)	0.524	
20	1.124	**0.015**	1.142	0.503	0.241(1.3)	2.072	P:llogist
30	0.076	0.125	0.240	0.506	0.034(-0.7)	0.630	(0.033)
40	0.159	0.455	0.199	0.120	0.641(0.8)	2.041	E:txvmax
50	0.049	0.199	0.090	0.057	0.056(2.0)	0.379	(0.042)

Table A.5: Comparison of average relative errors for five days prediction with DS2 ($l = 5$).

50% observation ($t_n = 21$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.269	0.173	**0.011**	0.293	0.055(0.4)	0.073	
20	0.079	0.078	0.456	1.251	0.046(1.3)	1.207	P:lxvmax
30	1.162	0.041	0.196	0.489	0.555(1.1)	0.341	(0.211)
40	1.059	0.154	1.041	0.264	0.279(1.7)	0.253	E:lxvmin
50	0.046	0.281	1.415	0.086	0.180(-0.9)	0.165	(2.011)

60% observation ($t_n = 25$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.138	0.177	1.026	0.047	0.472(1.0)	1.174	
20	0.222	0.101	**0.013**	0.021	0.058(0.0)	0.152	P:tlogist
30	0.318	0.148	1.125	0.059	0.657(1.0)	0.173	(0.039)
40	1.099	0.171	1.041	0.035	0.531(1.9)	0.332	E:lxvmin
50	0.026	0.158	0.071	0.024	0.374(0.4)	1.015	(0.075)

70% observation ($t_n = 29$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.133	0.044	2.028	0.814	0.476(0.6)	1.159	
20	0.027	0.104	1.029	1.019	0.380(0.0)	0.466	P:lxvmin
30	0.078	0.072	1.032	1.026	0.899(-0.7)	0.395	**(0.021)**
40	0.091	0.211	1.107	0.120	0.206(00.9)	1.055	E:lxvmin
50	0.063	0.062	0.108	0.029	0.331(0.7)	0.786	**(0.021)**

80% observation ($t_n = 33$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.031	0.061	1.023	1.096	0.394(0.5)	1.819	
20	1.016	0.139	1.035	0.643	0.312(2.0)	0.988	P:lxvmin
30	0.084	0.034	1.032	1.026	0.339(1.0)	1.195	**(0.026)**
40	1.024	0.243	0.065	0.075	0.471(0.0)	1.023	E:lxvmin
50	0.896	0.043	0.029	0.746	0.107(-0.3)	0.885	**(0.026)**

90% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.058	0.452	1.032	0.0846	0.876(1.3)	1.1521	
20	1.921	0.911	0.069	0.028	0.149(0.6)	1.182	P:txvmax
30	0.682	0.022	0.679	1.036	03297(1.5)	1.050	**(0.002)**
40	1.035	0.018	1.067	1.036	0.017(2.0)	1.133	E:lxvmin
50	0.080	0.032	0.034	0.019	0.063(1.8)	0.603	(0.039)

Table A.6: Comparison of average relative errors for ten days prediction with DS2 ($l = 10$).

50% observation ($t_n = 21$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
e 10	0.981	0.068	0.457	0.618	0.067(1.3)	0.541	
20	0.077	0.149	0.368	0.136	0.139(-0.7)	0.216	P:txvmin
30	0.043	**0.031**	0.185	0.066	0.791(2.0)	0.035	(0.059)
40	0.245	0.127	0.052	0.079	0.114(1.5)	0.058	E:lxvmin
50	0.041	0.049	0.042	0.086	0.383(1.0)	0.065	(2.613)
60% observation ($t_n = 25$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.075	0.091	0.118	0.852	0.814(2.0)	0.189	
20	0.068	0.159	0.238	1.390	0.755(0.0)	0.359	P:llogist
30	0.615	0.454	0.159	0.154	0.313(1.0)	0.273	(0.102)
40	0.692	0.479	0.138	0.147	0.983(0.0)	0.704	E:lxvmin
50	0.066	**0.060**	0.084	0.078	0.083(2.0)	0.136	(0.134)
70% observation ($t_n = 29$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox (Best λ)	Normal	Best Model
10	0.043	0.042	0.063	1.013	0.585(1.8)	1.051	
20	1.026	0.051	1.013	0.192	0.289(1.0)	0.367	P:lxvmin
30	0.081	0.196	0.047	0.698	0.385(0.8)	0.551	**(0.015)**
40	1.013	0.188	0.071	0.098	0.291(1.1)	0.341	E:lxvmin
50	0.058	0.038	0.062	0.089	0.173(0.6)	0.284	**(0.015)**

Table A.7: Comparison of average relative errors for fifteen days prediction with DS2 ($l = 15$).

50% observation ($t_n = 21$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox (Best λ)	Normal	Best Model
10	1.164	0.434	0.083	0.975	0.219(-0.1)	2.556	
20	0.896	0.169	0.176	1.889	0.476(2.09	0.756	P:tnorm
30	1.175	0.201	0.805	0.897	0.116(1.0)	1.569	(0.227)
40	1.879	0.266	1.046	1.128	0.336(1.7)	0.238	E:lxvmin
50	0.145	0.448	**0.070**	0.284	0.215(1.4)	0.533	(2.865)
50% observation ($t_n = 21$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	3.346	3.293	0.267	4.142	0.159(1.0)	3.162	
20	2.127	0.767	2.189	0.797	0.705(1.6)	1.158	P:llogist
30	0.994	3.158	1.396	1.032	0.211(1.3)	2.190	(0.161)
40	1.319	0.263	2.336	1.289	0.839(2.0)	1.097	E:lxvmin
50	0.183	3.309	0.192	0.160	**0.039** (1.1)	0.385	(0.184)

Table A.8: Comparison of average relative errors for twenty days prediction with DS2 ($l = 20$).

50% observation ($t_n = 21$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	2.060	0.090	1.060	1.145	0.473(1.0)	1.016	
20	1.066	0.182	1.109	2.217	0.672(2.0)	1.023	P:lxvmax
30	1.047	0.072	1.056	0.357	0.745(0.5)	1.302	(0.310)
40	1.055	0.139	0.856	1.201	1.036(0.9)	1.251	E:lxvmin
50	**0.084**	0.078	0.088	0.099	0.305(2.0)	0.392	(2.960)

Table A.9: Comparison of average relative errors for five days prediction with DS3 ($l = 5$).

50% observation ($t_n = 23$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.042	0.089	0.034	0.311	1.248(1.2)	0.332	
20	0.048	0.069	0.314	0.141	0.941(0.7)	1.319	P:exp
30	0.126	0.**027**	0.236	0.187	0.217(1.1)	1.628	(0.105)
40	0.151	0.163	0.147	0.321	1.246(-0.6)	2.489	E:lnorm
50	0.427	0.375	0.091	0.314	1.241(1.1)	0.371	(2.262)
60% observation ($t_n = 28$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.056	0.098	1.028	0.036	1.245(1.4)	1.895	
20	0.034	0.026	0.649	0.412	0.641(-1.0)	1.254	P:pareto
30	0.019	0.113	0.037	1.243	0.754(1.6)	0.987	(0.079)
40	0.108	0.033	0.017	0.595	0.281(1.6)	0.124	E:lnorm
50	0.016	0.109	0.082	**0.014**	0.943(2.0)	0.236	(1.255)
70% observation ($t_n = 32$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.039	0.112	1.623	1.254	0.679(1.3)	0.215	
20	1.040	0.029	0.589	2.045	1.248(1.9)	0.264	P:txvmax
30	1.942	0.026	0.729	0.689	0.473(-0.9)	0.647	(0.019)
40	0.025	0.092	0.031	0.128	0.341(-1.3)	0.198	E:gamma
50	0.064	0.062	0.063	**0.015**	0.069(1.4)	0.322	**(2.581)**
80% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.043	0.293	0.625	3.2589	0.985(1.7)	1.091	
20	0.096	0.117	0.048	1.2589	1.258(-0.3)	0.389	P:exp
30	0.031	0.021	**0.019**	1.031	0.489(1.4)	0.215	(0.038)
40	0.039	0.070	0.041	0.085	0.327(0.9)	0.608	E:lxvmin
50	0.124	0.129	0.063	0.047	0.264(1.1)	0.182	(2.993)
90% observation ($t_n = 41$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.153	0.025	0.618	1.203	0.291(0.8)	0.381	
20	1.921	0.111	0.377	0.698	0.746(-0.9)	0.343	P:txvmax
30	0.682	0.422	1.018	0.314	0.547(1.9)	0.658	(**0.012**)
40	1.035	0.618	0.314	0.228	0.149(0.9)	1.583	E:lxvmin
50	0.080	0.062	0.157	0.331	0.641(-0.8)	1.379	(3.064)

Table A.10: Comparison of average relative errors for five days prediction with DS3 ($l = 10$).

