2005

Dynamic learning with neural networks and support vector machines

Liang Tian
West Virginia University

Dynamic Learning with Neural Networks and Support Vector Machines

Liang Tian

Dissertation submitted to the

College of Engineering and Mineral Resources

at West Virginia University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer and Information Science

Afzel Noore, Ph.D., Chair

Powsiri Klinkhachorn, Ph.D.

Bojan Cukic, Ph.D.

Katerina Goseva-Popstojanova, Ph.D.

Wafik Iskander, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia

2005

Keywords: Neural Networks, Genetic Algorithm, Support Vector Machines,

Software Reliability Prediction, Short-Term Load Forecasting.

ABSTRACT

Dynamic Learning with Neural Networks and Support Vector Machines

Liang Tian

Neural network approach has proven to be a universal approximator for nonlinear continuous functions with an arbitrary accuracy. It has been found to be very successful for various learning and prediction tasks. However, supervised learning using neural networks has some limitations because of the black box nature of their solutions, experimental network parameter selection, danger of overfitting, and convergence to local minima instead of global minima. In certain applications, the fixed neural network structures do not address the effect on the performance of prediction as the number of available data increases. Three new approaches are proposed with respect to these limitations of supervised learning using neural networks in order to improve the prediction accuracy.

Dynamic learning model using evolutionary connectionist approach. In certain applications, the number of available data increases over time. The optimization process determines the number of the input neurons and the number of neurons in the hidden layer. The corresponding globally optimized neural network structure will be iteratively and dynamically reconfigured and updated as new data arrives to improve the prediction accuracy. *Improving generalization capability using recurrent neural network and Bayesian regularization.* Recurrent neural network has the inherent capability of developing an internal memory, which may naturally extend beyond the externally provided lag spaces. Moreover, by adding a penalty term of sum of connection weights, Bayesian regularization approach is applied to the network training scheme to improve the generalization performance and lower the susceptibility of overfitting. *Adaptive prediction model using support vector machines.* The learning process of support vector machines is focused on minimizing an upper bound of the generalization error that includes the sum of the empirical training error and a regularized confidence interval, which eventually results in better generalization performance. Further, this learning process is iteratively and dynamically updated after every occurrence of new data in order to capture the most current feature hidden inside the data sequence.

All the proposed approaches have been successfully applied and validated on applications related to software reliability prediction and electric power load forecasting. Quantitative results show that the proposed approaches achieve better prediction accuracy compared to existing approaches.

To my parents and my wife

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my committee chair, Dr. Afzel Noore, for his support, encouragement, and inspiration during the past five years. I could not have accomplished my Ph.D. dissertation without his enlightening guidance. He not only taught me the research methodology through his own knowledgeable and creative experiences, but also, he is and will continue to be a mentor in my life. Thank you for everything.

In addition, I would like to thank the members of my committee, Dr. Powsiri Klinkhachorn, Dr. Bojan Cukic, Dr. Katerina Goseva-Popstojanova, and Dr. Wafik Iskander for their constructive suggestions and invaluable guidance in preparation of this dissertation.

I would like to express my appreciation to Dr. Mike Henry, Dr. James Mooney, Dr. Raymond Morehead, and Dr. Ali Feliachi for their support in CSEE Dept. at West Virginia University.

I would like to thank Wei Yuan from University of Illinois at Urbana-Champaign, Shan Jiang from Louisiana State University, and Yan Sun from George Mason University for their help.

My Ph.D. research project was sponsored in part by a US DOE/EPSCoR West Virginia State Implementation Award through the Advanced Power & Electricity Research Center at West Virginia University.

Last, but by no means least, I wish to express my gratitude to my parents, for their encouragement and affection, and to my wife, Jiqin, for her understanding and

endless love. Without their love, I could not have gone this far.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AE	Average Relative Error
CASRE	Computer-Aided Software Reliability Estimation
D − ENN	Dynamic Evolutionary Neural Networks (Connectionist)
FFNN	Feed–Forward Neural Networks
GA	Genetic Algorithms
MLE	Maximum Likelihood Estimation
MSE	Mean Square Error
NHPP	Non–Homogeneous Poisson Process
NN	Artificial Neural Networks
RBFNN	Radial Basis Function Neural Networks
RE	Relative Error
RMSE	Root Mean Square Error
RNN	Recurrent Neural Networks
RNN + BR	Recurrent Neural Networks with Bayesian Regularization
SRGM	Software Reliability Growth Model
SRM	Structural Risk Minimization
SSE	Sum of Squared Error
STLF	Short–Term Load Forecasting
SVM	Support Vector Machines

Chapter 1

Introduction

1.1 Motivation

Artificial neural networks are powerful methods for classification and function approximation. Neural networks have better capabilities of fault tolerance, robustness, and adaptability compared to traditional analytical models. However, neural networks have some limitations such as experimental network parameter selection, danger of overfitting, and convergence to local minima instead of global minima.

Optimization of neural network structure design to improve forecasting performance is still a problem [1, 2, 3]. Although researchers have attempted to address these related issues, there is no standard method of designing the neural network structure to solve a specific problem efficiently [4]. Trial-and-error methods have been employed, which usually involve substantial amount of computation. Genetic algorithm is a powerful random search technique to deal with optimization problems [5]. It can discover the optimized network structure through global random searching process [6, 7, 8, 9, 10].

Most of the existing neural network approaches use a static structure with a predetermined number of input neurons and a predetermined number of hidden neurons

that are established during training. In certain applications, the number of available data increases over time. The fixed network structure does not address the effect on the performance of prediction as the number of data increases, and thus may not provide the best results.

Most of the supervised learning using neural networks adopts the gradient descent based back-propagation learning scheme to implement the empirical risk minimization principle, which only minimizes the mean square error during the training process and thus improves the training accuracy. In this case, the focus of the training process is model fitting and tends to cause overfitting. The error on the training data set is driven to a very small value for known data, but when out-of-sample data are presented to the network, the error is unpredictably large, which yields limited generalization capability. At the same time, due to the inherent nature of the gradient descent based learning scheme, getting stuck into local minima instead of global minima becomes common.

In the following sections, three new approaches are proposed with respect to the above-mentioned limitations of supervised learning using neural networks in order to obtain improvements. The proposed approaches will be tested and validated using two different types of applications. The first application is related to software reliability prediction, and the second application is electric power load short-term forecasting.

1.2 Research Objectives

This research focuses on improving both static and dynamic learning and generalization capabilities of different types of neural networks applied to forecasting. The research objectives are:

1. Design dynamic learning model using evolutionary connectionist approach.

2. Improve generalization capability using recurrent neural network and Bayesian regularization.

3. Develop adaptive prediction model using support vector machines.

1.3 Organization of Dissertation

This dissertation is organized as follows. Chapter 2 presents the related work. It focuses on overview of neural networks and genetic algorithm optimization techniques. In addition, it summarizes the related research in neural network based software reliability assessment and electric short-term load forecasting. Chapter 3 introduces the proposed *dynamic learning model using evolutionary connectionist approach*, and validation results when applied to software reliability prediction and short-term load forecasting. *Improving generalization capability using recurrent neural network and Bayesian regularization approach* is proposed and validated in Chapter 4. Chapter 5 outlines *adaptive prediction model using support vector machines approach*, and the corresponding validation results. Chapter 6 shows some extended experiment results and related discussion. Conclusions and future work are summarized in Chapter 7.

Chapter 2

Related Work

2.1 Neural Network Modeling

2.1.1 Artificial Neurons

A typical artificial neuron is shown in Fig. 2.1, with inputs x_i and weights w_i. The weighted summation function is denoted by:

$$v = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

As shown in Fig. 2.2, we use the typical sigmoidal function as the nonlinear activation function, which is denoted by:

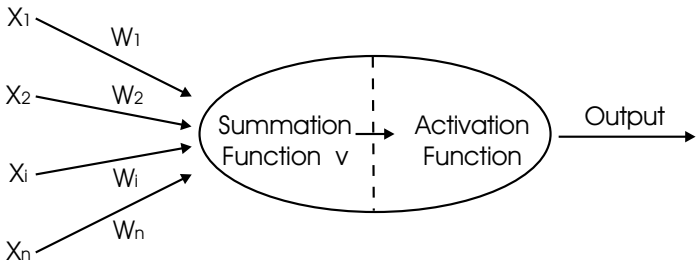

Figure 2.1: Schematic representation of an artificial neuron

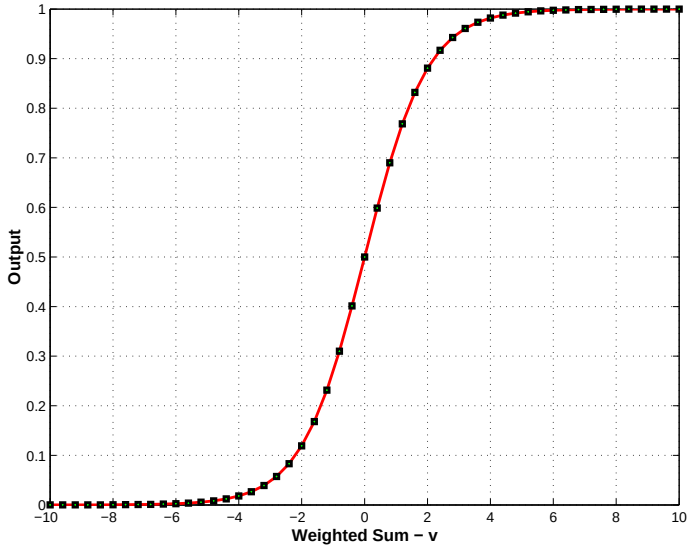

Figure 2.2: Illustration plot of sigmoidal activation function

$$\text{output} = \frac{1}{1 + e^{-v}} \tag{2.2}$$

2.1.2 Artificial Neural Networks

An artificial neural network can be defined as [11]:

> A data processing system consisting of a large number of simple, highly interconnected processing element (artificial neuron) in an architecture inspired by the structure of the cerebral cortex of the brain.

These processing elements are usually organized into a sequence of layers with connections between the layers. Multilayer feed-forward neural network is one of the most commonly used networks in various applications. As an example, the three-layer feed-forward neural network architecture is shown in Fig. 2.3.

w_{ij} is the weight connecting the ith input neuron and the jth hidden neuron, where $1 \leq i \leq k$, and $1 \leq j \leq m$. w'_j is the weight connecting the jth hidden neuron

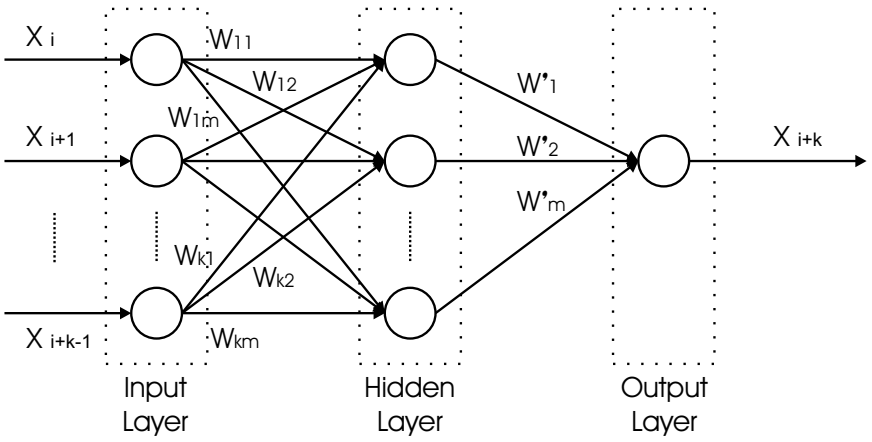

Figure 2.3: Three-layer feed-forward neural network

and the output neuron, where $1 \leq j \leq m$.

2.1.3 Features of Artificial Neural Networks

Neural networks learn by example and they constitute a distributed associative memory. Also, they are fault tolerant and capable of pattern recognition [12].

Advantages of Neural Networks

- Learning from data, mimicking human learning ability
- Can approximate any multivariate nonlinear function
- Robust to the presence of noisy data
- Parallel structure and can be easily implemented in hardware
- Can be applied to broad classes of tasks

Limitations of Neural Networks

- Long training or learning time
- Do not explain basic internal relations of physical variables, and do not increase our knowledge about the process
- Prone to bad generalizations due to large number of weights; tendency to overfit the data; limited performance on unseen data

- Little guidance is offered about neural network structure or optimization procedure

2.2 Genetic Algorithms and Evolutionary Computing

2.2.1 Basic Structure of Genetic Algorithms

Genetic Algorithms (GA) are global search and optimization techniques modeled from natural genetics, exploring search space by incorporating a set of candidate solutions in parallel [13]. GA maintains a population of candidate solutions where each candidate solution is coded as a binary string called chromosome. A chromosome encodes a parameter set for a group of variables being optimized [13]. A set of chromosomes forms a population, which is ranked by a fitness evaluation function. The fitness evaluation function provides information about how good each candidate solution is. This information guides the search of GA. More specifically, the fitness evaluation results determine the likelihood that a candidate solution is selected to produce candidate solutions in the next generation [13].

2.2.2 Mechanism of Evolutionary Computing

As shown in Fig. 2.4, the evolution from one generation to the next generation involves three steps [13, 14, 15, 2, 16, 1]:

- Fitness evaluation

  The current population is evaluated using the fitness evaluation function and ranked based on their fitness values.

- Selection

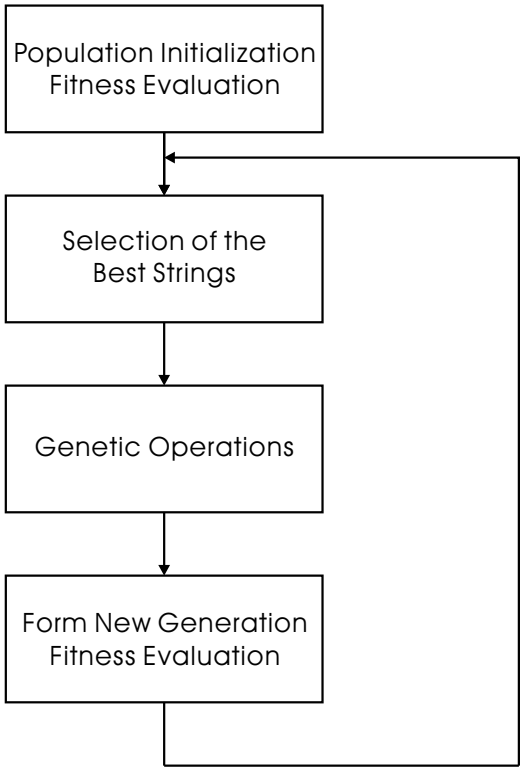

Figure 2.4: Illustration of genetic algorithm framework

GA stochastically select "parents" from the current population with a bias that better chromosomes are more likely to be selected. This process is implemented by using a selection probability that is determined by the fitness value.