50% observation ($t_n = 23$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.167	0.231	0.151	0.271	0.489(0.2)	0.636	
20	0.371	0.137	0.355	**0.112**	0.331(-0.7)	0.483	P:exp
30	0.292	0.156	0.276	0.331	0.634(-1.0)	0.841	(0.113)
40	0.287	0.356	0.199	0.344	1.012(-1.6)	1.125	E:lnorm
50	0.461	0.481	0.324	0.194	0.329(1.3)	0.562	(3.587)
60% observation ($t_n = 28$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.233	0.217	0.127	0.251	0.677(-2.4)	1.098	
20	0.549	0.356	0.179	0.187	0.181(1.2)	1.066	P:txvmax
30	0.142	0.191	0.137	0.137	0.649(-1.7)	2.112	(**0.100**)
40	0.241	0.239	0.104	0.149	0.982(1.0)	3.033	E:lnorm
50	0.431	0.429	0.231	0.255	0.624(-1.3)	1.108	(3.028)
70% observation ($t_n = 32$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.033	0.196	1.024	0.027	0.489(1.1)	0.526	
20	1.034	0.047	0.309	0.141	0.128(0.9)	0.579	P:txvmax
30	0.319	0.311	0.112	0.034	0.879(-0.3)	1.258	(**0.019)**
40	0.045	0.047	0.317	0.129	0.693(-1.7)	0.985	E:gamma
50	0.446	0.191	0.217	0.104	0.312(1.2)	0.698	(2.581)

Table A.11: Comparison of average relative errors for five days prediction with DS3 ($l = 15$).

50% observation ($t_n = 23$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.073	0.368	1.132	0.189	0.618(-1.2)	1.027	
20	1.087	0.818	1.081	0.412	1.028(0.4)	0.941	P:txvmax
30	1.152	0.092	0.473	0.441	0.987(-0.3)	1.034	(1.285)
40	0.983	1.072	0.625	0.458	2.056(1.0)	3.023	E:lnorm
50	**0.067**	0.089	0.078	0.081	0.689(1.8)	0.908	(2.252)
60% observation ($t_n = 28$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.151	0.147	1.028	0.411	0.985(1.0)	1.089	
20	0.058	1.028	0.466	0.349	0.875(-1.3)	1.187	P:txvmax
30	0.248	0.181	0.161	0.891	1.069(0.9)	1.137	(**0.035**)
40	0.189	0.259	0.174	0.319	0.467(-0.8)	1.1259	E:lnorm
50	0.443	0.433	0.299	0.313	0.745(1.3)	1.589	(1.255)

Table A.12: Comparison of average relative errors for five days prediction with DS3 ($l = 20$).

50% observation ($t_n = 23$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.331	0.289	0.693	0.889	0.229(1.6)	0.349	
20	0.214	0.236	0.1031	0.991	0.639(-1.1)	0.987	P:txvmax
30	0.673	0.196	0.981	1.029	1.021(1.6)	1.189	(1.285)
40	0.544	0.541	0.574	0.631	1.149(1.4)	4.214	E:lnorm
50	0.198	0.115	**0.057**	0.063	1.119(-1.9)	1.028	(2.252)

Table A.13: Comparison of average relative errors for five days prediction with DS4 ($l = 5$).

50% observation ($t_n = 54$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.377	0.371	0.374	0.501	0.710(1.7)	1.028	
20	0.527	0.551	0.292	0.421	0.699(1.0)	1.479	P:exp
30	0.209	0.578	0.239	0.336	0.402(-0.3	0.987	(0.853)
40	0.251	0.504	0.209	0.351	0.615(-0.9)	1.602	E:lxvmin
50	0.443	0.641	**0.131**	0.329	0.931(2.0)	1.494	(2.920)
60% observation ($t_n = 65$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.157	0.302	0.298	0.302	0.366(1.0)	1.031	
20	0.266	0.249	0.523	0.295	0.863(-1.1)	1.689	P:txvmax
30	0.238	0.291	0.322	1.243	0.944(1.4)	2.749	(**0.003**)
40	0.196	0.141	0.063	0.384	0.639(-2.3)	1.248	E:lxvmin
50	0.116	0.024	0.098	0.310	0.213(-3.0)	1.168	(3.193)
70% observation ($t_n = 76$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.062	0.183	0.381	0.400	0.223(-0.3)	0.895	
20	0.139	0.031	0.221	0.318	0.058(1.0)	1.025	P:txvmax
30	0.034	0.128	0.020	0.288	0.021(1.3)	1.350	(**0.007**)
40	0.243	0.278	0.116	0.129	0.058(0.2)	0.897	E:lxvmin
50	0.143	0.455	0.109	0.085	0.131(0.1)	1.029	(3.030)
80% observation ($t_n = 87$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.113	0.092	0.329	0.306	0.327(0.7)	1.061	
20	0.433	0.227	0.247	0.145	0.210(-1.3)	1.309	P:lxvmax
30	0.471	0.533	0.211	0.095	0.097(-2.4)	1.015	(**0.005**)
40	0.163	0.212	0.061	0.189	0.189(-1.9)	0.688	E:lxvmin
50	0.071	0.059	0.089	0.182	0.287(1.0)	0.682	(2.782)
90% observation ($t_n = 98$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.169	0.616	0.076	0.147	0.619(1.8)	1.034	
20	0.031	0.524	0.064	0.027	0.526(-0.7)	0.974	P:lxvmax
30	0.681	0.437	0.177	0.587	0.437(1.2)	0.1.369	**(0.000)**
40	0.049	0.354	0.198	0.035	0.349(0.1)	0.495	E:lxvmin
50	0.710	0.306	0.293	0.046	0.211(-0.1)	0.329	(2.928)

Table A.14: Comparison of average relative errors for five days prediction with DS4 ($l = 10$).

50% observation ($t_n = 54$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.278	0.637	0.629	0.621	0.223(1.1)	1.352	
20	0.425	0.089	0.053	0.505	0.055(1.2)	0.894	P:txvmax
30	0.212	0.861	0.044	0.053	0.021(1.3)	0.937	(0.031)
40	**0.016**	0.175	0.402	0.045	0.658(-0.7)	852	E:lxvmin
50	0.378	0.183	0.462	0.041	1.031(-2.3)	1.859	(5.374)
60% observation ($t_n = 65$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox	Normal	Best Model
10	0.428	0.127	0.447	0.223	0.366(1.1)	0.347	
20	0.095	0.058	0.037	0.055	0.263(-1.1)	1.437	P:txvmax
30	0.604	0.211	0.325	**0.021**	0.244(1.0)	0.749	(0.106)
40	0.407	0.511	0.127	0.058	0.139(1.3)	3.126	E:lxvmin
50	0.071	0.135	0.274	1.310	0.113(2.0)	0.869	(5.626)
70% observation ($t_n = 76$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.478	0.592	0.539	0.485	0.411(1.2)	1.665	
20	0.812	0.638	0.477	0.492	0.298(1.4)	2.035	P:lxvmax
30	1.485	0.069	0.313	0.406	0.191(1.6)	2.050	(**0.016**)
40	1.025	0.123	0.384	0.323	0.539(-2.2)	1.197	E:lxvmin
50	0.115	0.092	0.174	0.176	0.486(-0.1)	1.099	(5.576)
80% observation ($t_n = 87$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.218	1.039	0.016	0.221	0.602(-0.3)	1.258	
20	0.055	**0.043**	0.317	0.151	0.447(-2.7)	0.941	P:txvmax
30	0.483	0.528	0.071	0.122	0.378(2.0)	0.446	(0.058)
40	0.986	0.512	0.069	0.111	0.325(1.0)	1.214	E:lxvmin
50	0.790	0.529	0.087	0.133	0.689(1.9)	2.427	(5.698)

Table A.15: Comparison of average relative errors for five days prediction with DS4 ($l = 15$).

50% observation ($t_n = 54$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.379	0.207	0.426	0.555	0.327(1.3)	1.023	
20	1.039	0.981	0.392	0.492	0.218(0.2)	1.021	P:txvmax
30	1.031	1.082	0.330	0.409	0.627(-1.3)	0.647	(**0.040)**
40	0.887	0.247	0.244	0.401	0.548(1.7)	0.189	E:lxvmin
50	1.451	0.062	0.220	0.385	0.661(2.0)	1.214	(7.914)
60% observation ($t_n = 65$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox	Normal	Best Model
10	0.342	0.227	0.338	0.329	0.368(0.6)	0.689	
20	0.398	0.117	0.197	0.405	0.383(1.0)	2.018	P:txvmax
30	0.063	0.117	0.081	0.439	0.244(1.7)	0.698	(0.024)
40	0.279	0.594	0.314	0.610	0.471(-2.6)	1.125	E:lxvmin
50	**0.022**	0.161	0.109	0.651	0.663(-1.9)	2.056	(8.538)
70% observation ($t_n = 76$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.479	0.285	0.269	0.284	0.388(-2.2)	1.098	
20	0.803	1.031	0.056	0.061	0.355(0.6)	1.458	P:txvmax
30	1.023	0.865	0.044	0.096	0.298(-1.7)	0.998	(**0.028)**
40	1.076	0.829	0.037	0.052	0.208(-2.7)	1.025	E:lxvmin
50	0.412	0.361	0.457	0.181	0.636(-1.0)	1.091	(9.258)

Table A.16: Comparison of average relative errors for five days prediction with DS4 ($l = 20$).

50% observation ($t_n = 54$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.229	0.158	0.744	0.129	0.977(1.2)	1.096	
20	0.058	0.085	0.459	**0.036**	0.415(0.3)	0.697	P:txvmax
30	1.196	1.347	0.290	0.054	0.362(1.8)	1.108	(0.048)
40	0.194	0.094	0.121	0.148	0.507(-2.7)	1.236	E:lxvmin
50	1.186	0.369	0.767	0.262	0.461(2.6)	0.741	(10.461)
60% observation ($t_n = 65$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox	Normal	Best Model
10	0.716	0.045	0.186	0.355	0.539(1.1)	1.024	
20	0.149	1.079	0.045	0.265	0.477(1.1)	0.516	P:txvmax
30	**0.013**	0.044	0.068	0.219	0.394(1.1)	1.096	(0.124)
40	0.861	1.897	0.096	0.199	0.414(-2.4)	1.118	E:lxvmin
50	1.068	0.867	0.188	0.151	0.183(-1.9)	1.407	(11.458)

Appendix B

AE result for four DS5 ∼ DS8 data sets

In appendix B Tables B.1∼B.16 summarize the results on AE for the underlying data set DS5∼ DS8 at 50% ∼ 90% observation points of the whole data for the prediction length l=5, 10, 15, and 20 days. In DS5, 50%–90% all cases SRGM with gamma, txvmax and lxvmax offer less AEs than the MIMO neural networks (See table B.1 for l=5). In addition, for the prediction length l=10, 15, and 20 days, SRGM with txvmax and lxvmax provide best prediction result than MIMO methods except 50% observation point in Tables B.2, B.3 and B.4. Tables B.5–B.8 present the prediction results on AE for DS6. Similar to DS1, DS2, and DS3, the MIMO neural networks can predict the cumulative number of software faults early testing phase, say, 50%–70% observation point, better than SRGMs. In this data, MIMO with FT, BT and AT1 give the better prediction results. For DS7, the prediction length l=10, and 20 days almost all testing phase MIMO with FT, BT and AT1 deliver small error than SRGMs in Tables B.10 and B.12 respectively. On the other hand, for l=5, and 15 days SRGM with txvmax presents best prediction method (See Tables B.9 and B.11). Tables B.13–B.16 shows the results on AE for DS8. In that case , without few cases all most testing phase MIMO with AT2, BT and FT provide best result than SRGMs. Comparing the data transform methods with the non-transformed one, we cannot find any case where Normal provides the best prediction result in Table include in appendix A and B. Finally, we cannot obtain the strong conclusion on how to design the MIMO for the purpose of software fault prediction. However, the lesson learned from the numerical experiments suggests that the multi-stages look-ahead prediction of software fault count is possible with the MIMO neural network, and that the data transform from the Poisson data to the Gaussian data works better to predict the number of software faults accurately.

B.1 DS5

B.2 DS6

B.3 DS7

B.4 DS8

Table B.1: Comparison of average relative errors for five days prediction with DS5 ($l = 5$).

50% observation ($t_n = 55$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.103	0.198	0.0143	0.066	0.536(1.2)	0.984	
20	0.175	0.052	0.042	0.086	0.385(1.3)	1.023	P:gamma
30	0.247	0.426	0.086	0.064	0.482(-0.5)	0.612	(**0.008**)
40	0.144	0.094	0.181	0.311	0.352(0.0)	1.021	E:gamma
50	0.111	0.364	0.119	0.171	0.685(-2.0)	1.011	(**0.008**)
60% observation ($t_n = 67$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.017	0.096	0.044	0.047	0.087(-2.5)	0.389	
20	0.059	0.021	0.065	0.033	1.038(1.2)	1.023	P:txvmax
30	0.023	0.096	0.028	0.065	0.018(0.8)	0.412	(**0.001**)
40	0.241	0.127	0.029	0.062	0.431(-2.3)	1.012	E:gamma
50	0.048	0.019	0.065	0.017	0.632(-0.9)	1.033	(3.098)
70% observation ($t_n = 78$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.257	0.184	0.053	0.056	0.015(1.3)	0.895	
20	0.056	0.088	0.117	0.124	0.387(1.9)	1.025	P:txvmax
30	0.023	0.022	0.065	0.028	0.521(1.3)	1.350	(**0.001**)
40	0.018	0.124	0.042	0.241	0.258(0.2)	0.897	E:gamma
50	0.048	0.022	0.101	0.012	0.131(0.1)	1.029	(3.024)
80% observation ($t_n = 89$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.213	0.252	0.038	0.857	0.144(1.7)	1.152	
20	0.033	0.027	0.024	0.674	0.125(-2.3)	1.631	P:lxvmax
30	0.021	0.073	0.053	0.095	0.423(-3.0)	2.015	(**0.000**)
40	0.163	0.212	0.031	0.189	0.325(2.0)	0.589	E:gamma
50	0.071	0.059	0.097	0.911	0.124(1.0)	0.378	(3.035)
90% observation ($t_n = 100$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.037	0.689	0.193	0.237	0.229(1.4)	1.378	
20	0.059	0.014	0.126	0.047	0.226(1.7)	1.234	P:txvmax
30	0.179	0.237	0.077	0.507	0.407(-2.0)	3.698	(**0.001**)
40	0.074	0.021	0.191	0.355	0.114(-1.1)	2.458	E:gamma
50	0.261	0.127	0.203	0.092	0.227(0.3)	1.017	(2.878)

Table B.2: Comparison of average relative errors for five days prediction with DS5 ($l = 10$).

50% observation ($t_n = 55$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.055	0.079	0.048	0.058	0.396(1.2)	0.514	
20	0.124	0.065	0.101	0.089	0.189(-2.3)	0.457	P:gamma
30	0.081	0.369	0.189	0.056	0.478(0.5)	0.414	(0.029)
40	0.278	0.178	0.639	0.566	0.147(0.6)	1.044	E:gamma
50	0.451	0.502	**0.021**	0.061	0.685(-2.0)	0.618	(0.029)
60% observation ($t_n = 67$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.637	0.727	0.191	0.123	1.021(1.0)	1.069	
20	0.074	0.237	0.095	0.041	1.112(-2.0)	2.045	P:lxvmax
30	0.121	0.403	0.138	0.015	0.871(-1.8)	0.698	(**0.004)**
40	0.316	0.307	0.077	0.089	0.571(2.1)	0.508	E:gamma
50	0.408	0.239	0.067	0.251	0.689(-1.1)	0.439	(5.691)
70% observation ($t_n = 78$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.219	0.275	0.191	0.123	0.364(-0.3)	0.699	
20	0.047	0.337	0.095	0.041	0.511(1.1)	1.011	P:txvmax
30	0.022	0.369	0.138	0.032	0.519(0.3)	1.220	(**0.00)**
40	0.116	0.512	0.077	0.129	0.589(-2.8)	0.634	E:gamma
50	0.141	0.239	0.067	0.052	0.481(-0.9)	1.021	(0.008)
80% observation ($t_n = 89$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.638	0.723	0.052	0.058	0.158(0.0)	1.895	
20	1.023	1.027	0.036	0.049	0.105(1.0)	2.056	P:lxvmax
30	0.951	2.088	0.101	0.032	0.698(-1.1)	3.065	(**0.008)**
40	1.032	0.711	0.059	0.160	1.011(-2.7)	1.027	E:gamma
50	0.241	0.589	0.256	0.033	0.278(-1.0)	0.748	(5.526)
90% observation ($t_n = 100$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.487	0.247	0.322	0.217	0.375(-3.0)	1.017	
20	0.317	0.655	0346	0.222	0.203(-2.9)	2.002	P:txvmax
30	0.0.691	0.367	0.108	0.021	0.691(0.0)	1.691	(**0.003)**
40	0.348	0.136	0.031	0.180	0.114(2.0)	1.025	E:gamma
50	0.343	0.292	0.269	0.179	326(-0.3)	0.894	(5.230)

Table B.3: Comparison of average relative errors for five days prediction with DS5 ($l = 15$).