- Reproduction

  GA reproduce "children" from selected "parents" using genetic operations, such as crossover or resemblance and mutation.

This cycle of fitness evaluation, selection, and reproduction terminates when an acceptable solution is found, a convergence criterion is met, or when a predetermined limit on the number of iterations is reached.

2.2.3 Comparison with Orthogonal Least Squares Optimization

The comparison between Genetic Algorithms and Orthogonal Least Squares Optimization is described as follows [12]:

(1) Description

- Genetic Algorithm

  Searches globally using a probabilistic random search technique analogous to the natural evolution for optimum solutions.

- Orthogonal Least Squares

  Searches locally, selecting from the given set of basis functions to find an optimal subset of basis function.

(2) Search Strategy

- Genetic Algorithm

  Employs a multipoint search strategy to continuously select the set of solutions with higher fitness value, which is similar to natural reproduction. The fittest survive whereas the rest are "disqualified". The whole selection procedure is carried out in a probabilistic random manner.

- Orthogonal Least Squares

  A set of basis functions is selected for a network from a previously defined set of basis functions that have varying shapes and locations. The selection of basis functions depends on the associated approximation levels of the basis function. The selection procedure maximally selects the basis functions with higher approximation levels to form a subset of bases.

(3) Search Space

- Genetic Algorithm

  This is a random probabilistic method that searches globally. There are no restrictions on the search space. If an optimal solution exists, GA is capable of finding it.

- Orthogonal Least Squares

  This is a structured search that only searches locally. Unless the global optimal solution is included in the set of basis functions, Orthogonal Least Squares is not capable of finding it.

(4) Efficiency

- Genetic Algorithm

  Although GA is powerful in finding the optimal solution, the path it takes to get to this solution is complicated and may not be repeatable because of the random nature of this technique. There are often several paths the optimization algorithm could take to arrive at the same solution, which makes this procedure time-consuming.

- Orthogonal Least Squares

  Orthogonal Least Squares does not guarantee an optimal solution but a solution close to it if the initial set of basis functions covers the input space adequately. The optimization is faster than GA, and the optimization procedure is easily repeatable because of the nature of the search.

2.3 Background of NN-based Software Reliability Prediction

Software reliability is defined as the probability of a failure free operation of software for a specified period of time in a given environment [17, 18, 19]. The best approach to evaluate software reliability quantitatively is to use software reliability models. A software reliability model is a set of mathematical equations that are used to describe the behavior of software failures with respect to time and predict software reliability performance such as the mean time between failures and the number of residual faults [20, 17, 21].

Software reliability models must cover two different types of situations. One is finding faults and fixing them, and the other is referring to "no fault removal". "No fault removal" actually means "deferred fault removal". When the failures are identified, the underlying faults will not be removed until the next release [17, 21]. This situation is simple and usually occurs during validation test and operation phase. Most of software reliability models deal with the process of finding and fixing faults that usually occur during software verification process. Thus, if it is assumed that fault removal process does not introduce new faults, the software reliability will increase with the progress of debugging. A software reliability model describing such fault detection and removal phenomenon is called a software reliability growth model [22, 23, 24].

The scope of this research is to propose new modeling approaches and apply them to software reliability growth prediction for validation purposes. Most of the existing analytical software reliability growth models depend on *a priori* assumptions about the nature of software faults and the stochastic behavior of software failure process [25, 26, 27, 28, 29, 30]. As a result, each model has a different predictive performance across various projects. A general model that can provide accurate predictions under multiple circumstances is most desirable [27, 28, 29]. It has been shown that a neural

network approach is a universal approximator for any non-linear continuous function with an arbitrary accuracy [26, 8]. The underlying failure process can be learned and modeled based on only failure history of a software system rather than based on *a priori* assumptions [27, 31]. Consequently, it has become an alternative method in software reliability modeling, evaluation and prediction. Karunanithi et al. [27, 28] were the first to propose a neural network approach for software reliability growth modeling. Adnan et al. [32, 33], Aljahdali et al. [34, 35], Ho et al. [36], Park et al. [29], and Sitte [37] have also made contributions to software reliability growth prediction using neural networks, and have obtained better results compared to the traditional analytical models with respect to predictive performance.

Most of the published literature used neural network to model the relationship between software failure time and the sequence number of failures. Some examples are: execution time as input and the corresponding accumulated number of defects disclosed as desired output [27, 28], and failure sequence number as input and failure time as desired output [29]. Recent studies focus on modeling software reliability based on time-lagged neural network structure. Aljahdali et al. [34] used the recent days' faults observed before the current day as multiple inputs to predict the number of faults initially resident at the beginning of testing process. Cai et al. [26] and Ho et al. [36] used the recent inter-failure time as multiple inputs to predict the next failure time.

The effect of both the number of input neurons and the number of neurons in hidden layers were determined using a selected range of predetermined values [26, 33]. For example, 20, 30, 40, and 50 input neurons were selected in Cai's experiment [26], while 1, 2, 3, and 4 input neurons were selected in Adnan's experiment [33]. The effect on the structure was studied by independently varying the number of input neurons or the number of neurons in hidden layers [26] instead of considering all possible combinations.

2.4 Background of NN-based Short-Term Load Forecasting

One of the most crucial requirements for the operation activities of power systems is short-term hourly load forecasting and the extension to several days in the future. With the recent world-wide deregulation of the power utility industry, improving the accuracy of short-term load forecasting (STLF) is becoming significant. It is driven by the changing structure of the power utility industry, newly introduced retailers, and other power marketing participants. As a result, highly accurate and reliable short-term load forecasting provides optimal energy transaction allocation, optimal scheduling plans and optimal bidding strategies under the new deregulation environment [38, 39, 40, 41, 42].

Most of the existing load forecasting methods are based on either time series models or curve-fitting procedures. The advantages of these models are easy physical interpretations of model parameters. However, due to the inherent linear characteristics, it seems inadequate for those models to discover the known highly nonlinear interrelationship among load data profile and the relationship between load data and some related variables such as temperature, humidity, and other weather factors. Recent studies have used artifical neural network for load forecasting due to its proven ability to be a universal approximator for any non-linear continuous function with an arbitrary accuracy [8]. Thus, this data-driven approach can make it easier to model the complex load forecasting problem when we have a large amount of data but at the same time, very little *a priori* knowledge about the system that generates the data [38, 40].

A number of research papers have reported successful experiments of applying neural network approach to short-term load forecasting compared to traditional methods with respect to predictive performance [40, 43]. Most of the published literature

used at least two or more variables as input variables to the neural network forecasting system. Some examples of input variables are: load and temperature [44, 45, 46]; load, temperature and humidity [42]; load, temperature, seasons and day type [47]; load, temperature, day type, and electricity price [41]; load, temperature, seasons, day type and day of week [48], etc. Each input variable could be a multi-dimensional vector representing the profile data, which increases the input nodes of the neural network. Chen et al. [41] used 100 input nodes to represent the profile data for input variables load, temperature, day type, and electricity price. It is very common to find many forecasting systems with huge amount of input nodes, which implies that too many parameters need to be estimated based on comparatively too few sample data points during the neural network training process, and thus may not yield satisfactory forecasting performance [40]. Using the input variable that has the largest impact on forecasting performance may be a good alternative.

Further, optimization of neural network structure design, including selecting the number of input variables, input nodes and the number of hidden neurons, to improve forecasting performance is becoming more and more important and desirable. Charytoniuk et al. [45] proposed an optimal input variable selection approach based on singular value decomposition techniques. Most effective delayed load inputs were selected by correlation analysis in Barghinia et al. [44]. Huang et al. [46] employed gray relational analysis for proper selection of input variables and input nodes in order to improve learning efficiency of neural network. Several reasonable choices of delayed load inputs of a particular day such as Monday-Sunday, were also suggested. Genetic algorithm can be used as an optimization search scheme to determine the optimal or near optimal network structure design. Srinivasan [49] successfully used a genetic algorithm to evolve the optimum neural network structure including the connecting weights between load, temperature input variables and the forecasted load during the training process. Tsao et al. [50] applied evolutionary programming to optimize the

weights and biases of a neural network. However, it is clear that there is no integrated selection or optimization approach considering both the number of input nodes, the number of neurons in the hidden layer, and all the possible combinations based on minimal number of input variables.

Chapter 3

Dynamic Learning Using Evolutionary Connectionist

3.1 Proposed Approach

In certain applications, the number of available data increases over time. The fixed neural network structures do not address the effect on the performance of prediction as the number of data increases. The proposed dynamic learning optimization process for evolutionary connectionist model (D - ENN) is described by the following procedure.

1. For every occurrence of new data, optimize the neural network structure by finding the optimal number of input neurons and the optimal or near-optimal number of neurons in the hidden layer using the genetic algorithm based on currently available history data.

2. Apply training data patterns to the optimized neural network structure until the neural network converges, and predict next-step data.

3. Repeat the above steps as new data arrive.

The fitness evaluation function is defined as:

$$fitness = \frac{1}{1 + err} \tag{3.1}$$

$$err = \sum_{i=1}^{p} \frac{|\widehat{x}_i - x_i|^2}{p} \tag{3.2}$$

where p is the number of exemplars used during the training process. $\widehat{x}_i$ and x_i are the predicted output and the actual output respectively during the back-propagation learning process and err is the mean squared error.

The genetic algorithm optimization process is described in the following procedure:

1. Randomize population.

2. Evaluate the fitness function for each individual in the population.

3. Select the first two individuals with the highest fitness values and copy directly to the next generation without any genetic operation.

4. Select the remaining individuals in the current generation and apply crossover and mutation genetic operations accordingly to reproduce the individuals in the next generation.

5. Repeat from the second step until all individuals in population meet the convergence criteria or the number of generations exceeds the pre-defined maximum values.

6. Decode the converged individuals in the final generation and obtain the optimized neural network structure with optimal number of input neurons, and optimal number of neurons in the hidden layer.

3.2 Data Sets Description and Pre-processing

3.2.1 Data Sets in Software Reliability Prediction Application

The performance of the proposed approaches have been tested using the same real-time control application and flight dynamic application data sets as cited in Park et al. [29] and Karunanithi et al. [27]. We choose a common baseline to compare the results with related work cited in the literature. All four data sets used in the experiments are summarized as follows:

DATA SET # 1: Real-time command and control application consisting of 21,700 assembly instructions and 136 failures.

DATA SET # 2: Flight dynamic application consisting of 10,000 lines of code and 118 failures.

DATA SET # 3: Flight dynamic application consisting of 22,500 lines of code and 180 failures.

DATA SET # 4: Flight dynamic application consisting of 38,500 lines of code and 213 failures.

DATA SET # 1 is obtained from Musa *et al.* [17]. DATA SET # 2, DATA SET # 3, and DATA SET # 4 are equivalent to DATA-11, DATA-12, and DATA-13 as cited in Park *et al.* [29] and Karunanithi *et al.* [27].

3.2.2 Data Sets in Short-Term Load Forecasting Application

The performance validation of our proposed approach is conducted using the same actual power load measurements recorded daily over a period of two years in Berkeley, California as cited in Karayiannis *et al* [42]. We choose a common baseline to compare our results with related work cited in the literature and use the same data range of

training (April 10 – June 9, 1999) and testing (Jan. 25 – March 26, 2001) in our proposed approach.

3.2.3 Data Pre-processing

All the inputs and outputs of the network are scaled and normalized within the range of [0.1, 0.9] to minimize the impact of absolute scale. For this purpose, the actual values are scaled using the following relationship [51]:

$$y = \frac{0.8}{\Delta}x + (0.9 - 0.8 \times \frac{x_{max}}{\Delta}) \tag{3.3}$$

where, y is the scaled value we feed into our network, x is the actual value before scaling, x_{max} is the maximum value in the samples. x_{min} is the minimum value among all the samples, and Δ is defined as $(x_{max} - x_{min})$. After the training process, we test the prediction performance by scaling back all the network outputs to their actual values using the following equation:

$$x = \frac{y - 0.9}{0.8} \times \Delta + x_{max} \tag{3.4}$$

3.3 Application in Software Reliability Prediction

3.3.1 Modeling Rationale

Unlike traditional neural network based software reliability growth modeling approaches, we model the inter-relationship among software failure time. Suppose x_i is the failure time of the ith failure. We want to model x_{i+k} by using $x_i, x_{i+1}, \ldots, x_{i+k-1}$, representing a functional relationship between the observed software failures and the future software failures.

$$x_{i+k} = f(x_i, x_{i+1}, \ldots, x_{i+k-1}) \quad i = 1, 2, \ldots \tag{3.5}$$

where k is the number of the input neurons in the network.

Genetic algorithm is used to globally optimize the neural network architecture after every occurrence of software failure time data. The optimization process determines the number of the delayed input neurons k corresponding to the previous failure time data sequence and the number of neurons in the hidden layer. The corresponding globally optimized number of delayed input neurons and the number of neurons in the hidden layer will be iteratively and dynamically reconfigured as new failure time data arrive in order to predict $\widehat{x}_{i+k}$.

3.3.2 Performance Metrics

Our choice for using specific performance measures for assessing the predictive accuracy was based on similar measures used by other researchers. We believe it is reasonable to compare our results with existing work using the same data sets and same performance evaluation metrics. This provides us the opportunity to quantitatively gauge the efficacy of our proposed approach. In addition, the relative error (RE) and/or average relative error (AE) are widely used in [26, 52, 53, 28, 27, 31, 54, 29, 37, 55] for assessment of predictive accuracy of cumulative patterns.

Let $\widehat{x}_i$ be the predicted value of failure time and x_i be the actual value of failure time. n is the number of data points in the test data set.

Relative Error (RE) is given by:

$$RE = \left| \frac{\widehat{x}_i - x_i}{x_i} \right| \tag{3.6}$$

Average Relative Prediction Error (AE) is given by:

$$AE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\widehat{x}_i - x_i}{x_i}\right| \times 100 \tag{3.7}$$

Predictability represented by Relative Error (RE) is defined by the percentage of the predicted values fall within a pre-determined range of RE compared to their actual observed values.

The larger the value of Predictability, or the smaller the value of AE, the closer are the predicted values to the actual values.

3.3.3 Test Results

To establish a baseline for the proposed dynamic evolutionary connectionist approach, we first experimented with an evolutionary neural network approach. For each data set, the first 50% of data are used for training purposes. At the same time, the neural network architecture is optimized by using genetic algorithm based on training performance. The remaining 50% of data will be used for testing the real predictive power of the model.

The performance results of predictability represented by Relative Error (RE) in both training and test process using the four data sets are shown in Table 3.1.