50% observation ($t_n = 55$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.525	0.431	0.095	**0.025**	0.127(1.0)	0.963	
20	0.277	0.169	0.111	0.092	0.453(-1.1)	1.258	P:txvmax
30	0.435	0.326	0.132	0.049	0.616(-0.3)	1.028	(0.034)
40	0.233	0.139	0.156	0.511	0.263(0.7)	1.037	E:gamma
50	0.242	0.655	0.159	0.032	0.591(-2.7)	1.341	(0.044)
60% observation ($t_n = 67$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.637	0.727	0.043	0.049	1.201(0.0)	2.023	
20	0.074	0.237	0.051	0.005	0.308(-2.9)	3.063	P:txvmax
30	0.021	0.403	0.107	0.078	0.199(-1.6)	2.301	(**0.007**)
40	0.016	0.307	0.109	0.124	0.895(2.3)	1.258	E:gamma
50	0.288	0.239	0.051	0.589	0.239(-1.7)	1.241	(8.269)
70% observation ($t_n = 78$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.236	0.048	0.153	0.050	0.489(1.3)	0.822	
20	0.153	0.104	0.072	0.451	0.896(1.2)	1.040	P:txvmax
30	0.127	0.067	0.079	0.032	0.734(-0.7)	0.879	(**0.003**)
40	0.122	0.121	0.134	0.259	0.333(1.4)	1.055	E:gamma
50	0.114	0.132	0.059	0.144	0.353(1.9)	1.443	(8.018)
80% observation ($t_n = 89$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.065	0.051	0.098	0.591	0.369(1.2)	1.234	
20	0.091	1.023	0.103	0.158	0.213(1.2)	1.152	P:txvmax
30	0.098	0.0678	0.156	0.163	0.671(-0.9)	1.126	(**0.002**)
40	0.108	0.0491	0.163	0.612	0.481(1.2)	0.981	E:gamma
50	0.212	0.053	0.119	0.825	0.347(-1.5)	1.024	(7.969)

Table B.4: Comparison of average relative errors for five days prediction with DS5 ($l = 20$).

50% observation ($t_n = 55$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.248	1.611	1.235	0.623	1.287(-1.1)	1.083	
20	1.021	1.069	0.301	0.092	0.653(1.3)	0.568	P:txvmax
30	1.201	0.636	0.309	0.049	1.016(0.0)	0.928	(0.037)
40	0.181	0.039	0.256	0.511	0.153(-3.0)	1.237	E:gamma
50	0.042	1.025	0.159	**0.032**	0.555(-2.6)	0.369	(0.068)
60% observation ($t_n = 67$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.522	0.712	0.011	0.063	1.201(-0.9)	1.203	
20	0.291	0.363	0.029	0.059	1.235(1.2)	4.211	P:txvmax
30	0.128	0.577	0.631	0.037	1.021(1.6)	3.266	(**0.009**)
40	0.589	0.637	0.621	0.667	1.296(-2.7)	1.219	E:gamma
50	0.683	0.699	0.181	0.411	0.961(2.0)	0.987	(10.857)
70% observation ($t_n = 78$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.627	0.265	0.037	0.333	0.097(1.7)	0.548	
20	0.841	0.162	0.101	0.260	0.289(0.6)	0.961	P:txvmax
30	0.622	0.685	0.062	0.092	0.281(1.6)	1.011	(0.021)
40	0.798	0.859	0.118	**0.014**	0.488(-0.3)	1.478	E:gamma
50	0.771	0.832	0.629	0.199	1.241(-0.9)	1.124	(10.484)

Table B.5: Comparison of average relative errors for five days prediction with DS6 ($l = 5$).

50% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.425	0.754	0.514	**0.034**	0.423(1.1)	1.258	
20	0.287	0.332	0.457	0.095	0.397(1.2)	1.023	P:gamma
30	0.237	0.262	0.414	0.049	0.374(0.5)	0.985	(0.039)
40	0.189	0.259	0.141	0.389	0.189(1.0)	1.521	E:gamma
50	0.159	0.398	0.618	0.132	0.158(-2.8)	1.187	(0.039)
60% observation ($t_n = 44$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.348	0.475	0.144	0.247	0.475(-2.9)	0.502	
20	0.724	0.963	0.335	0.333	1.038(-1.6)	0.559	P:gamma
30	0.381	0.271	**0.079**	0.265	0.366(1.2)	0.526	(0.082)
40	0.296	0.489	0.129	0.162	0.144(-2.3)	2.589	E:gamma
50	0.632	0.498	0.265	0.117	0.396(1.9)	1.962	(0.082)
70% observation ($t_n = 51$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.891	0.226	0.075	0.163	0.809(1.2)	0.895	
20	0.143	0.196	0.375	0.181	0.779(-1.3)	1.025	P:txvmax
30	0.979	0.108	0.407	0.411	0.201(-2.3)	1.350	(**0.052**)
40	0.147	0.318	0.249	0.249	0.158(-2.7)	0.897	E:gamma
50	0.189	0.136	0.114	0.289	0.694(2.0)	3.699	(2.608)
80% observation ($t_n = 58$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.098	0.278	0.163	0.275	0.404(1.5)	2.693	
20	0.557	0.386	0.189	0.164	0.863(-2.1)	1.478	P:txvmax
30	0.577	0.584	0.407	0.258	0.691(2.0)	1.962	(0.046)
40	0.618	0.341	0.318	0.154	0.158(-2.2)	1.203	E:gamma
50	0.482	0.209	**0.045**	0.169	0.744(1.6)	0.961	(2.696)
90% observation ($t_n = 66$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.777	0.113	0.277	0.164	0.229(1.4)	1.378	
20	0.353	0.275	0.035	0.525	0.226(1.7)	1.234	P:lxvmax
30	0.259	0.401	0.110	0.053	0.407(-2.0)	3.698	(**0.001**)
40	0.405	0.677	0.906	0.135	0.114(-1.1)	2.458	E:gamma
50	0.079	0.042	0.091	0.033	0.227(0.3)	1.017	(2.765)

Table B.6: Comparison of average relative errors for five days prediction with DS6 ($l = 10$).

50% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.181	0.524	0.111	0.177	0.524(1.0)	2.412	
20	0.123	0.235	**0.057**	0.062	0.432(-1.7)	1.965	P:gamma
30	0.458	0.597	0.151	0.132	0.597(-2.5)	0.895	(0.077)
40	0.355	0.228	0.187	0.139	0.581(1.9)	1.052	E:gamma
50	0.121	0.142	0.305	0.286	0.781(-3.0)	1.063	(0.077)
60% observation ($t_n = 44$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.538	0.339	0.416	0.164	0.586(1.2)	0.963	
20	0.131	0.398	0.395	0.216	0.623(0.6)	0.619	P:gamma
30	0.607	0.425	0.288	0.187	0.728(1.9)	1.205	(0.111)
40	0.232	0.401	0.295	0.353	1.025(2.8)	1.369	E:gamma
50	0.375	0.644	0.195	**0.109**	0.844(-2.2)	1.366	(0.111)
70% observation ($t_n = 51$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.891	0.226	**0.087**	0.157	0.289(1.7)	0.774	
20	0.143	0.196	0.375	0.166	0.509(0.3)	1.325	P:txvmax
30	0.979	0.108	0.407	0.766	0.201(1.3)	0.963	(0.089)
40	0.147	0.318	0.249	0.631	0.529(-2.6)	0.489	E:gamma
50	0.189	0.096	0.114	0.177	0.869(-2.2)	1.259	(4.578)
80% observation ($t_n = 58$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.210	0.248	0.181	0.235	0.404(1.9)	1.502	
20	0.301	0.432	0.379	0.122	0.163(1.8)	1.269	P:txvmax
30	0.290	0.233	0.263	0.234	0.691(-2.6)	2.541	(**0.075**)
40	0.362	0.366	0.189	0.238	0.558(-2.7)	2.639	E:gamma
50	0311	0.776	0.160	0.244	0.744(1.2)	0.940	(4.814)

Table B.7: Comparison of average relative errors for five days prediction with DS6 ($l = 15$).

50% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.425	0.558	0.151	**0.115**	1.032(1.6)	3.256	
20	0.477	0.544	0.452	0.196	1.252(-1.1)	1.569	P:gamma
30	0.336	0.272	0.411	0.119	0.940(-1.5)	1.638	(0.117)
40	0.642	0.547	0.177	0.146	0.663(-1.9)	1.205	E:gamma
50	0.191	0.182	0.352	0.159	0.699(2.3)	1.235	(0.117)
60% observation ($t_n = 44$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.131	0.379	0.199	0.254	0.891(1.0)	1.023	
20	0.248	**0.129**	0.174	0.116	0.425(1.2)	0.967	P:gamma
30	0.188	0.371	0.156	0.079	0.226(-1.1)	1.265	(0.133)
40	0.285	0.396	0.166	0.813	1.923(-2.7)	1.023	E:gamma
50	0.414	0.515	0.125	0.248	0.296(-1.8)	1.963	(0.133)
70% observation ($t_n = 51$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.625	0.823	**0.119**	0.152	0.314(-1.7)	1.231	
20	0.442	0.471	0.375	0.155	0.261(1.3)	1.021	P:txvmax
30	0.861	0.604	0.407	0.386	0.369(1.9)	0.581	(0.121)
40	0.781	0.333	0.249	0.768	0.432(1.2)	0.633	E:gamma
50	0.099	0.205	0.129	0.199	0.722(-2.7)	1.068	(6.408)
80% observation ($t_n = 58$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.378	0.298	0.129	0.235	0.372(2.0)	1.943	
20	0.2871	0.386	0.131	0.292	0.574(-2.3)	2.041	P:txvmax
30	0.108	0445	0.363	0.127	0.314(1.1)	1.879	(**0.102)**
40	0.241	0.375	0.289	0.261	0.295(-2.0)	1.603	E:gamma
50	0.298	0.369	0.160	0.259	0.287(1.9)	1.256	(6.795)

Table B.8: Comparison of average relative errors for five days prediction with DS6 ($l = 20$).

50% observation ($t_n = 37$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.869	0.498	0.241	0.143	1.029(1.2)	1.2896	
20	0.731	0.848	0.471	0.133	0.6391.1)	2.059	P:gamma
30	0.599	0.288	0.174	0.163	0.592(2.3)	1.633	(0.145)
40	0.378	0.191	0.158	0.259	0.558(1.3)	0.986	E:gamma
50	**0.122**	0.193	0.128	0.185	0.569(-0.9)	1.369	(0.145)
60% observation ($t_n = 44$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.462	0.261	0.262	0.125	0.302(-1.1)	0.979	
20	0.327	0.267	0.181	0.193	0.291(1.9)	0.749	P:gamma
30	0.536	0.293	0.265	0.291	0.671(1.3)	1.025	(0.167)
40	0.216	0.270	0.199	0.195	0.369(0.7)	1.634	E:gamma
50	0.296	0.351	0.167	**0.137**	0.319(2.0)	0.606	(0.167)
70% observation ($t_n = 51$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	**0.133**	0.151	0.323	0.158	0.987(1.8)	0.789	
20	0.458	0.402	0.225	0.374	0.954(1.6)	1.429	P:txvmax
30	0.119	0.329	0.258	0.271	1.0325(1.2)	2.029	(0.149)
40	0.413	0.418	0.481	0.228	0.998(-2.0)	2.556	E:gamma
50	0.275	0.585	0.151	0.173	1.028(1.6)	3.026	(8.115)

Table B.9: Comparison of average relative errors for five days prediction with DS7 ($l = 5$).

50% observation ($t_n = 41$)

	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.298	0.311	**0.028**	0.053	0.241(1.9)	2.509	
20	0.374	0.473	0.166	0.271	0.625(2.0)	1.353	P:txvmax
30	0.607	0.221	0.231	0.044	1.023(-0.5)	1.472	(0.029)
40	0.409	0.689	0.219	0.316	0.809(1.6)	2.187	E:lxvmin
50	0.177	0.176	0.261	0.236	1.158(-2.8)	1.605	(2.952)

60% observation ($t_n = 49$)

	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.535	0.432	0.219	0.111	2.051(-1.2)	1.069	
20	0.922	0.130	0.326	0.396	0.559(-1.9)	1.623	P:txvmax
30	0.142	0.115	0.291	0.068	0.713(1.1)	0.987	(**0.007)**
40	0.183	0.176	0.094	0.189	1.044(-2.9)	3.089	E:lxvmin
50	0.302	0.138	0.133	0.279	0.454(-1.9)	1.023	(3.036)

70% observation ($t_n = 57$)

	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.929	0.823	0.181	0.473	0.263(1.6)	4.526	
20	0.374	0.893	0.134	0.144	0.497(0.9)	2.051	P:txvmax
30	0.216	0.756	0.212	0.148	0.198(0.3)	1.625	(**0.0031)**
40	0.409	0.491	0.044	0.146	1.311(1.1)	1.368	E:lxvmin
50	0.575	0.665	0.126	0.142	1.504(-2.4)	1.649	(2.852)

80% observation ($t_n = 65$)

	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.525	0.432	0.096	0.121	0.255(1.1)	2.421	
20	0.496	0.089	0.125	1.020	1.089(-2.9)	1.754	P:txvmax
30	0.223	0.854	0.496	0.569	0.854(-2.0)	1.739	(0.043)
40	0.189	0.195	0.176	**0.041**	1.596(0.5)	3.800	E:llogist
50	0.453	0.138	0.369	0.127	1.087(1.1)	2.761	2.725)

90% observation ($t_n = 73$)

	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.635	1.043	0.422	0.329	0.289(-2.4)	1.584	
20	0.133	1.065	0.118	0.315	0.242(0.5)	2.683	P:txvmax
30	0.861	1.025	0.311	0.395	0.169(-2.7)	1.609	(**0.004)**
40	0.496	1.106	0.052	0.411	1.4061(2.0)	1.107	E:gamma
50	1.082	1.021	0.067	0.126	0.523(-2.3)	1.517	(2.948)

Table B.10: Comparison of average relative errors for five days prediction with DS7 ($l = 10$).

50% observation ($t_n = 41$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.256	0.440	0.131	**0.041**	0.461(1.3)	2.243	
20	0.266	0.393	0.067	0.071	0.163(2.0)	2.249	P:txvmax
30	0.278	0.189	0.095	0.098	1.297(-1.6)	1.162	(0.045)
40	0.828	0.615	0.051	0.108	0.512(-1.6)	1.346	E:lxvmin
50	0.154	0.162	0.202	0.116	1.054(-3.0)	1.696	(5.356)
60% observation ($t_n = 49$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.138	0.317	0.051	0.089	1.051(2.0)	1.479	
20	0.390	0.064	0.096	0.406	0.487(0.9)	1.516	P:txvmax
30	**0.018**	0.629	0.077	0.411	1.162(-1.3)	0.962	(0.021)
40	0.296	0.608	0.135	0.035	1.954(-3.0)	3.581	E:lxvmin
50	0.199	0.159	0.148	0.139	0.394(-2.9)	1.521	(5.481)
70% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.399	0.906	0.110	0.088	1.021(1.5)	1.259	
20	0.948	0.812	**0.042**	0.257	0.981(1.6)	4.259	P:txvmax
30	0.643	0.607	0.121	0.273	0.887(-1.3)	3.509	(0.053)
40	0.654	0.993	0.081	0.279	1.201(1.6)	2.641	E:lxvmin
50	0.455	0.566	0.123	0.332	1.891(-2.9)	2.031	(5.061)
80% observation ($t_n = 65$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.196	0.271	0.099	0.116	0.293(1.7)	1.513	
20	0.056	0.161	0.122	0.251	1.243(0.1)	1.611	P:txvmax
30	0.311	0.435	0.121	0.213	0.336(0.0)	2.390	(0.052)
40	0.421	0.663	**0.048**	0.091	0.821(0.5)	1.450	E:llogist
50	0.333	0.821	0.078	0.193	0.821(-1.9)	1.426	(4.971)

Table B.11: Comparison of average relative errors for five days prediction with DS7 ($l = 15$).

50% observation ($t_n = 41$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.275	0.191	0.437	0.451	0.9851(0.3)	1.021	
20	0.466	0.336	0.159	0.147	0.163(2.0)	1.032	P:txvmax
30	0.736	0.263	0.095	**0.038**	1.297(-1.6)	1.234	(0.052)
40	0.540	0.623	0.051	0.069	0.512(-1.6)	0.963	E:lxvmin
50	0.098	0.131	0.263	0.126	1.054(-3.0)	0.639	(5.356)
60% observation ($t_n = 49$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.189	0.221	0.421	0.198	1.981(-2.0)	1.686	
20	0.133	0.311	0.096	0.420	0.398(1.1)	1.299	P:txvmax
30	0.265	0.411	0.077	0.321	1.631(-1.2)	2.158	(**0.028**)
40	0.161	0.069	0.135	0.078	1.023(1.2)	1.116	E:lxvmin
50	0.181	0.163	0.148	0.344	1.389(-3.0)	1.389	(7.104)
70% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	**0.049**	0.264	0.217	0.065	1.649(0.0)	1.686	
20	0.255	0.524	0.315	0.162	0.564(1.6)	0.391	P:txvmax
30	0.275	0.199	0.381	0.233	0.596(-1.3)	1.158	(0.053)
40	0.618	0.541	0.199	0.293	1.116(1.0)	1.484	E:lxvmin
50	0.978	0.407	0.196	0.461	1.389(2.0)	2.027	(5.061)
80% observation ($t_n = 65$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.098	0.254	0.421	0.065	0.963(0.1)	2.632	
20	0.274	0.526	0.315	0.162	1.244(-2.6)	1.302	P:txvmax
30	0.574	0.861	0.164	0.233	1.269(0.3)	1.205	(**0.056**)
40	0.556	0.424	0.154	0.093	0.562(-2.5)	1.211	E:llogist
50	0.297	0.337	0.195	0.153	0.961(2.0)	1.302	(7.208)

Table B.12: Comparison of average relative errors for five days prediction with DS7 ($l = 20$).