Table 3.1: Performance Results

RE $\leq$ 5%	Training Process	Test Process
DATA-1	67.24%	89.23%
DATA-2	80.00%	94.44%
DATA-3	89.61%	100.00%
DATA-4	94.62%	97.17%

For example, using DATA-2, 80% of the predicted values fall within 5% of their actual observed values in the training data set, while 94.44% of the predicted values

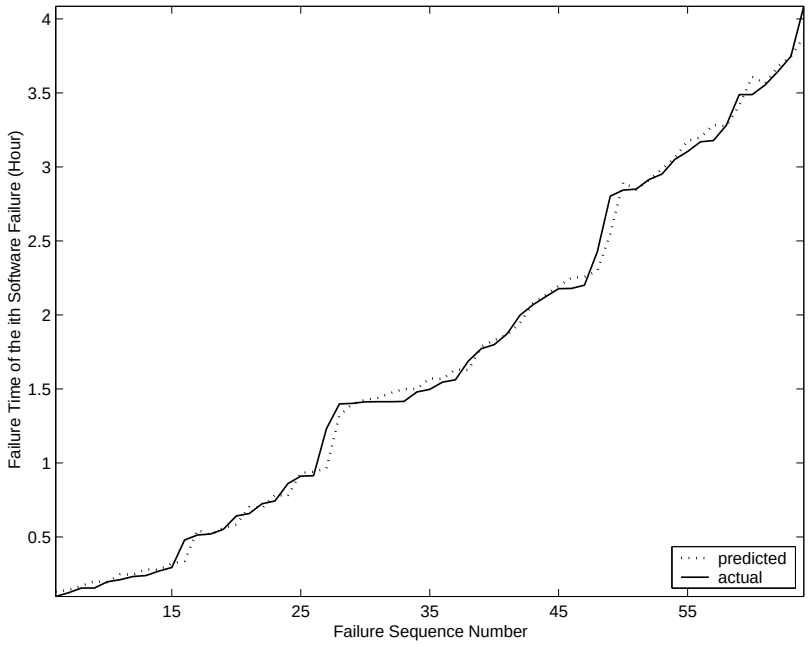

Figure 3.1: Performance using DATA-1 with training data set.

fall within 5% of their actual observed values in the test data set.

Fig. 3.1, Fig. 3.3, Fig. 3.5 and Fig. 3.7 show the predicted and actual values of the failure time in all four data sets during training process. Fig. 3.2, Fig. 3.4, Fig. 3.6 and Fig. 3.8 show the predicted and actual values of the failure time in all four data sets during test process.

Then, we experimented with the evolutionary neural network approach in a dynamic environment. To determine the next-step-predictability, we iteratively present the failure time data one at a time to the dynamically learned and optimized network. $x_i, x_{i+1}, \ldots, x_{i+k-1}$ are used to predict the value of x_{i+k}, where k is the number of delayed input neurons in the network we identified through genetic algorithm. Then the predicted and the actual values of failure time are compared. The results of the predictability represented by relative error (RE) using the four data sets are shown in Table 3.2.

For example, using DATA-3, 96% of the next-step predicted values fall within

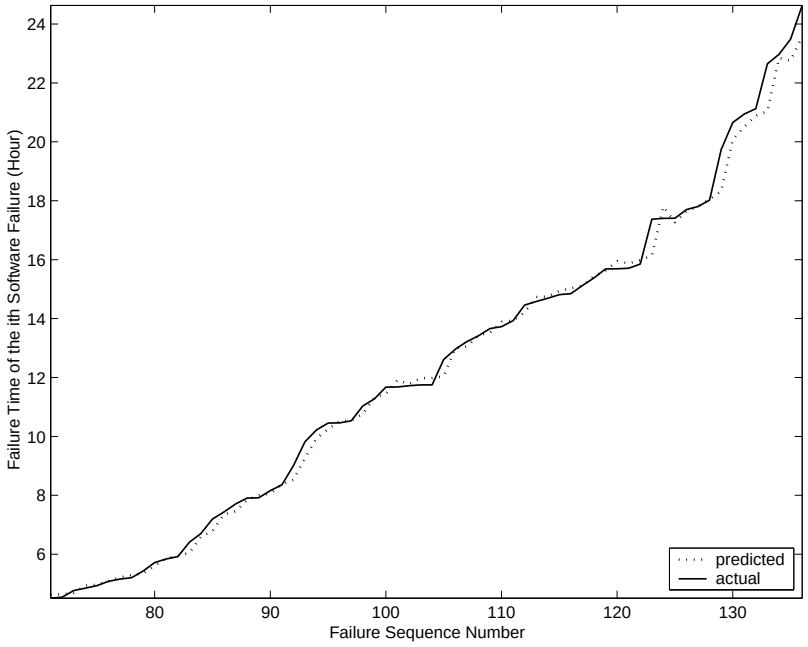

Figure 3.2: Performance using DATA-1 with test data set.

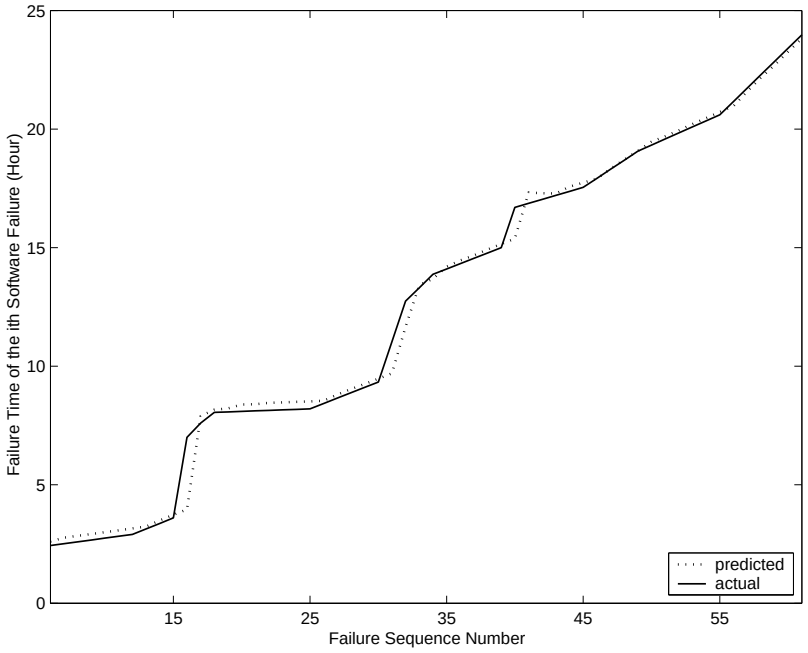

Figure 3.3: Performance using DATA-2 with training data set.

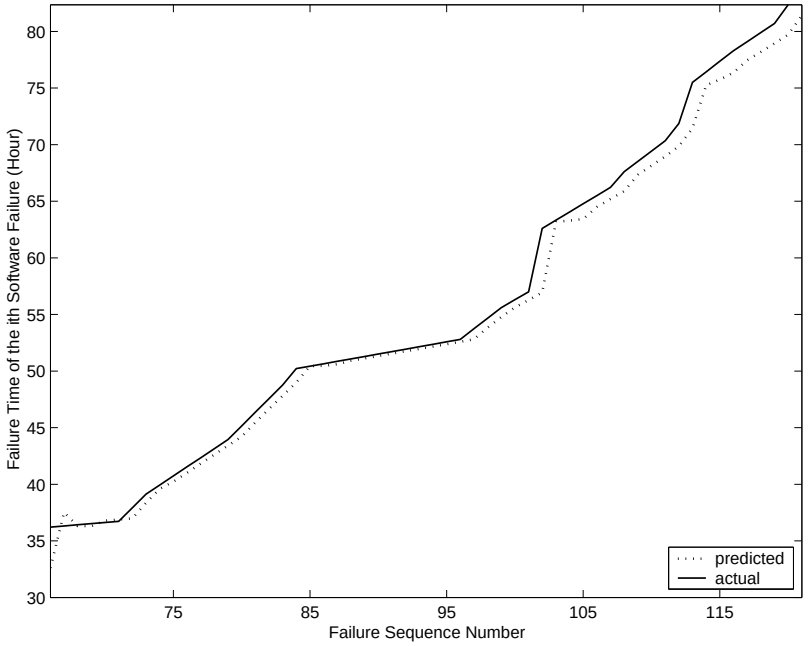

Figure 3.4: Performance using DATA-2 with test data set.

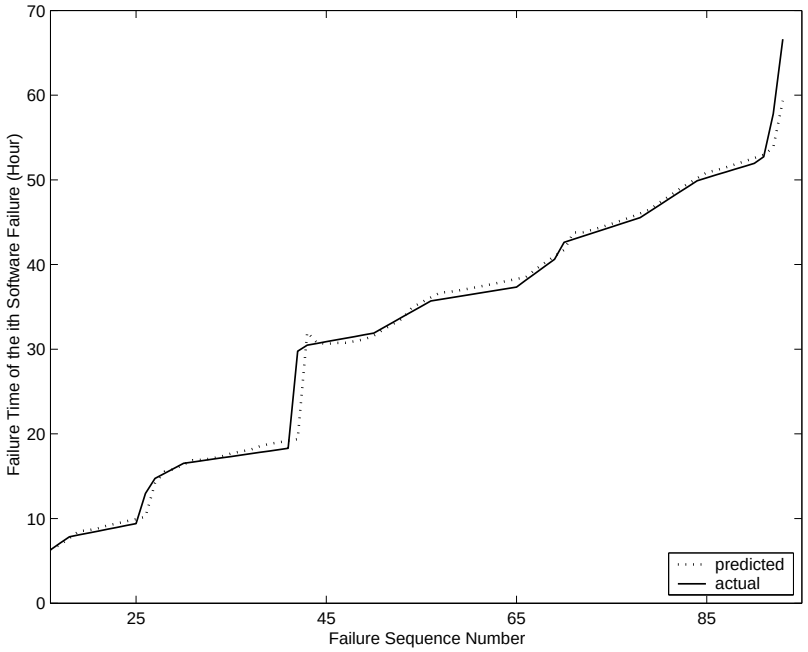

Figure 3.5: Performance using DATA-3 with training data set.

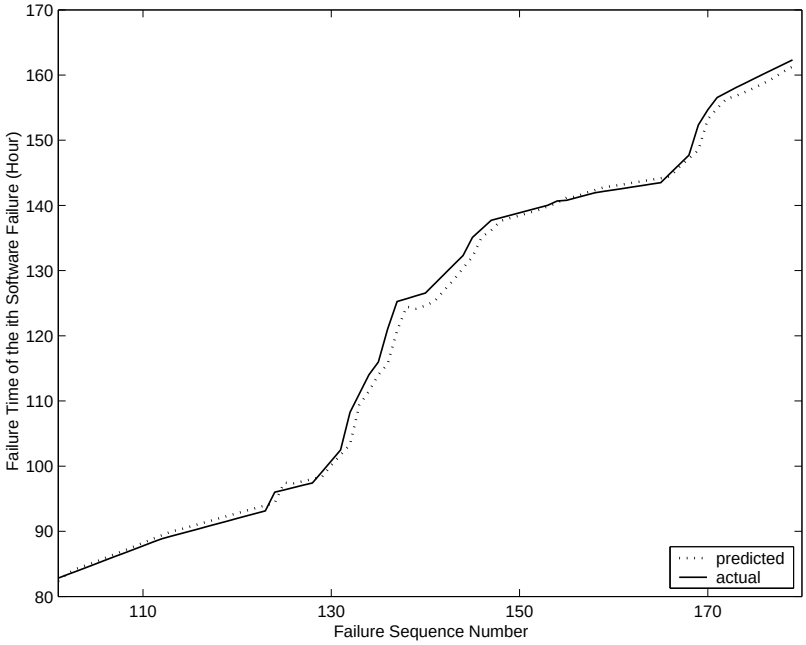

Figure 3.6: Performance using DATA-3 with test data set.

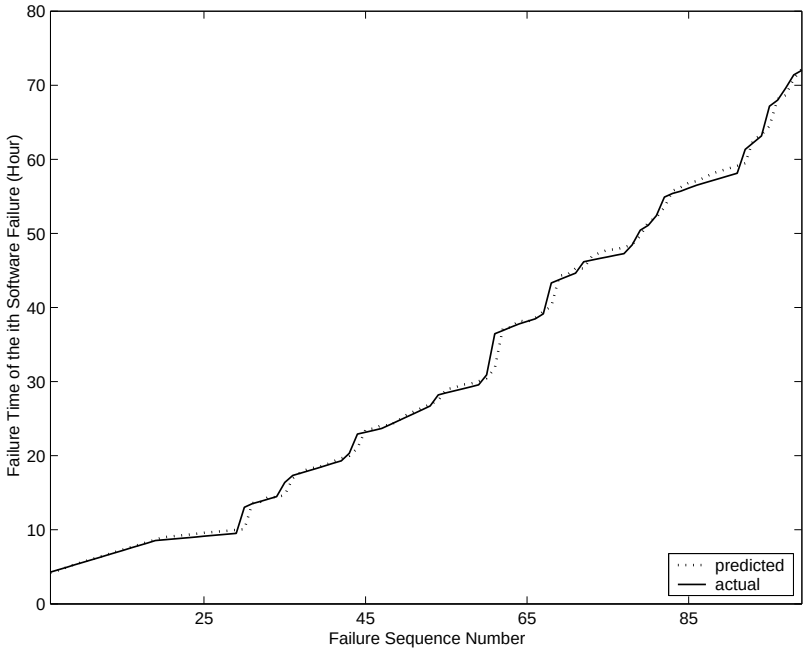

Figure 3.7: Performance using DATA-4 with training data set.

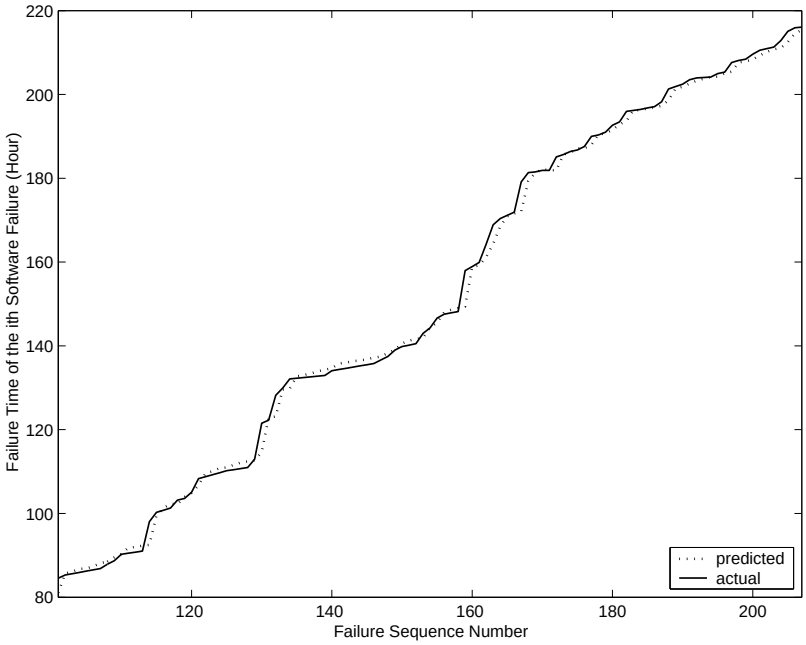

Figure 3.8: Performance using DATA-4 with test data set.