50% observation ($t_n = 41$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.395	0.112	0.272	0.249	0.946(-1.2)	0.963	
20	0.253	0.741	0.140	**0.059**	0.827(-2.2)	1.632	P:lxvmax
30	0.516	0.170	0.162	0.264	1.063(-2.7)	1.258	(0.063)
40	0.156	0.393	0.321	0.241	0.632(1.8)	1.421	E:lxvmin
50	0.147	0.126	0.287	0.352	1.961(2.0)	2.036	(10.097)
60% observation ($t_n = 49$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.079	0.202	0.211	0.073	0.627(-2.3)	1.234	
20	0.389	0.253	0.253	0.103	0.837(1.0)	0.961	P:txvmax
30	0.689	0.172	0.083	0.321	0.279(-1.9)	1.603	(0.065)
40	0.891	0.919	0.422	0.173	0.887(1.9)	1.962	E:lxvmin
50	0.995	0.482	**0.063**	0.155	1.063(-2.7)	2.633	(9.858)
70% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.228	0.093	0.104	0.481	1.021(1.2)	1.532	
20	0.263	0.208	0.107	0.552	0.563(1.7)	1.471	P:txvmax
30	0.145	0.296	0.205	0.395	0.896(-2.3)	1.204	(0.090)
40	0.384	0.509	0.301	**0.084**	0.635(-3.0)	1.020	E:lxvmin
50	0.142	0.353	0.422	0.723	2.311(-2.7)	0.784	(9.14)

Table B.13: Comparison of average relative errors for five days prediction with DS8 ($l = 5$).

50% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.261	0.281	0.191	0.196	0.289(1.1)	2.589	
20	0.178	0.235	**0.131**	0.191	0.396(0.7	2.280	P:lxvmax
30	0.241	0.298	0.183	0.296	0.214(0.5)	1.277	(0.139)
40	0.394	0.307	0.384	0.503	0.323(1.1)	1.399	E:gamma
50	0.141	0.143	0.220	0.339	0.639(2.0)	1.148	(1.738)
60% observation ($t_n = 68$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.019	0.747	0.101	0.189	1.213(0.6)	1.495	
20	0.094	0.643	0.194	0.258	0.603(1.0)	1.893	P:lxvmax
30	0.205	0.533	0.412	0.625	0.589(1.3)	0.791	(**0.045**)
40	0.258	0.651	0.225	0.281	0.521(1.2)	1.498	E:lxvmin
50	0.269	0.441	0.205	1.032	1.204(2.0)	1.364	(2.051)
70% observation ($t_n = 80$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.466	0.117	0.213	0.218	0.433(-1.6)	0.649	
20	0.227	0.245	0.139	0.059	0.396(-0.9)	2.489	P:txvmax
30	0.647	0.921	0.313	0.091	0.638(-0.3)	1.212	(**0.018**)
40	0.361	0.461	0.163	0.245	1.293(1.8)	0.962	E:lxvmin
50	0.163	0.221	0.254	0.234	0.562(-2.3)	1.036	(3.125)
80% observation ($t_n = 91$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.113	0.457	0.401	**0.031**	0.397(1.9)	3.941	
20	0.366	0.229	0.278	1.020	0.741(-1.3)	1.277	P:lxvmax
30	0.422	0.224	0.341	0.569	0.661(-2.3)	1.194	(0.032)
40	0.296	0.279	0.234	0.116	1.524(-0.5)	1.432	E:lxvmin
50	0.305	0.164	0.063	0.410	1.711(1.3)	1.491	(2.931)
90% observation ($t_n = 103$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.445	0.227	0.296	0.309	1.023(1.3)	1.054	
20	0.552	0.496	0.369	**0.201**	1.361(1.3)	1.631	P:tlogist
30	0.451	0.351	0.313	0.228	1.061(-2.7)	1.619	(0.218)
40	0.415	0.223	0.363	0.312	1.011(2.2)	1.152	E:lxvmin
50	0.391	0.921	0.296	0.840	0.633(-2.8)	0.963	(0.299)

Table B.14: Comparison of average relative errors for five days prediction with DS8 ($l = 10$).

50% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.561	0.422	0.402	0.364	1.122(1.5)	2.209	
20	0.378	0.341	0.319	0.457	1.115(0.7)	1.552	P:exp
30	0.241	0.493	0.301	0.169	1.429(0.5)	1.493	(1.061)
40	0.269	0.429	0.397	0.523	**0.171(1.1)**	1.477	E:gamma
50	0.489	0.354	0.359	0.382	1.363(-2.3)	0.616	(2.595)
60% observation ($t_n = 68$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	1.315	0.663	0.181	0.088	0.941(0.8)	1.044	
20	0.256	0.919	0.236	0.169	0.912(1.1)	1.244	P:lxvmax
30	0.179	0.884	0.214	**0.068**	0.369(-1.3)	0.961	(0.072)
40	0.198	0.931	0.271	0.092	1.021(-2.2)	2.369	E:lxvmin
50	1.180	0.935	0.152	0.181	0.931(-3.0)	1.458	(3.768)
70% observation ($t_n = 80$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.466	0.269	**0.085**	0.257	1.513(-2.2)	1.393	
20	0.227	0.879	0.158	0.188	1.026(-2.9)	3.201	P:txvmax
30	0.647	0.507	0.169	0.556	0.238(0.0)	1.899	(0.091)
40	0.361	0.361	0.131	0.441	1.201(1.8)	1.128	E:lxvmin
50	0.163	0.931	0.192	0.361	1.622(-2.8)	0.963	(7.881)
80% observation ($t_n = 91$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.549	0.448	0.082	0.314	0.996(1.3)	1.289	
20	0.551	0.654	0.237	0.639	0.996(-2.6)	1.321	P:lxvmax
30	0.214	0.941	0.118	0.059	0.336(-3.0)	1.116	(**0.045**)
40	0.263	0.198	0.211	0.991	1.011(0.5)	1.966	E:lxvmin
50	0.323	0.133	0.418	0.553	1.211(1.6)	1.023	(4.884)
90% observation ($t_n = 103$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.369	0.443	0.153	0.369	0.596(-2.3)	0.962	
20	0.695	0.123	0.299	0.194	0.336(1.9)	1.951	P:exp
30	0.336	0.512	0.414	0.148	0.667(-1.9)	2.622	(**0.084**)
40	0.456	0.772	0.174	0.322	0.629(-2.3)	1.189	E:lxvmin
50	0.336	0.663	0.095	0.115	0.624(-3.0)	1.261	(5.206)

Table B.15: Comparison of average relative errors for five days prediction with DS8 ($l = 15$).

50% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.289	0.941	**0.096**	0.231	0.699(1.4)	3.014	
20	0.994	1.213	0.105	0.421	1.063(-0.7)	3.217	P:lxvmax
30	0.841	0.748	0.161	0.541	0.962(-3.0)	1.032	(0.310)
40	0.554	0.339	0.169	0.221	0.663(-3.0)	1.231	E:gamma
50	0.336	0.421	0.332	0.223	0.921(-2.1)	1.921	(3.491)
60% observation ($t_n = 68$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.636	0.179	0.335	0.228	0.526(-2.8)	2.0974	
20	1.012	0.237	0.287	0.269	0.316(-1.3)	1.313	P:lxvmax
30	0.112	**0.067**	0.154	0.621	1.321(1.3)	1.208	(0.088)
40	0.603	0.172	0.332	0.186	0.963(-2.7)	1.394	E:lxvmin
50	0.443	0.485	0.269	0.171	1.014(0.0)	2.298	(5.546)
70% observation ($t_n = 80$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.306	0.259	0.642	0.189	0.963(2.1)	1.122	
20	0.578	0.366	0.198	0.324	0.984(1.1)	2.650	P:txvmax
30	0.347	0.402	0.594	0.232	0.778(-1.0)	2.482	(0.091)
40	0.258	0.605	0.377	**0.089**	0.633(0.8)	1.029	E:lxvmin
50	0.151	0.348	0.159	0.285	0.691(1.3)	1.869	(7.881)
80% observation ($t_n = 91$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.536	0.271	0.366	0.276	0.697(1.7)	1.501	
20	0.311	0.399	0.296	0.099	0.552(0.3)	1.059	P:txvmax
30	0.251	0.732	0.033	0.261	0.732(-2.5)	1.027	(**0.018**)
40	0.636	0.266	0.246	0.159	1.463(-1.5)	2.029	E:lxvmin
50	0.412	0.480	0.223	0.192	0.477(0.0)	1.048	(3.125)

Table B.16: Comparison of average relative errors for five days prediction with DS8 ($l = 20$).

50% observation ($t_n = 57$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.415	**0.134**	0.172	0.231	0.629(-1.1)	2.414	
20	0.894	0.229	0.195	0.508	0.335(1.1)	1.210	P:lxvmax
30	0.904	0.248	0.261	0.288	0.393(2.0)	3.141	(0.347)
40	0.687	0.169	0.369	0.296	0.567(0.0)	1.399	E:gamma
50	0.345	0.271	0.221	0.255	0.306(-3.0)	2.431	(4.334)
60% observation ($t_n = 68$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.585	0.172	0.487	**0.098**	0.994(1.2)	2.033	
20	0.874	0.508	0.178	0.205	0.945(1.1)	1.030	P:lxvmax
30	0.521	0.159	0.251	0.116	1.074(-0.7)	1.019	(0.105)
40	0.633	0.254	0.433	0.252	0.708(1.1)	1.006	E:lxvmin
50	0.381	0.584	0.336	0.141	0.366(1.4)	1.028	(7.323)
70% observation ($t_n = 80$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.436	**0.088**	0.104	0.235	0.543(1.8)	1.33	
20	0.195	0.247	0.201	0.323	0.944(-1.1)	1.421	P:txvmax
30	0.612	0.161	0.141	0.350	0.748(0.0)	2.358	(0.091)
40	0.607	0.441	0.285	0.332	0.709(-0.8)	1.723	E:lxvmin
50	0.741	0.504	0.173	0.411	0.504(1.1)	2.391	(7.981)
80% observation ($t_n = 91$)							
	MIMO						SRGM
k	AT1	AT2	FT	BT	BoxCox(Best λ)	Normal	Best Model
10	0.158	0.509	0.179	0.096	0.359(1.1)	2.398	
20	0.268	0.979	0.214	0.254	0.669(-2.3)	1.269	P:txvmax
30	0.288	0.604	0.146	0.266	0.604(1.5)	1.259	(0.104)
40	0.291	0.637	0.346	**0.101**	0.995(1.3)	1.332	E:lxvmin
50	0632	0.335	0.124	0.186	0.497(0.8)	2.303	(9.154)

Bibliography

[1] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of competing software reliability predictions," *IEEE Transactions on Software Engineering*, **SE-12** (9), pp. 950–967 (1986).

[2] G. Abaei, and A. Selamat, "Analysis of software fault prediction models using machine learning techniques," *International Journal of Computer & Information Science*, **13** (2), pp. 29–36 (2012).

[3] J. A. Achcar, D. K. Dey, and M. Niverthi, "A Bayesian approach using nonhomogeneous Poisson processes for software reliability models," *Frontiers in Reliability* (A. P. Basu, K. S. Basu and S. Mukhopadhyay, eds.), pp. 1–8 (1998).

[4] F. J. Anscombe, "The transformation of Poisson, binomial and negative binomial data," *Biometrika*, **35** (3/4), pp. 246–254 (1948).

[5] M. S. Bartlett, "The square root transformation in the analysis of variance," *Journal of the Royal Statistical Society*, **3** (1), pp. 68–78 (1936).

[6] E. K. Blum, and L. K. Li, "Approximation theory and feedforward networks," *Neural Networks*, **4** (4), pp. 511–515 (1991).

[7] K. Y. Cai, *Software Defect and Operational Profile Modeling*, *Kluwer Academic Publishers*, Boston (1998).

[8] G. E. P. Box, and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society, Series B (Methodological)*, **26** (2), pp. 211–252 (1964).

[9] H. Cheng, P.-N. Tan, J. Gao and J. Scripps, "Multistep-ahead time series prediction," *Advances in Knowledge Discovery and Data Mining* (W. K. Ng, M. Kitsuregawa, and J. Li, eds.), LNAI **3918**, pp. 765–774 (2006) .

[10] S. R. Dalal, and A. A. McIntosh, "When to stop testing for large software systems with changing code," *IEEE Transactions on Software Engineering*, **20** (4), pp. 318–323 (1994).

[11] T. Dohi, Y. Nishio, and S. Osaki, , "Optimal software release scheduling based on artificial neural networks," *Annals of Software Engineering*, **8**, pp. 167–185 (1999).

[12] M. Fisz, "The limiting distribution of a function of two independent random variables and its statistical application," *Colloquium Mathematicum*, **3**, pp. 138–146 (1955).

[13] F. Fnaiech, N. Fnaiech, and M. Najim, "A new feedforward neural network hidden layer neuron pruning algorithm," *IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '01)*, (2001).

[14] A. L. Goel, and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measuress," *IEEE Transactions on Reliability*, **R-28** (3), pp. 206–211 (1979).

[15] A. L. Goel, "Software reliability models: assumptions, limitations and applicability," *IEEE Transactions on Software Engineering*, **SE-11** (12), pp. 1411–1423 (1985).

[16] S. S. Gokhale, and K. S. Trivedi, "Log-logistic software reliability growth model," *Proceedings of The 3rd IEEE International High-Assurance Systems Engineering Symposium (HASE-1998)*, pp. 34–41, IEEE CPS (1998).

[17] Q. P. Hu, M. Xie, S. H. Ng, and G. Levitin, "Robust recurrent neural network modeling for software fault detection and correction prediction", *Reliability Engineering & System Safety*, **92** (3), pp. 332–340 (2007).

[18] J. T. G. Hwang, and A. A. Ding, " Prediction intervals for artificial neural

networks," *Journal of the American Statistical Association*, **92** (433), pp. 748–757 (1997).

[19] N. Karunanithi, D. Whitley, and Yashwant K., " Prediction of software reliability using connectionist models, " *IEEE Transactions on Software Engineering*, **SE-18**, pp. 563–574 (1992).

[20] N. Karunanithi, D. Whitley, and K. Yashwant, " Using neural networks in reliability prediction, " *IEEE Software*, **9**, pp. 53–59 (1992).

[21] N. Karunanithi, Y. K. Malaiya, and D. Whitley, "Prediction of software reliability using neural networks," *Proceedings of The 2nd International Symposium on Software Reliability Engineering (ISSRE-1991)*, pp. 124–130, IEEE CPS (1991).

[22] N. Karunanithi, and Y. K. Malaiya, "The scaling problem in neural networks for software reliability prediction," *Proceedings of The 3rd International Symposium of Software Reliability Engineering (ISSRE-1992)*, pp. 76–82, IEEE CPS (1992).

[23] N. Karunanithi, and Y. K. Malaiya, "Neural networks for software reliability engineering," *Handbook of Software Reliability Engineering* (M. R. Lyu, ed.), pp. 699–728, McGraw-Hill, New York (1996).

[24] T. M. Khoshgoftaar, D. L. Lanning, and A. S. Pandya, "A neural network modeling for detection of high-risk program," *Proceedings of The 4th International Symposium on Software Reliability Engineering (ISSRE-1993)*, pp. 302–309, IEEE CPS (1993).

[25] T. M. Khoshgoftaar, and R. M. Szabo, "Predicting software quality during testing using neural network models: a comparative study," *International Journal of Reliability, Quality and Safety Engineering*, **1**, pp. 303–319 (1994).

[26] T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, and S. J. Aud, "Application of neural networks to software quality modeling of a very large telecommunication system," *IEEE Transactions on Neural Networks*, **8** (4), pp. 902–909 (1997).

[27] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, "A comprehensive review of neural network-based prediction intervals and new advances," *IEEE Transactions on Neural Networks*, **22** (9), pp. 1341–1356 (2010).

[28] B. Littlewood, "Rationale for a modified Duane model," *IEEE Transactions on Reliability*, **R-33** (2), pp. 157–159 (1984).

[29] M. R. Lyu (ed.), *Handbook of Software Reliability Engineering*, *McGraw-Hill*, New York (1996).

[30] P. A. Lewis, and G. S. Shedler, " Simulation of nonhomogeneous Poisson processes by thinning," Technical Report, DTIC Document 1978.

[31] Y. W. Leung, "Optimal software release time with a given cost budget," *Journal of Systems and Software*, **17**, pp. 233–242 (1992).

[32] M. Makitalo and A. Foi " A closed-form approximation of the exact unbiased inverse of the Anscombe variance-stabilizing transformation," *IEEE Trans Image Process* **20** (9), pp. 2697–2698 (2011).

[33] R. Mahajana, S. K. Guptab, and R. K. Bedib, "Design of software fault prediction model using BR technique," *Procedia Computer Science*, **46**, pp. 849–858 (2015).

[34] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability, Measurement, Prediction, Application*, *McGraw-Hill*, New York (1987).

[35] D. J. C. MacKay, "The evidence framework applied to classification networks", *Neural Computation,* **4** (5), pp. 720–736 (1987).

[36] D. Nix, and A. Weigend, "Estimating the mean and variance of the target probabilty distribution," *Proceedings of The 1994 IEEE International Conference on Neural Networks*, **1**, pp. 55–60, IEEE CPS (1994).