Table 3.2: Performance Results

Predictability ($RE \leq 5\%$)			
DATA-1	DATA-2	DATA-3	DATA-4
83%	89%	96%	95%

5% of their actual observed values. The results show that our proposed evolutionary neural networks approach provides highly accurate on-line prediction capability.

Table 3.3 summarizes the average relative prediction error using our proposed approach. The values of the prediction errors obtained are low. Fig. 3.9, Fig. 3.10, Fig. 3.11 and Fig. 3.12 show the prediction performance profile in detail for all four real-time control and flight dynamic application data sets. The proposed evolutionary connectionist model dynamically learns and optimizes the neural network architecture whenever a new failure time data arrives, and is easily implemented to predict failures

in real-time.

Table 3.3: Comparison of Average Relative Prediction Error

Data Sets	Comparison of Test Data Sets ($AE\%$)			
	Proposed D - ENN	FFNN (Ref. [29])	RNN (Ref. [27])	FFNN (Ref. [27])
DATA-1	2.72	2.58	2.05	2.50
DATA-2	2.65	3.32	2.97	5.23
DATA-3	1.16	2.38	3.64	6.26
DATA-4	1.19	1.51	2.28	4.76

3.4 Application in Short-Term Load Forecasting

3.4.1 Modeling Rationale

Traditionally, power system load is characterized by a combination of four components, which is given by:

$$Load = L_{normal} + L_{weather} + L_{special} + L_{random} \tag{3.8}$$

where $Load$ is total system load, L_{normal} is a standardized load shape according to different type of day throughout the year. $L_{weather}$ is the weather-related part of the load, such as humidity and temperature, and $L_{special}$ represents some unusual events that contribute to the major deviation of typical load behavior. L_{random} represents minor random factors that have impact on the load behavior [41].

Unlike the traditional mapping characteristics, we model the inter-relationship among power load data independent of other factors.

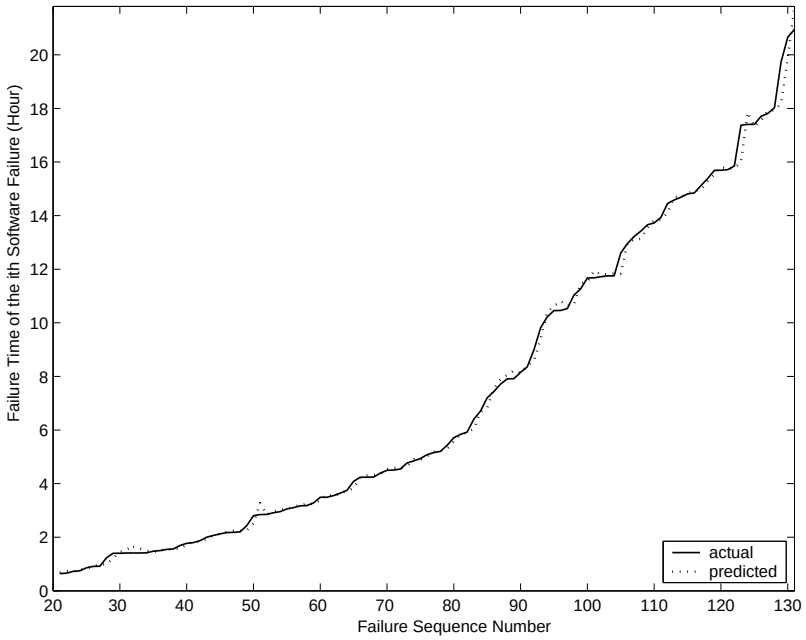

Figure 3.9: Prediction performance using DATA-1.

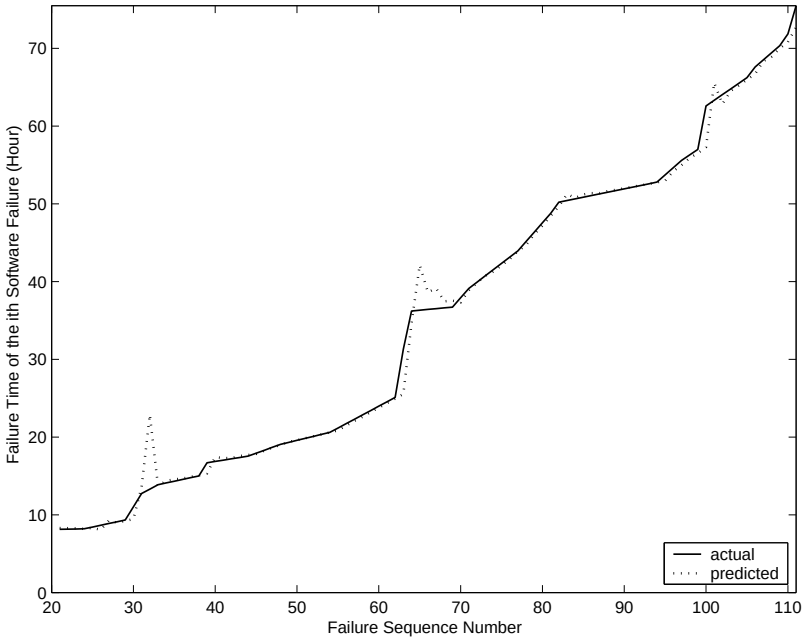

Figure 3.10: Prediction performance using DATA-2.

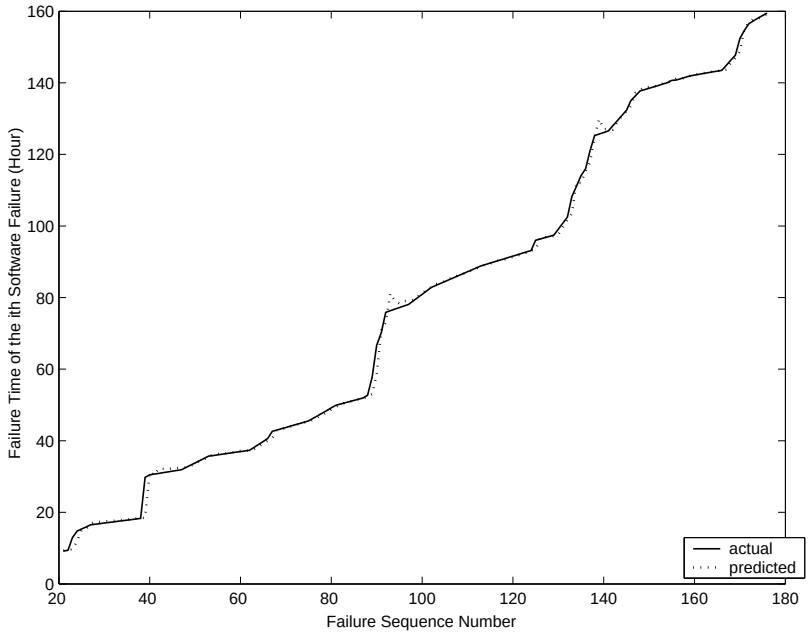

Figure 3.11: Prediction performance using DATA-3.

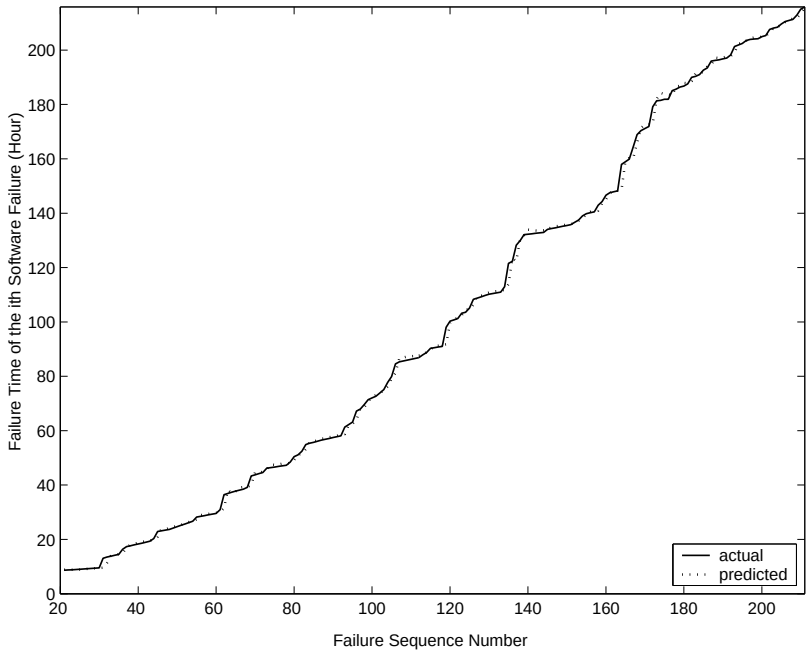

Figure 3.12: Prediction performance using DATA-4.

We assume that there exists nonlinear relationship between $x(d_{i+k})$ and $x(d_i)$, $x(d_{i+1})$, ..., $x(d_{i+k-1})$, where $x(d_i)$ is the corresponding load data in day d_i. Then, we want to forecast $x(d_{i+k})$ using

$$x(d_{i+k}) = f(x(d_i), x(d_{i+1}), \ldots, x(d_{i+k-1})) \quad i = 1, 2, \ldots \tag{3.9}$$

where k is the number of delayed load input neurons in the network.

Our proposed framework of evolutionary neural network modeling for short-term load forecasting is described by the following procedure.

1. Collect the historical daily load data.

2. Optimize the neural network architecture by finding the optimal number of input neurons and the optimal number of neurons in the hidden layer using the genetic algorithm procedure described in Section 3.1.

3. Input the unknown data points to our well-trained and generalized neural network and validate the predictive performance.

3.4.2 Performance Metrics

The following statistical metrics are used for comparing prediction performance, namely, Mean Square Error (MSE) and Root Mean Square Error ($RMSE$).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{x}(d_i) - x(d_i))^2 \tag{3.10}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\widehat{x}(d_i) - x(d_i))^2} \tag{3.11}$$

where $\widehat{x}(d_i)$ is the predicted value of daily average load, $x(d_i)$ is the actual value of daily average load, and n is the number of days during training and testing. The smaller the values of MSE and RMSE, the closer are the predicted values to the actual values.

3.4.3 Test Results

The performance validation of our proposed approach was conducted using the actual power load measurements recorded daily over a period of two years in Berkeley, California as cited in [42].

The performance results of predictability represented by Relative Error (RE) in both training and test process are shown in Table 3.4. The larger the value of predictability, the closer are the predicted values to the actual values.

Table 3.4: Performance Results

RE $\leq 5\%$	Training Process	Test Process
Daily Average Load Data	94.23%	98.11%

Table 3.5 summarizes the results of daily average load forecasting using our proposed approach based on the commonly used statistical metrics Mean Square Error (MSE) and Root Mean Square Error ($RMSE$). Karayiannis *et al.* [42] applied both feed-forward neural network (FFNN) and cosine radial basis function neural network (RBFNN) approaches for daily average load forecasting based on input variables of past load, temperature and humidity. These results are also summarized in Table 3.5. For example, using our proposed approach with the same testing data set, the Root Mean Square Error (RMSE) is 0.0187 by using load as the only input variable. The error is lower than the results obtained by RBFNN approach (0.1120) and FFNN approach (0.1702) in Karayiannis *et al.* [42] that use multiple input variables. The results show that our proposed approach yields better generalization capability and

lower prediction error compared to other neural network approaches.

Table 3.5: Performance Comparisons

	Performance Metrics	Proposed D - ENN	RBFNN [42]	FFNN [42]
Training Data	MSE	5.1942×10^5	3.2296×10^7	3.0120×10^7
04/10 – 06/09, 1999	$RMSE^*$	0.0274	0.2160	0.2086
Testing Data	MSE	2.4129×10^5	8.6792×10^6	2.0034×10^7
01/25 – 03/26, 2001	$RMSE^*$	0.0187	0.1120	0.1702
Input Variables Used		Load only	Load, Temperature, Humidity	

* scaled by the mean value of load data.

3.5 Summary

In this chapter, we proposed an evolutionary optimization approach for neural network architecture. In certain applications, the number of available data increases over time. The optimization process determines the number of the input neurons and the number of neurons in the hidden layer. The corresponding globally optimized neural network structure will be iteratively and dynamically reconfigured and updated as new data arrive to improve the prediction accuracy.

The proposed approach has been successfully applied and validated on applications related to software reliability prediction and electric power load forecasting. The data sets used for software reliability prediction are four real-time control application and flight dynamic application data sets. The data sets used for short-term load forecasting are the actual power load measurements recorded daily over a period of two years in Berkeley, California. We choose a common baseline to compare the results

with related work cited in the literature. Quantitative results show that the proposed approach achieves better prediction accuracy compared to existing approaches. For software reliability prediction, we obtain statistically higher prediction accuracy compared to the existing neural network models. For short-term load forecasting, the proposed approaches yield lower prediction error using minimal number of input variables compared to the existing approaches that use multiple input variables.

The research contributions in this chapter are also summarized in the following articles [56, 57, 58]:

- L. Tian and A. Noore, "On-line prediction of software reliability using an evolutionary connectionist model," *Journal of Systems and Software,* vol. 77, no. 2, pp. 173–180, Aug. 2005.

- L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," *Reliability Engineering and System Safety,* vol. 87, no. 1, pp. 45–51, Jan. 2005.

- L. Tian and A. Noore, "Short-term load forecasting using optimized neural network with genetic algorithm," in *Proceedings of the 8th International Conference on Probabilistic Methods Applied to Power Systems,* (Ames, IA), pp. 135–140, Sep. 2004.

Chapter 4

Improving Generalization Capability Using Recurrent Neural Network and Bayesian Regularization

4.1 Recurrent Neural Network

One of the major problems for multiple-input single-output purely static feed-forward neural network modeling is that we have to determine the exact number of inputs in advance. Earlier studies have selected this in an ad hoc manner and may not yield a globally optimized solution. This is the reason for using genetic algorithm to optimize the network structure. More importantly, for those applications where time information is involved, feed-forward neural network does not have the capability of incorporating dynamic temporal property internally, which may have impact on the network prediction performance. If time can be represented by the effect it has on processing, the network will perform better in terms of responsiveness to temporal se-

quences. This responsiveness can be obtained by providing feedback of data generated by the network back into the units of the network to be used in future iterations.

Recurrent neural network has the inherent capability of developing an internal memory, which may naturally extend beyond the externally provided lag spaces, and hence relaxing the requirements for the determination of external number of inputs in time-related prediction applications [59].

Recurrent neural networks are feedback networks in which the current activation state is a function of previous activation state and the current inputs. This feedback path allows recurrent networks to learn to recognize and generate time-varying patterns [60].

A simple illustration of recurrent network is shown in Fig. 4.1.

For simplicity and comparison purposes, we first consider the most elementary feed-forward network shown in Fig. 4.1(a), where the input, hidden, and output layers each has only one neuron. When the input $x(t_0)$ at time t_0 is applied to the input layer, the output $v(t_0)$ of the hidden layer and the output $y(t_0)$ of the output layer are given by:

$$v(t_0) = \Phi(w_{12} \times x(t_0)) \tag{4.1}$$

$$\begin{aligned} y(t_0) &= \Phi(w_{23} \times v(t_0)) \\ &= \Phi(w_{23} \times \Phi(w_{12} \times x(t_0))) \end{aligned} \tag{4.2}$$

where $\Phi(\cdot)$ is the activation function. As shown in Fig. 4.1(b), recurrent neural network has feedback from the output layer to the hidden layer and feedback from the hidden layer to the input layer through the recurrent neurons labeled R. The corresponding feedback weights are w_{32} and w_{21}, respectively. When the input $x(t_1)$ is applied to the input layer, the output $v(t_1)$ of the hidden layer and the output $y(t_1)$

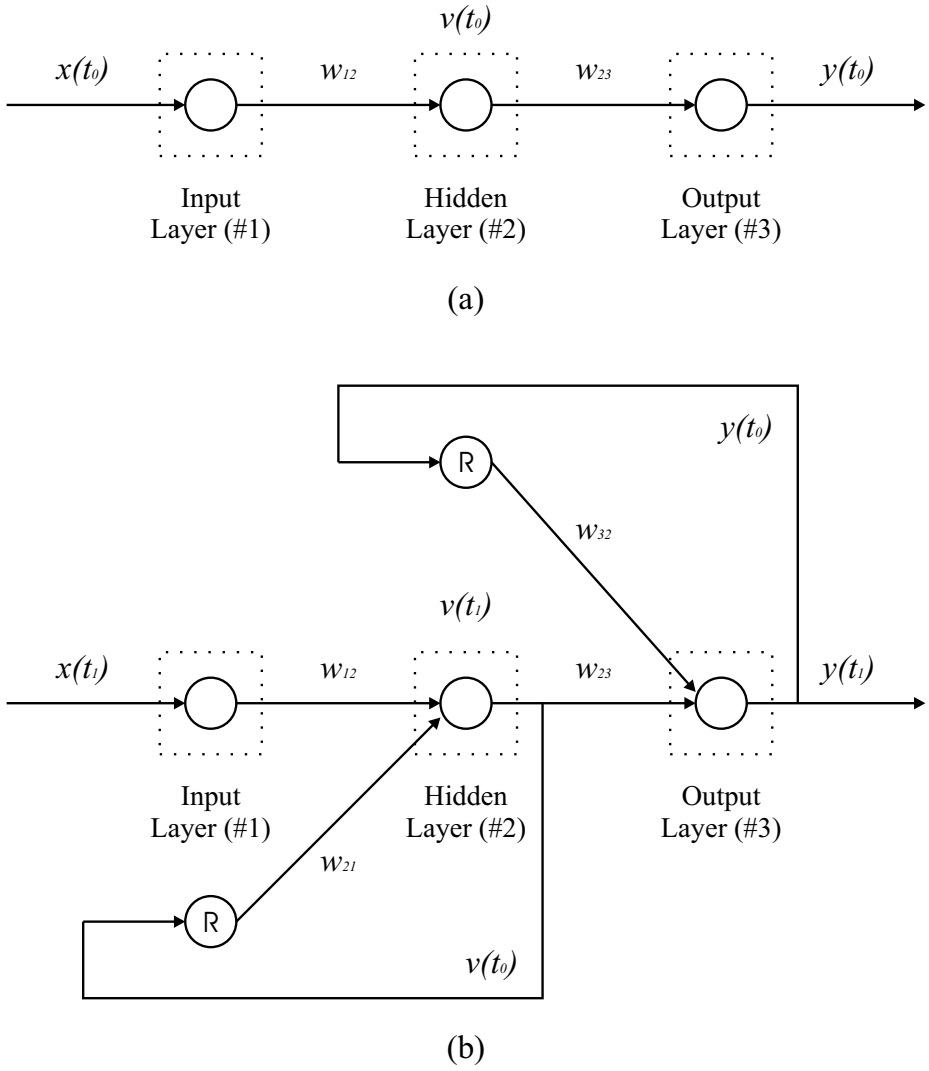

Figure 4.1: Static feed-forward neural network in (a) and recurrent neural network in (b) with feedback connections.

of the output layer are given by:

$$\begin{aligned} v(t_1) &= \Phi(w_{12} \times x(t_1) + w_{21} \times v(t_0)) \\ &= \Phi(w_{12} \times x(t_1) + w_{21} \times \Phi(w_{12} \times x(t_0))) \qquad (4.3) \\ y(t_1) &= \Phi(w_{23} \times v(t_1) + w_{32} \times y(t_0)) \\ &= \Phi(w_{23} \times \Phi(w_{12} \times x(t_1) + w_{21} \times \Phi(w_{12} \times x(t_0))) \\ &+ w_{32} \times \Phi(w_{23} \times \Phi(w_{12} \times x(t_0)))) \qquad (4.4) \end{aligned}$$

Without loss of generality, we assume that the input layer, the hidden layer, and the output layer each has multiple neurons, and there could be more than one hidden layer. Each processing element of a recurrent neural network is denoted by the following generalized equations [61]:

$$p_{[l,n]}(t_i) = \sum_{m=1}^{N_{[l]}} w_{[l,m][l,n]} q_{[l,m]}(t_{i-1}) + \sum_{m=1}^{N_{[l-1]}} w_{[l-1,m][l,n]} q_{[l-1,m]}(t_i) + b_{[l,n]} \qquad (4.5)$$

$$q_{[l,n]}(t_i) = \Phi_{[l,n]}(p_{[l,n]}(t_i)) \qquad (4.6)$$

where,

$p_{[l,n]}(t_i)$	is the internal state variable of the n^{th} neuron in the l^{th} layer at time t_i
$q_{[l,n]}(t_i)$	is the output of the n^{th} neuron in the l^{th} layer at time t_i
$b_{[l,n]}$	is the bias of the n^{th} neuron in the l^{th} layer
$w_{[l,m][l',n]}$	is the weight associated with the link between the m^{th} neuron of the l^{th} layer to the n^{th} neuron of the l'^{th} layer
$\Phi(\cdot)$	is the activation function

4.2 Bayesian Regularization

A desirable neural network model should have small errors not only in the training data set, but also in the validation or testing data set [62]. The ability to adapt to previously known data as well as unknown data requires improving generalization. Regularization constrains the size of the network parameters. When the parameters in a network are kept small, the response of the network will be smooth [62]. With regularization, the performance function is modified by adding a term that consists of the mean of the sum of squares of the neural network weights and biases. The mean squared error with regularization performance, mse_{reg}, is given by:

$$mse_{reg} = \beta \times mse + (1 - \beta) \times msw \tag{4.7}$$

where, β is the performance ratio and represents the relative importance of errors vs. weight and bias values, mse is the mean squared error during training, and msw is the mean squared weights and biases.

By using this modified performance function, mse_{reg}, the neural network is forced to have smaller weights and biases, which causes the network to respond smoother, represent the true function rather than capture the noise, and is less likely to overfit.

The major problem with regularization is to choose the performance ratio coefficient β. MacKay [63] has done extensive work on the application of Bayes rule for optimizing regularization. Hessian matrix computation is required for regularization optimization. In order to minimize the computational overhead, Foresee and Hagan [64] proposed using a Gauss-Newton approximation to the Hessian matrix, which is readily available while Levenberg-Marquardt algorithm is used as neural network training scheme.

The Bayesian optimization of the regularization coefficient with a Gauss-Newton approximation to the Hessian matrix is described in the following procedures [64]:

1. Initialize $\beta(\beta = 1)$ and the weights.
2. Minimize the performance function msereg by using Levenberg-Marquardt algorithm.
3. Compute the effective number of parameters using Gauss-Newton approximation to the Hessian matrix available in the Levenberg-Marquardt training algorithm.
4. Derive the new estimate of β.
5. Repeat from Step 2-4 until the convergence is obtained. Thus, the optimized value for β is chosen.

Kwok et al. [65], Ishikawa [66], Gencay et al. [62], and Chua et al. [67] all reported better generalization performance by using Bayesian regularization in other types of neural networks. In this research, Bayesian regularization with recurrent training scheme (RNN + BR) is used for improving the generalization capability.

4.3 Application in Software Reliability Prediction

4.3.1 Formulation of the Neuro-Predictor

In recurrent neural network structure, each processing element has the task of mapping both an external input and the previous internal state to some desired output. Thus, the internal representation developed are sensitive to temporal context [60]. More specifically, in our failure time modeling, the input-output pattern fed into the network is the failure temporal sequence. Thus, the recurrent network can learn and recognize the inherent temporal patterns of input-output pair. For one-step-ahead prediction, the input sequence and the desired output sequence should have one step delay during the training process. The desired objective is to force the network to recognize the one-step-ahead temporal pattern. A sample input sequence and the corresponding one-step-ahead desired output sequence is defined as:

$$InputSequence: \quad x(t_0), x(t_1), \cdots, x(t_{i-1}), x(t_i), x(t_{i+1}), \cdots$$

$$OutputSequence: \quad x(t_1), x(t_2), \cdots, x(t_i), x(t_{i+1}), x(t_{t+2}), \cdots$$

where $x(t_i)$ is the failure time in the training data sequence, and t_i is the failure time sequence index. The activation function in our modeling approach is linear for the output layer, and it is hyperbolic tangent sigmoidal for hidden layer neurons. Once the network is trained based on sufficient training data sequence, the unknown data sequence will be presented to the network to validate the performance.

4.3.2 Performance Metrics

The performance metrics used are the same as described in Section 3.3.2, Relative Error (RE) and Average Relative Prediction Error (AE).

4.3.3 Test Results

The performance results of predictability represented by Relative Error (RE) in both training and test process using the four data sets are shown in Table 4.1.

Table 4.1: Performance Results

RE $\leq$ 5%	Training Process	Test Process
DATA-1	70.77%	88.81%
DATA-2	78.33%	91.23%
DATA-3	87.23%	98.87%
DATA-4	93.00%	97.30%

For example, in data set DATA-2, 78.33% of the predicted values fall within 5% of their actual observed value in the training data set, while 91.23% of the predicted values fall within 5% of their actual observed value in the test data set. Similarly, the results for other data sets show that our proposed approach provides highly accurate generalization capability.

We next compute the average relative prediction error (AE) on all four data sets. Table 4.2 summarizes the results of modeling the temporal inter-relationship among software failure time sequence using our proposed recurrent neural network with Bayesian regularization (RNN + BR). Park *et al.* [29] applied failure sequence number as input and failure time as desired output in feed-forward neural network (FFNN). Based on the learning pair of execution time and the corresponding accumulated number of defects disclosed, Karunanithi *et al.* [27] employed both feed-forward neural network (FFNN) and recurrent neural network (RNN) structures to model the failure process. These results are also summarized in Table 4.2. In all four data sets, the results show that using recurrent neural network with Bayesian regularization yields a lower average relative prediction error compared to other neural network approaches.

Fig. 4.2, Fig. 4.4, Fig. 4.6 and Fig. 4.8 show the predicted and actual values of the failure time in all four data sets during training process. Fig. 4.3, Fig. 4.5, Fig. 4.7 and Fig. 4.9 show the predicted and actual values of the failure time in all four data sets during test process. In all cases, the recurrent neural network with Bayesian regularization produces promising results during both training and testing.

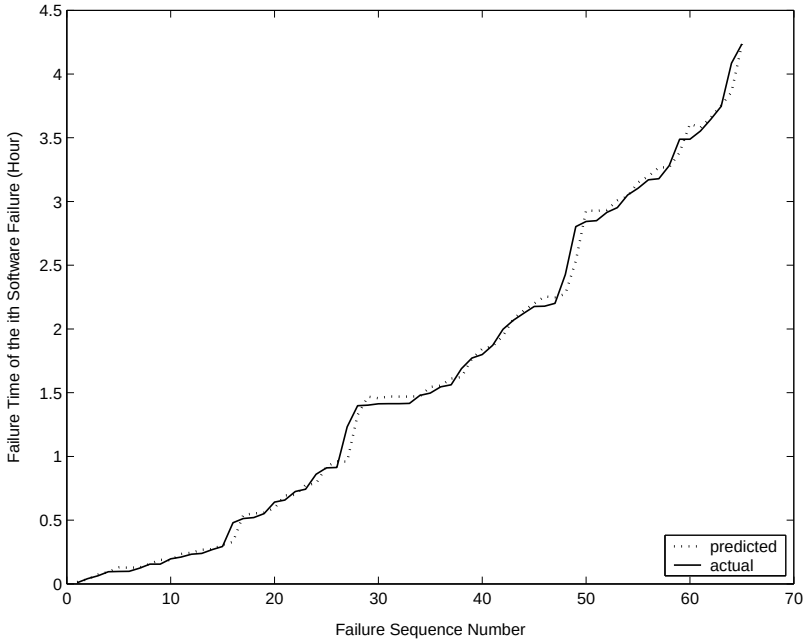

Figure 4.2: Performance using DATA-1 with training data set.

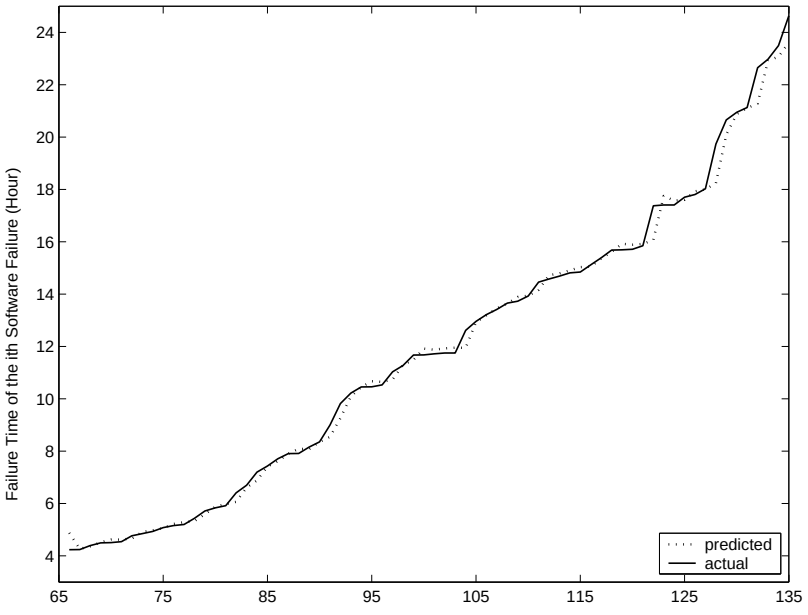

Figure 4.3: Performance using DATA-1 with test data set.