[37] S. Noekhah, A. A. Hozhabri, and H. S. Rizi, "Software reliability prediction model based on ICA algorithm and MLP neural network," *Proceedings of The 7th Intenational Conference on e-Commerce in Developing Countries (ECDC-2013)*, pp. 1–15, IEEE CPS (2013).

[38] M. Ohba, "Infection S-shaped software reliability growth model," *Stochastic Models in Reliability Theory* (S. Osaki and Y. Hatoyama, eds.), pp. 144–165, Springer-Verlag, Heidelberg (1984).

[39] K. Ohishi, H. Okamura, and T. Dohi, "Gompertz software reliability model: estimation algorithm and empirical validation," *Journal of Systems and Software*, **82** (3), pp. 535–543 (2009).

[40] H. Okamura, T. Dohi, and S. Osaki, "Software reliability growth models with normal failure time distributions," *Reliability Engineering & System Safety*, **116**, pp. 135–141 (2013).

[41] H. Okamura, and T. Dohi, "SRATS: Software reliability assessment tool on spreadsheet," *Proceedings of The 24th International Symposium on Software Reliability Engineering (ISSRE-2013)*, pp. 100–117, IEEE CPS (2013).

[42] P.-F. Pai, and W.-C. Hong, "Software reliability forecasting by support vector machines with simulated annealing algorithms," *Journal of Systems and Software*, **79** (6), pp. 747–755 (2006).

[43] H. Pham, *Software Reliability*, Springer-Verlag, London (2000).

[44] H. Pham, and X. Zhang, "A software cost model with warranty and risk costs," *IEEE Transactions on Computers*, **48** (1), pp. 71-75 (1999).

[45] H. Pham, "Software reliability and cost models: Perspectives, comparison, and practice," *European Journal of Operational Researchy*, **149**, pp. 475-489 (2003).

[46] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Torner, "Evaluating long-term predictive power of standard reliability growth models on automotive systems," *Proceedings of The 24th International Symposium on Software Reliability Engineering (ISSRE-2013)*, pp. 228–237, IEEE CPS (2013).

[47] Y. Saito, and T. Dohi, "Software reliability assessment via non-parametric maximum likelihood estimation," *IEICE Transactions on Fundamentals*

of Electronics, Communications and Computer Sciences (A), **E98-A** (10), pp. 2042–2050 (2015).

[48] R. Sitte, "Comparison of software reliability growth predictions: neural networks vs. parametric recalibration," *IEEE Transactions on Reliability*, **48** (3), pp. 285–291 (1999).

[49] K. Sharma, R. Garg, C. K. Nagpal, and R. K. Garg, "Selection of optimal software reliability growth models using a distance based approach," *IEEE Transactions on Reliability*, **59** (2), pp. 266–276 (2010).

[50] Y.-S. Su, and C.-Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," *Journal of Systems and Software*, **80**, pp. 606–615 (2007).

[51] L. Tian, and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," *Reliability Engineering & System Safety*, **87**, pp. 45–51 (2005)

[52] L. Tian, and A. Noore, "On-line prediction of software reliability using an evolutionary connectionist model," *Journal of Systems and Software*, **77**, pp. 173–180 (2005).

[53] M. Xie, *Software Reliability Modelling*, *World Scientific, Singapore*, (1991).

[54] X. Xiao, and T. Dohi, "Wavelet shrinkage estimation for NHPP-based software reliability models," *IEEE Transactions on Reliability*, **62**(1), pp. 211–225 (2013).

[55] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, **R-32** (5), pp. 475–478 (1983).

[56] ftp://ftp.sas.com/pub/neural/FAQ3.html# **A-hu**

[57] B. Yang, X. Li, M. Xie, and F. Tan, "A generic data-driven software reliability model with model mining technique," *Reliability Engineering & System Safety*, **95** (6), pp 671–678 (2010).

[58] M. Zhao, and M. Xie, "On maximum likelihood estimation for a general non-homogeneous Poisson process," *Scandinavian Journal of Statistics*, **23** (4), pp. 597-607 (1996) .

[59] D. Tada, X. Xiao, and H. Yamamoto, "Wavelet shrinkage estimation using unbiased inverse transformation for software reliability assessment," *Proc. of The 7th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modeling (APARM2016)*, pp. 493–500 (2016).

[60] R. D. D. Veaux, J. Schumi, J. Schweinsberg, and L. H. Ungar, "Prediction intervals for neural networks via nonlinear regression, " *Technometrics*, **40** (4), pp. 273-282 (1998).

[61] J. Park, N. Lee, and J. Baik, "On the long-term predictive capability of data-driven software reliability model: an empirical evaluation," *Proceedings of The 25th International Symposium on Software Reliability Engineering (ISSRE-2014)*, pp. 45–54, IEEE CPS (2014).

[62] T. Kaneishi, and T. Dohi, "Software reliability modeling and evaluation under incomplete knowledge on fault distribution," *Proceedings of The 7th IEEE International Conference on Software Security and Reliability (SERE-2013)*, pp. 3–12, IEEE CPS (2013).

[63] Y. Saito, T. Moroga, and T. Dohi, "Optimal software release decision based on nonparametric inference approach," *Journal of the Japan Industrial Management Association*, **66**, pp. 396–405 (2016).

[64] M. C.van Pul, "Asymptotic properties of a class of statistical models in software reliability," *Scandinavian Journal of Statistics*, **19** (3), pp. 225–253 (1992).

[65] H. Joe, "Statistical inference for general-order-statistics and nonhomogeneous-Poisson-process software reliability models," *IEEE Transactions on Software Engineering*, **15** (11), pp. 1485–1490 (1989).

[66] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*,**(61)**, pp. 85–117 (2015).

[67] S. A. Sherer, “Software fault prediction,” *Journal of Systems and Software*, **29** (2), pp. 97–105 (1995).

[68] D. F. Specht, “Probabilistic neural network,” *Neural Networks*, **(3)**, pp. 109–118 (1990).

[69] R. Caruana, “Generalization vs. net size,” *Neural Information Processing Systems*,Tutorial.

[70] P. Guo, and M. R. Lyu, “A pseudo inverse Learning algorithm for feed forward neural networks with stacked generalization applications to software reliability growth data,” *Neurocomputing*,**(56)** pp. 101–121 (2004).

[71] P. K. Kapurr, S. K. Khatri, and D. N. Goswami, “A generalized dynamic integrated software reliability growth model based on neural-network approach,” *in Proceedings of The International Conference on Reliability, Safety and Quality Engineering*, pp. 831–838 (2008).

[72] X. Zhang, and H. Pham, “A software cost model with error removal times and risk costs,” *International Journal of Systems Science*, **29** (4), pp. 435-442 (1998).

[73] Y. Liu, J. A. Starzyk, and Z. Zhu, “Optimizing number of hidden neurons in neural networks,” *in Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*, pp. 121-126 (2007).

Publication List of the Author

[A1] B. Momotaz, T. Dohi, Estimating prediction interval of cumulative number of software faults using back propagation algorithm, *International Journal of Computer and Information Science*, vol. 17, no. 2, pp. 25–34, May 2016.

[A2] Begum, M, T. Dohi, A neuro-based software fault prediction with Box-Cox ower transformation, *Journal of Software Engineering and Applications*, vol. 10, no. 3, pp. 288–309, March 2017.

[A3] M. Begum, T. Dohi, Optimal stopping time of software system test via artificial neural network with fault count data, *Journal of Quality in Maintenance Engineering*, (accepted).

[A4] M. Begum, T. Dohi, Prediction interval of cumulative number of software faults using multilayer perceptron, *Applied Computing & Information Technology (R. Lee, ed.), Studies in Computational Intelligence*, vol. 619, pp. 43–58, Springer International Publishing, Switzerland, 2016.

[A5] M. Begum, T. Dohi, Optimal software release decision via artificial neural network approach with bug count data, *Proceedings of The 7th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modeling (APARM 2016)*, pp. 17–24, McGraw-Hill, Taiwan, Aug 2016.

[A6] M. Begum, T. Dohi, Prediction interval of cumulative number of software faults using multilayer perceptron, *The 3rd International Conference on Applied Computing & Information Technology (ACIT2015)*, Okayama, Japan, July 12–16, 2015

[A7] M. Begum, T. Dohi, Prediction interval of software fault-detection time using back propagation algorithm, *The 2015 (66th) Chugoku-branch Joint Convention of Institutes of Electrical and Information Engineers (RENTAI 2015)*, Yamaguchi, Japan, October 17, 2015 (Received award)

[A8] M. Begum, T. Dohi, Optimal software release decision via back propagation algorithm, *The 2016 (67th) Chugoku-branch Joint Convention of Institutes of Electrical and Information Engineers (RENTAI 2016)*, Higashi-Hiroshima, Japan, October 22, 2016