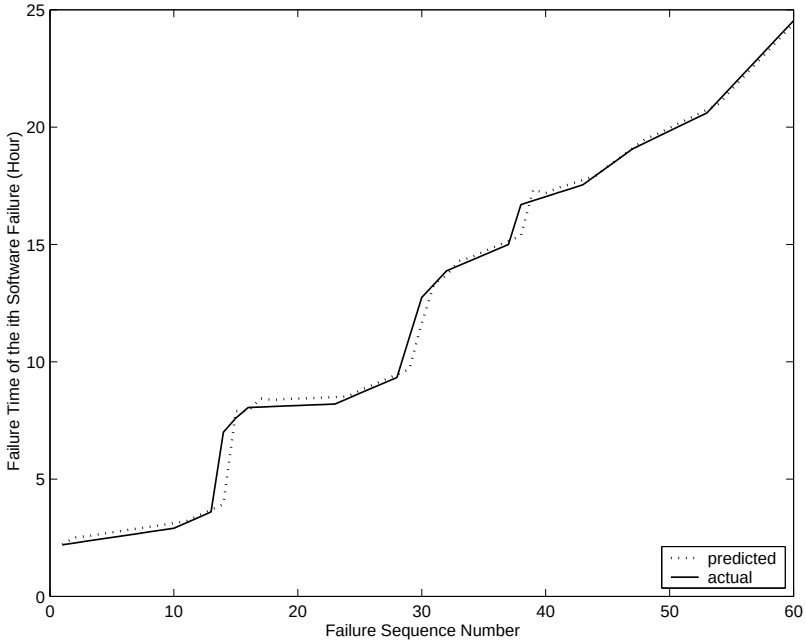

Figure 4.4: Performance using DATA-2 with training data set.

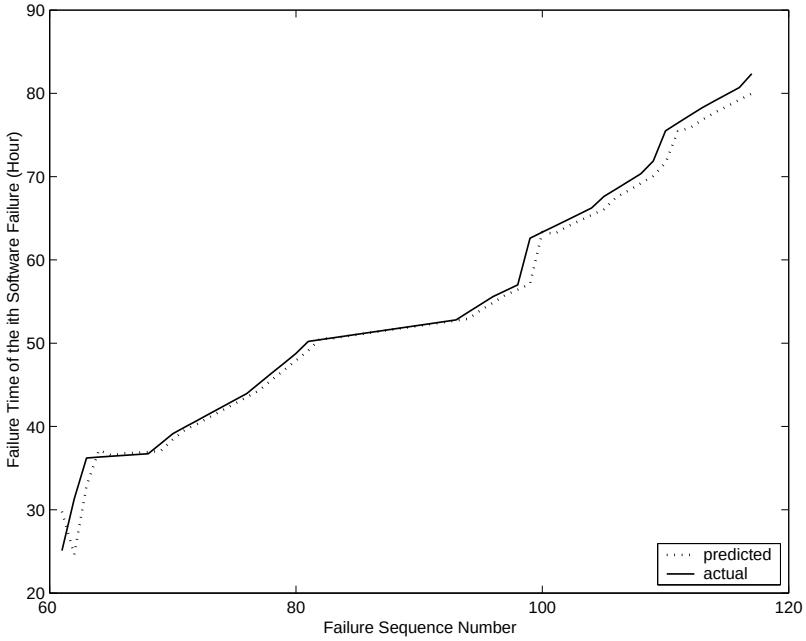

Figure 4.5: Performance using DATA-2 with test data set.

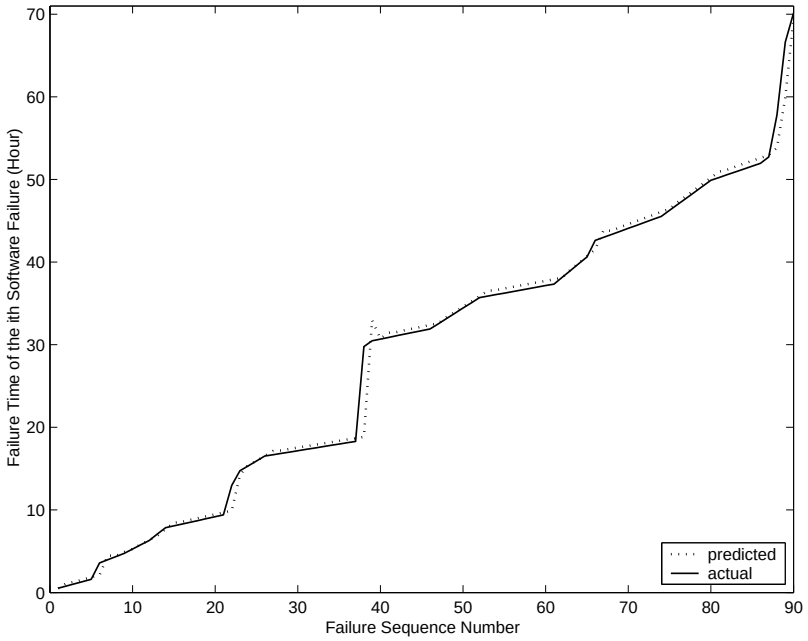

Figure 4.6: Performance using DATA-3 with training data set.

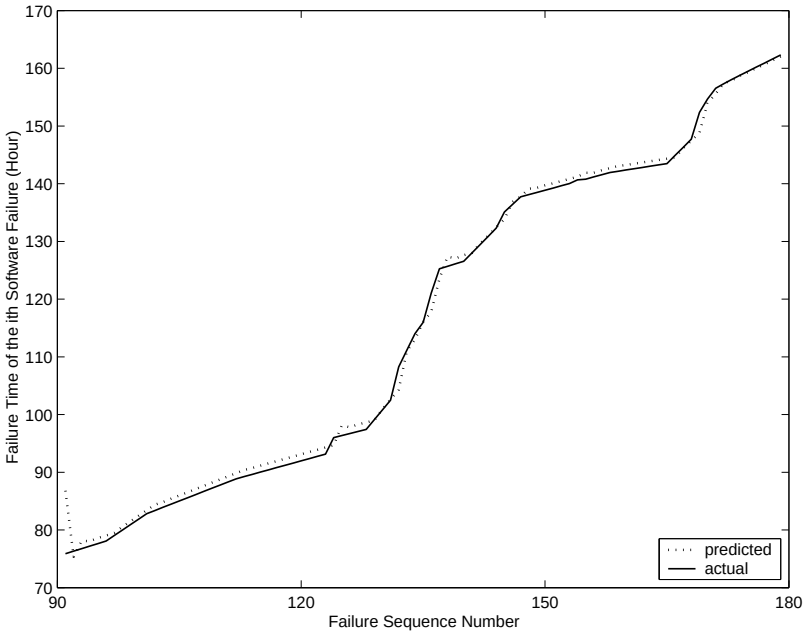

Figure 4.7: Performance using DATA-3 with test data set.

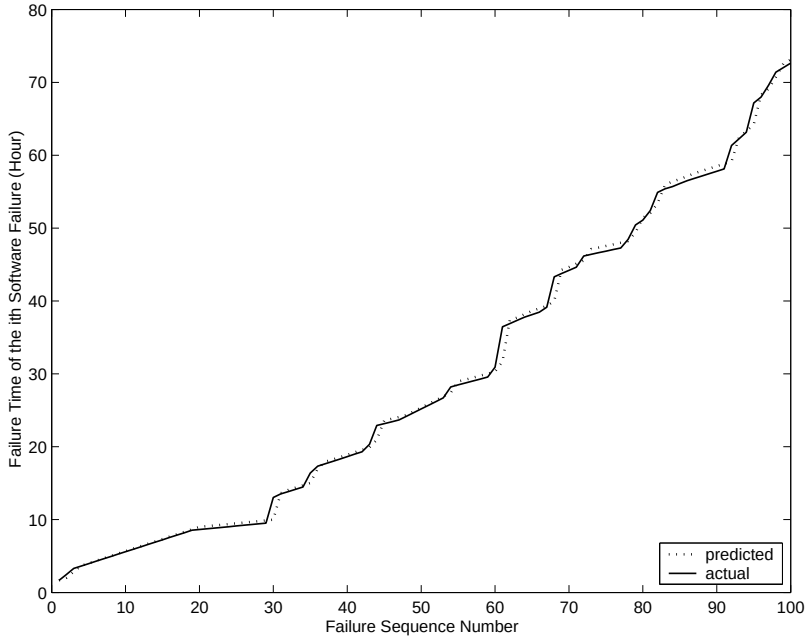

Figure 4.8: Performance using DATA-4 with training data set.

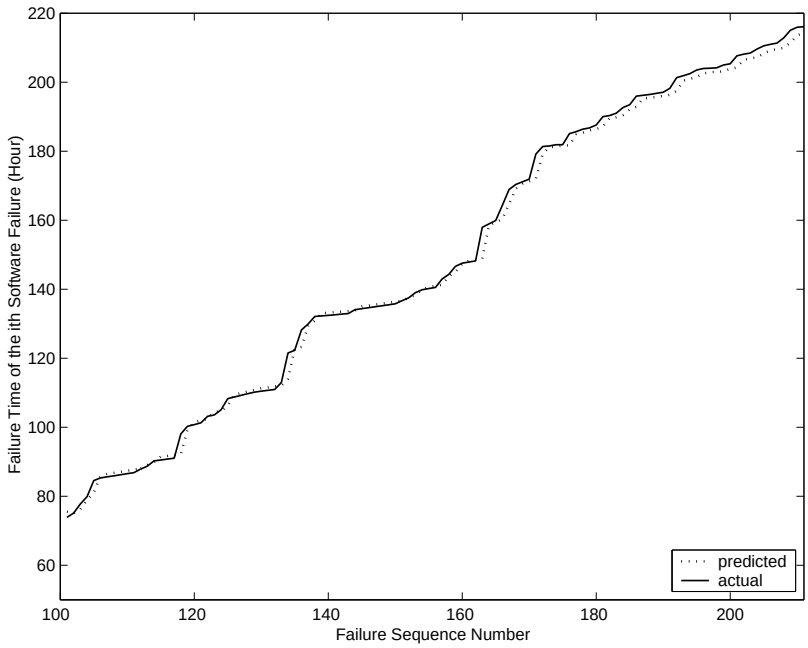

Figure 4.9: Performance using DATA-4 with test data set.

Table 4.2: Average Relative Prediction Error (%)

Data Sets	Comparison of Test Data Sets			
	Proposed RNN + BR	FFNN (Ref. [29])	RNN (Ref. [27])	FFNN (Ref. [27])
DATA-1	1.83	2.58	2.05	2.50
DATA-2	2.06	3.32	2.97	5.23
DATA-3	0.97	2.38	3.64	6.26
DATA-4	0.98	1.51	2.28	4.76

4.4 Application in Short-Term Load Forecasting

4.4.1 Modeling Rationale

Unlike the traditional mapping characteristics, we model the inter-relationship among power load data sequence. Using recurrent neural network and Bayesian regularization, the input sequence and the corresponding one-step-ahead desired output sequence are defined as:

$$InputSequence: \qquad x(d_0), x(d_1), \cdots, x(d_{i-1}), x(d_i), x(d_{i+1}), \cdots$$

$$OutputSequence: \qquad x(d_1), x(d_2), \cdots, x(d_i), x(d_{i+1}), x(d_{t+2}), \cdots$$

where $x(d_i)$ is the corresponding load data in day d_i. Once the network is trained based on the available training data sequence, the unknown data sequence will be presented to the network to validate the performance.

4.4.2 Performance Metrics

The performance metrics used are the same as described in Section 3.4.2, Mean Square Error (MSE) and Root Mean Square Error ($RMSE$).

4.4.3 Test Results

The performance results of predictability represented by Relative Error (RE) in both training and test process are shown in Table 4.3. The larger the value of predictability, the closer are the predicted values to the actual values.

Table 4.3: Performance Results

RE $\leq 5\%$	Training Process	Test Process
Daily Average Load Data	90.16%	90.16%

Table 4.4 summarizes the results of daily average load forecasting using our proposed approach based on the commonly used statistical metrics Mean Square Error (MSE) and Root Mean Square Error ($RMSE$). Karayiannis *et al.* [42] applied both feed-forward neural network (FFNN) and cosine radial basis function neural network (RBFNN) approaches for daily average load forecasting based on input variables of past load, temperature and humidity. These results are also summarized in Table 4.4. For example, using our proposed approach with the same testing data set, the Root Mean Square Error (RMSE) is 0.0286 by using load as the only input variable. The error is lower than the results obtained by RBFNN approach (0.1120) and FFNN approach (0.1702) in Karayiannis *et al.* [42] that use multiple input variables. The results show that our proposed approach yields better generalization capability and lower prediction error compared to other neural network approaches.

Table 4.4: Performance Comparisons

	Performance Metrics	Proposed RNN + BR	RBFNN [42]	FFNN [42]
Training Data	MSE	5.4832×10^5	3.2296×10^7	3.0120×10^7
04/10 – 06/09, 1999	$RMSE^*$	0.0282	0.2160	0.2086
Testing Data	MSE	5.6354×10^5	8.6792×10^6	2.0034×10^7
01/25 – 03/26, 2001	$RMSE^*$	0.0286	0.1120	0.1702
Input Variables Used		Load only	Load, Temperature, Humidity	

* scaled by the mean value of load data.

4.5 Summary

In this chapter, we proposed an modeling approach by using recurrent neural network and Bayesian regularization for Improving generalization capability. Recurrent neural network has the inherent capability of developing an internal memory, which may naturally extend beyond the externally provided lag spaces. Moreover, by adding a penalty term of sum of connection weights, Bayesian regularization approach is applied to the network training scheme to improve the generalization performance and lower the susceptibility of overfitting.

The proposed approach has been successfully applied and validated on applications related to software reliability prediction and electric power load forecasting. The data sets used for software reliability prediction are four real-time control application and flight dynamic application data sets. The data sets used for short-term load forecasting are the actual power load measurements recorded daily over a period of two years in Berkeley, California. We choose a common baseline to compare the results with related work cited in the literature. Quantitative results show that the proposed approach achieves better prediction accuracy compared to existing approaches.

For software reliability prediction, we obtain statistically higher prediction accuracy compared to the existing neural network models. For short-term load forecasting, the proposed approaches yield lower prediction error using minimal number of input variables compared to the existing approaches that use multiple input variables.

The research contributions in this chapter are also summarized in the following article [68]:

- L. Tian and A. Noore, "Software reliability prediction using recurrent neural network with Bayesian regularization," *International Journal of Neural Systems,* vol. 14, no. 3, pp. 165–174, June 2004.

Chapter 5

Adaptive Modeling Using Support Vector Machines

5.1 SVM Learning in Function Approximation

As a novel type of machine learning algorithm, support vector machine (SVM) has gained increasing attention from its original application in pattern recognition to the extended application in function approximation and regression estimation [69, 70, 71, 72, 73, 74]. Based on the structural risk minimization (SRM) principle, the learning scheme of SVM is focused on minimizing an upper bound of the generalization error that includes the sum of the empirical training error and a regularized confidence interval, which will eventually result in better generalization performance. Moreover, unlike other gradient descent based learning scheme that requires nonlinear optimization with the danger of getting trapped into local minima, the regularized risk function of SVM can be minimized by solving a linearly constrained quadratic programming problem, which can always obtain a unique and global optimal solution. Thus, the possibility of being trapped at local minima can be effectively avoided [70, 75, 73, 76].

5.1.1 Estimation of Real-Valued Functions

Notation

x_i	n-dimensional input vector, $x_i \in \Re^n$
y_i	target output value, $y_i \in \Re$
ϕ	high-dimensional feature space mapping function
w	weights vector
b	bias term
R	regularized risk function
$\|\|w\|\|^2$	weights vector norm
C	regularization constant
ϵ	Vapnik's linear loss function with ϵ-insensitivity zone
ξ_i, ξ_i^*	slack variables
α_i, α_i^*	Lagrange multipliers
K	kernel function

The basic idea of SVM for function approximation is mapping the data x into a high-dimensional feature space by a nonlinear mapping and then performing a linear regression in this feature space [75]. Assume that a total of l pairs of training patterns are given during SVM learning process,

$$(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_l, y_l)$$

where the inputs are n-dimensional vectors $x_i \in \Re^n$, and the target outputs are continuous values $y_i \in \Re$. The SVM model used for function approximation is:

$$f(x) = w \cdot \phi(x) + b \tag{5.1}$$

where $\phi(x)$ is the high-dimensional feature space that is nonlinearly mapped from the input space x. Thus, a nonlinear regression in the low-dimensional input space is transferred to a linear regression in a high-dimensional feature space [75]. The coefficients w and b can be estimated by minimizing the following regularized risk function R [69, 77, 75, 76, 73, 74, 78, 70, 79]:

$$R = \frac{1}{2}||w||^2 + C\frac{1}{l}\sum_{i=1}^{l}|y_i - f(x_i)|_\epsilon \tag{5.2}$$

where

$$|y_i - f(x_i)|_\epsilon = \begin{cases} 0 & \text{if } |y_i - f(x_i)| \leq \epsilon, \\ |y_i - f(x_i)| - \epsilon & \text{otherwise.} \end{cases} \tag{5.3}$$

$||w||^2$ is the weights vector norm, which is used to constrain the model structure capacity in order to obtain better generalization performance. The second term is the Vapnik's linear loss function with ϵ-insensitivity zone as a measure for empirical error. The loss is zero if the difference between the predicted and observed value is less than or equal to ϵ. For all other cases, the loss is equal to the magnitude of the difference between the predicted value and the radius ϵ of ϵ-insensitivity zone. C is the regularization constant, representing the trade-off between the approximation error and the model structure. ϵ is equivalent to the approximation accuracy requirement for the training data points. Further, two positive slack variables ξ_i and ξ_i^* are introduced. We have

$$|y_i - f(x_i)| - \epsilon = \begin{cases} \xi_i & \text{for data "above" an } \epsilon \text{ tube,} \\ \\ \xi_i^* & \text{for data "below" an } \epsilon \text{ tube.} \end{cases} \tag{5.4}$$

Thus, minimizing the risk function R in Equation 5.2 is equivalent to minimizing the objective function R_{w,ξ,ξ^*}.

$$R_{w,\xi,\xi^*} = \frac{1}{2}||w||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{5.5}$$

subject to constraints

$$\begin{cases} y_i - w \cdot \phi(x_i) - b \leq \epsilon + \xi_i & i = 1, \ldots, l, \\ w \cdot \phi(x_i) + b - y_i \leq \epsilon + \xi_i^* & i = 1, \ldots, l, \\ \xi_i, \xi_i^* \geq 0 & i = 1, \ldots, l. \end{cases} \tag{5.6}$$

This constrained optimization problem is typically solved by transforming into the dual problem, and its solution is given by the following explicit form:

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(x_i, x) + b \tag{5.7}$$

5.1.2 Lagrange Multipliers

In Equation 5.7, α_i and α_i^* are the Lagrange multipliers with $\alpha_i \times \alpha_i^* = 0$ and $\alpha_i, \alpha_i^* \geq 0$ for any $i = 1, \ldots, l$. They can be obtained by maximizing the following form:

$$-\epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) \tag{5.8}$$

subject to constraints

$$\begin{cases} \sum_{i=1}^{l} \alpha_i^* = \sum_{i=1}^{l} \alpha_i \\ \\ 0 \leq \alpha_i, \alpha_i^* \leq C \qquad\qquad i = 1, \ldots, l. \end{cases} \tag{5.9}$$

After learning, only some of coefficients $(\alpha_i - \alpha_i^*)$ in Equation 5.7 differ from zero, and the corresponding training data points are referred to as support vectors. It is obvious that only the support vectors can fully decide the decision function in Equation 5.7.

5.1.3 Kernel Function

In Equation 5.7, $K(x_i, x)$ is defined as the kernel function, which is the inner product of two vectors in feature space $\phi(x_i)$ and $\phi(x)$. By introducing the kernel function, we can deal with the feature spaces of arbitrary dimensionality without computing the mapping relationship $\phi(x)$ explicitly [70]. Some commonly used kernel functions are polynomial kernel function and Gaussian kernel function.

5.2 Adaptive Modeling

In certain applications, the number of available data increases over time during a dynamic system. Accordingly, the SVM learning process is iteratively and dynamically updated after every occurrence of new data in order to capture the most current feature hidden inside the data sequence. After the SVM learning process is complete based on the currently available data, next-step information will be predicted.

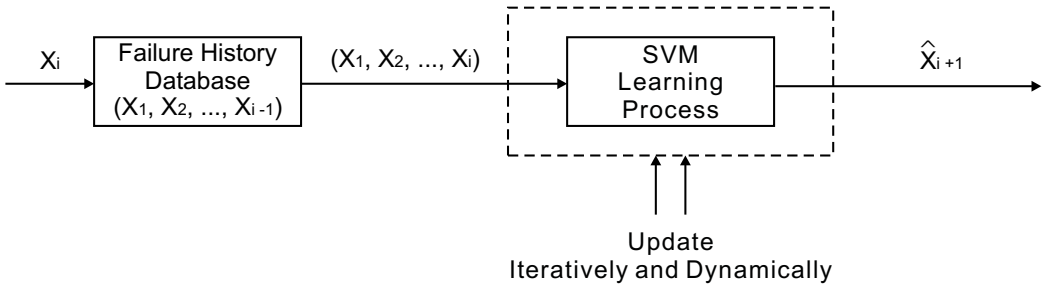

Figure 5.1: Dynamic software reliability prediction framework.

5.3 Application in Software Reliability Prediction

5.3.1 Formulation of the SVM-Predictor

The proposed software reliability prediction system shown in Fig. 5.1 consists of a failure history database and an iteratively and dynamically updated SVM learning-predicting process. When a software failure, x_i, occurs, the failure history database is updated and the accumulated failure data $(x_1, x_2, \ldots, x_i)$ is made available to the SVM learning process. The number of failure data increases over time during a dynamic system. Accordingly, the SVM learning process is iteratively and dynamically updated after every occurrence of new failure time data in order to capture the most current feature hidden inside the software failure sequence. After the SVM learning process is complete based on the currently available history failure data, next-step failure information, $\widehat{x}_{i+1}$, will be predicted.

In our proposed approach, unlike the existing mapping characteristics, we model the inter-relationship among software failure time data. More specifically, the input-output pattern fed into the network is the failure temporal sequence. The SVM learning scheme is applied to the failure time data, forcing the network to learn and recognize the inherent internal temporal property of software failure sequence. For one-step-ahead prediction, the input sequence and the desired output sequence should have one step delay during the learning process. The desired objective is to force the

network to recognize the one-step-ahead temporal pattern. A sample input sequence and the corresponding one-step-ahead desired output sequence is defined as:

$$Input\ Sequence: \quad x_0, x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots$$

$$Output\ Sequence: \quad x_1, x_2, \cdots, x_i, x_{i+1}, x_{i+2}, \cdots$$

where x_i is the failure time of the ith failure in the learning process. Once the network is trained based on all the currently available history failure data using the SVM learning procedure, the one-step-ahead failure time will be predicted. Accordingly, the SVM learning process is iteratively and dynamically updated after every occurrence of new failure time data in order to capture the most current feature hidden inside the software failure sequence.

5.3.2 Performance Metrics

The performance metrics used are the same as described in Section 3.3.2, Predictability represented by Relative Error (RE) and Average Relative Prediction Error (AE).

5.3.3 Test Results

The results of the predictability represented by Relative Error (RE) using the four data sets are shown in Table 5.1. For example, using DATA-3, 95.63% of the predicted values fall within 5% of their actual observed values. The results show that our proposed SVM predicting approach provides highly accurate prediction capability.

Table 5.2 summarizes the results of modeling the temporal inter-relationship among software failure time sequence using our proposed SVM approach. We use the same data sets as cited in Park et al. [29] and Karunanithi et al. [27] in order to

Table 5.1: Performance Results

Predictability ($RE \leq 5\%$)			
DATA-1	DATA-2	DATA-3	DATA-4
87.07%	93.88%	95.63%	95.31%

establish a common baseline for comparison purposes. Park et al. [29] applied failure sequence number as input and failure time as desired output in feed-forward neural network (FFNN). Based on the learning pair of execution time and the corresponding accumulated number of defects disclosed, Karunanithi et al. [27] employed both feed-forward neural network (FFNN) and recurrent neural network (RNN) structures to model the failure process. These results are also summarized in Table 5.2. For example, using our proposed approach with data set DATA-3, the average relative prediction error (AE) is 1.24%. This error is lower than the results obtained by Park et al. [29] (2.38%) using feed-forward neural network, Karunanithi et al. [27] (3.64%) using recurrent neural network, and Karunanithi et al. [27] (6.26%) using feed-forward neural network. In all four data sets, the next-step prediction results show that using our proposed SVM approach yields a lower average relative prediction error compared to other neural network approaches, and is easily implemented to predict failures dynamically. Fig. 5.2, Fig. 5.3, Fig. 5.4 and Fig. 5.5 show the predicted and actual values of the failure time for each data set.

5.4 Application in Short-Term Load Forecasting

5.4.1 Formulation of the SVM-Predictor

In our proposed approach, unlike the traditional mapping characteristics, we model the inter-relationship among power load data independent of other factors, such as

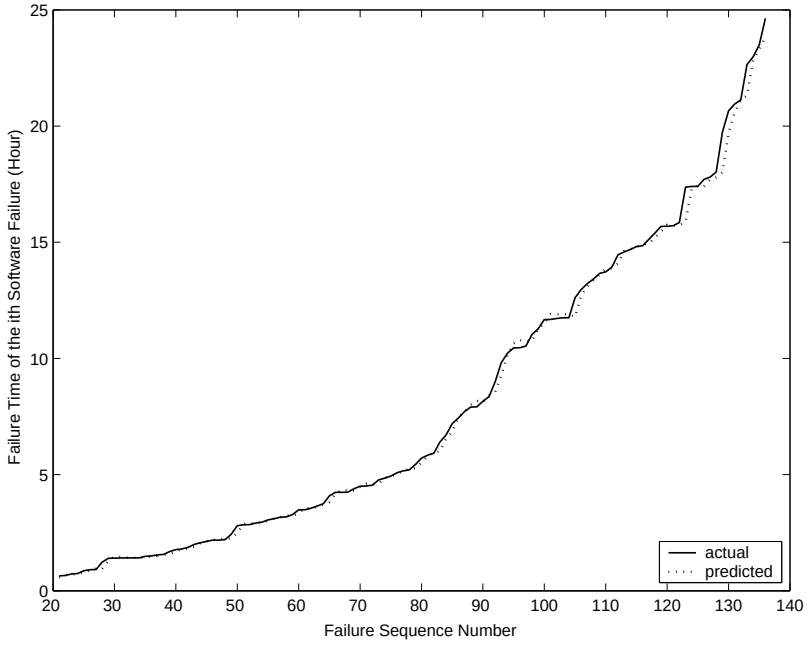

Figure 5.2: Prediction performance using DATA-1.

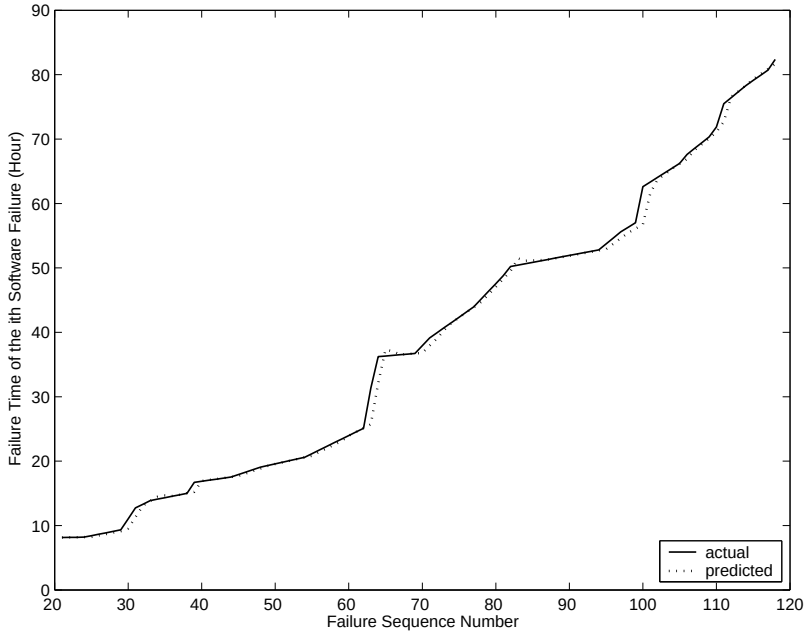

Figure 5.3: Prediction performance using DATA-2.

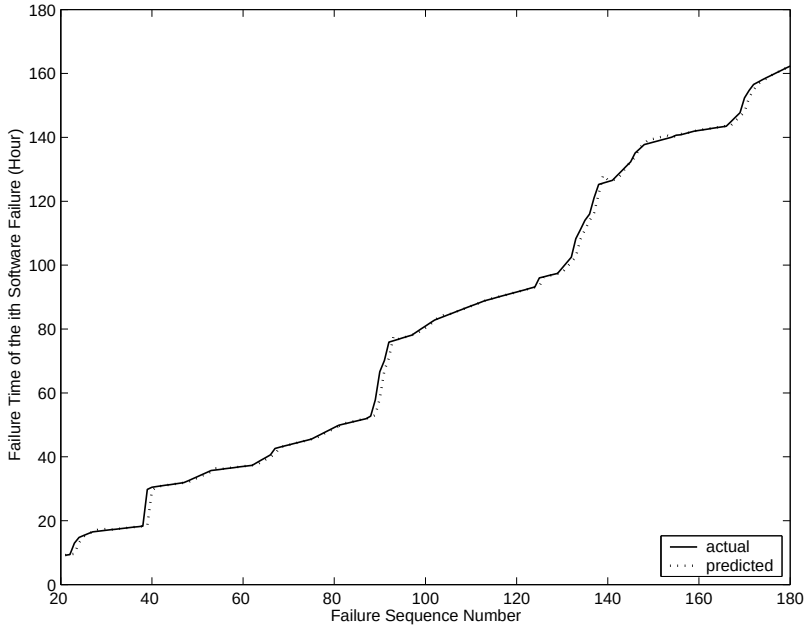

Figure 5.4: Prediction performance using DATA-3.

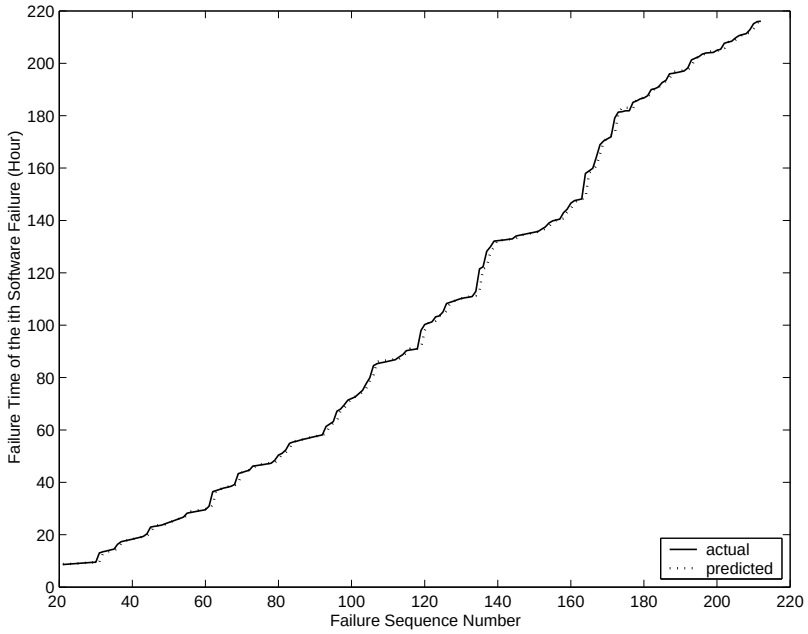

Figure 5.5: Prediction performance using DATA-4.

Table 5.2: Comparison of Average Relative Prediction Error ($AE\%$)

Data Sets	Proposed SVM Approach	FFNN (Ref. [29])	RNN (Ref. [27])	FFNN (Ref. [27])
DATA-1	2.44	2.58	2.05	2.50
DATA-2	1.52	3.32	2.97	5.23
DATA-3	1.24	2.38	3.64	6.26
DATA-4	1.20	1.51	2.28	4.76

humidity and temperature. More specifically, in our proposed short-term load forecasting approach, the input-output pattern fed into the network is the daily average load sequence. For one-day-ahead prediction, the input sequence and the desired output sequence should have one day delay during the training process. The desired objective is to force the network to recognize the one-day-ahead temporal load pattern. A sample input sequence and the corresponding one-day-ahead desired output sequence is defined as:

$$\textit{Input Sequence}: \quad x(d_1), x(d_2), \cdots, x(d_{i-1}), x(d_i), x(d_{i+1}), \cdots$$

$$\textit{Output Sequence}: \quad x(d_2), x(d_3), \cdots, x(d_i), x(d_{i+1}), x(d_{i+2}), \cdots$$

where $x(d_i)$ is the daily average load value at day d_i in the training data sequence. Gaussian kernel function is used in our support vector machine learning process. Once the support vector machine is trained based on the training data sequence, the unseen data sequence will be presented to the network to test the performance.

5.4.2 Performance Metrics

The following statistical metrics are used for comparing prediction performance, namely, Mean Square Error (MSE), Root Mean Square Error (RMSE), and Durbin-Watson d Statistic.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{x}(d_i) - x(d_i))^2 \tag{5.10}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\widehat{x}(d_i) - x(d_i))^2} \tag{5.11}$$

$$\text{Durbin-Watson } d \text{ Statistic} = \frac{\sum_{i=2}^{n}(\widehat{\epsilon}(d_i) - \widehat{\epsilon}(d_{i-1}))^2}{\sum_{i=1}^{n}(\widehat{\epsilon}(d_i))^2} \tag{5.12}$$

where $\widehat{x}(d_i)$ is the predicted value of daily average load, $x(d_i)$ is the actual value of daily average load, and n is the number of days during training and testing. The smaller the values of MSE and RMSE, the closer are the predicted values to the actual values. $\widehat{\epsilon}(d_i)$ is the residual at day d_i and $(\widehat{\epsilon}(d_i) - \widehat{\epsilon}(d_{i-1}))$ represents the difference between a pair of successive residuals. Durbin-Watson d Statistic is commonly used to test for the presence of residual correlation. If the residuals are uncorrelated, the value of d is close to 2, indicating no relationship between $\widehat{\epsilon}(d_i)$ and $\widehat{\epsilon}(d_{i-1})$ and hence implies the confidence in the validity of a model.

5.4.3 Test Results

Table 5.3 summarizes the results of daily average load forecasting using our proposed approach based on the commonly used statistical metrics Mean Square Error

(MSE) and Root Mean Square Error (RMSE). Karayiannis *et al.* [42] applied both feed-forward neural network (FFNN) and cosine radial basis function neural network (RBFNN) approaches for daily average load forecasting based on input variables of past load, temperature and humidity. These results are also summarized in Table 5.3. For example, using our proposed SVM approach with the same testing data set, the Root Mean Square Error (RMSE) is 0.0512. These errors are lower than the results obtained by RBFNN approach (0.1120) and FFNN approach (0.1702) in Karayiannis *et al.* [42] that use multiple input variables. The value of Durbin-Watson d Statistic is 1.7315 in the training data set and is 1.7171 in the test data set of our proposed SVM approach, and hence implies the confidence in the validity of our proposed approach. The results show that our proposed SVM approach yields better generalization capability and lower prediction error compared to other neural network approaches.

Table 5.3: Performance Comparisons

	Performance Metrics	Proposed SVM Approach	RBFNN [42]	FFNN [42]
Training Data	MSE	1.8840×10^6	3.2296×10^7	3.0120×10^7
04/10 – 06/09, 1999	$RMSE^*$	0.0522	0.2160	0.2086
Testing Data	MSE	1.8114×10^6	8.6792×10^6	2.0034×10^7
01/25 – 03/26, 2001	$RMSE^*$	0.0512	0.1120	0.1702
Input Variables Used		Load only	Load, Temperature, Humidity	

* scaled by the mean value of load data.

5.5 Summary

In this chapter, we proposed an adaptive prediction model using support vector machines. The learning process of support vector machines is focused on minimizing an upper bound of the generalization error that includes the sum of the empirical training error and a regularized confidence interval, which eventually results in better generalization performance. Further, this learning process is iteratively and dynamically updated after every occurrence of new data in order to capture the most current feature hidden inside the data sequence.

The proposed approach has been successfully applied and validated on applications related to software reliability prediction and electric power load forecasting. The data sets used for software reliability prediction are four real-time control application and flight dynamic application data sets. The data sets used for short-term load forecasting are the actual power load measurements recorded daily over a period of two years in Berkeley, California. We choose a common baseline to compare the results with related work cited in the literature. Quantitative results show that the proposed approach achieves better prediction accuracy compared to existing approaches. For software reliability prediction, we obtain statistically higher prediction accuracy compared to the existing neural network models. For short-term load forecasting, the proposed approaches yield lower prediction error using minimal number of input variables compared to the existing approaches that use multiple input variables.

The research contributions in this chapter are also summarized in the following articles [80, 81]:

- L. Tian and A. Noore, "Dynamic software reliability prediction: An approach based on support vector machines," *International Journal of Reliability, Quality and Safety Engineering,* vol. 12, no. 4, Aug. 2005.

- L. Tian and A. Noore, "A novel approach for short-term load forecasting using

support vector machines," *International Journal of Neural Systems,* vol. 14, no. 5, pp. 329–335, Oct. 2004.

Chapter 6

Results and Discussions

6.1 Effect of Training Size on Prediction Performance

Most of the traditional neural network approaches use an arbitrary data partition for training and testing [34, 26, 36, 29]. For example, 70% of the collected data were used in the training phase, and the remaining 30% of the collected data were used in the testing phase [34]. Approximately 20% of data were used for training, and all the remaining data were used for validation in [29]. The fixed number of collected data (last 30 failure data) were used for validation purposes in Cai's experiments irrespective of the total data set size [26]. Ho et al. [36] adopted a general 80%-90% training and 10%-20% testing proportion out of total 74 data points, and more specifically, 10 observations were used as out-of-sample testing set. Clearly, there is no rigorous criteria on the training and testing partitioning with respect to the performance validation.

Since our proposed approaches are tailored for dynamic applications, we investigate the effect of the size of training patterns on the next-step prediction error. When

the number of software failure time data is large, the amount of time taken for training with all available data can be a limiting factor. The rate of occurrence of the failure data depends on the maturity of the software. If the failure occurrence rate is high, the amount of time available to accurately predict the next failure is small. With these practical constraints, trade-offs between the size of data to be used for training and the next-step prediction error become critical. Also, when selecting a subset of data for training, we determine if the data from the earliest failure observations or the most recent occurrences yields lower prediction error.

Assuming there are i data points available, the first set of experiment is performed starting with the earliest observations in time to predict the $(i+1)$th data. This is the order in which the software failure time data is generated and is typically used for training purposes. Table 6.1 summarizes the average relative error (AE) of next-step prediction for four real-time control and flight dynamic application data sets when the training pattern size is increased in increments of 10%. The second set of experiment includes data starting with the most recent software failure data to predict the $(i+1)$th data. Table 6.2 summarizes the average relative error of next-step prediction for all data sets.

Table 6.1: Effect of Training Size on Average Relative Error (AE) Starting from the Earliest Data

	Percentage of the available failure data used for training									
Data Set	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
AE%(DATA-1)	33.29	12.69	6.78	57.83	6.67	7.15	7.98	7.71	2.90	2.26
AE%(DATA-2)	82.05	9.22	8.04	3.59	7.35	3.72	9.40	2.05	3.74	2.04
AE%(DATA-3)	5.05	7.61	9.23	12.09	6.98	5.04	3.71	5.90	1.18	0.28
AE%(DATA-4)	74.49	3.81	9.25	5.66	8.69	2.55	1.69	1.70	1.84	0.32

Table 6.2: Effect of Training Size on Average Relative Error (AE) Starting from the Most Recent Data

	Percentage of the available failure data used for training									
Data Set	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
AE%(DATA-1)	14.94	33.10	78.46	4.16	2.30	2.28	34.58	4.16	2.41	2.26
AE%(DATA-2)	965.93	885.14	12.28	4.90	1.25	1.24	1.40	1.39	1.25	1.13
AE%(DATA-3)	0.72	1.48	1.45	0.66	0.40	0.35	0.32	0.40	0.32	0.28
AE%(DATA-4)	0.59	0.54	1.14	0.36	0.34	0.36	0.29	0.29	0.29	0.27

The results from Table 6.1 and Table 6.2 show that when all available data are used, both approaches yield the best prediction performance and there is very little difference between the two approaches. However, if the data set becomes large, or if the rate of failure occurrence is high, then a subset of data selected from the most recent set of data gives lower errors than data selected from the earliest observations. In this research, we use all available time data as training patterns. The number of delayed input neurons and the number of neurons in the hidden layer are computed every time a new data is added to the dataset.

6.2 Results Summary in Software Reliability Prediction

Our choice for using specific performance measures for assessing the predictive accuracy was based on similar measures used by other researchers. We believe it is reasonable to compare our results with existing work using the same data sets and same performance evaluation metrics. This provides us the opportunity to quantitatively gauge the efficacy of our proposed approach. In addition, the relative error (RE) and/or

average relative error (AE) are widely used in [26, 52, 53, 28, 27, 31, 54, 29, 37, 55] for assessment of predictive accuracy.

Table 6.3 summarizes the results of our proposed three approaches when applied to software reliability prediction modeling. Park et al. [29] applied failure sequence number as input and failure time as desired output in feed-forward neural network (FFNN). Based on the learning pair of execution time and the corresponding accumulated number of defects disclosed, Karunanithi et al. [27] employed both feed-forward neural network (FFNN) and recurrent neural network (RNN) structures to model the failure process. These results are also summarized in Table 6.3. For example, using our proposed D - ENN approach with data set DATA-3, the average relative prediction error (AE) is 1.16%; using our proposed RNN - BR approach, the average relative prediction error (AE) is 0.97%; using our proposed SVM approach, the average relative prediction error (AE) is 1.24%. These errors are lower than the results obtained by Park et al. [29] (2.38%) using feed-forward neural network, Karunanithi et al. [27] (3.64%) using recurrent neural network, and Karunanithi et al. [27] (6.26%) using feed-forward neural network. In all four data sets, the next-step prediction results show that using our proposed approaches yields a lower average relative prediction error compared to other neural network approaches.

Table 6.3: Comparison of Average Relative Prediction Error

Data Sets	Comparison of Test Data Sets					
	Proposed D - ENN	Proposed RNN + BR	Proposed SVM	FFNN (Ref. [29])	RNN (Ref. [27])	FFNN (Ref. [27])
DATA-1	2.72	1.83	2.44	2.58	2.05	2.50
DATA-2	2.65	2.06	1.52	3.32	2.97	5.23
DATA-3	1.16	0.97	1.24	2.38	3.64	6.26
DATA-4	1.19	0.98	1.20	1.51	2.28	4.76

6.3 Discussions in Software Reliability Prediction

6.3.1 Data Type Transformation

There are two common types of software failure data: time-between-failures data (time-domain data) and failure-count data (interval-domain data). The individual times at which failure occurred are recorded for time-domain data collection. The time can be either actual failure time or time between successive failures. The interval-domain approach is represented by counting the number of failures occurring during a fixed interval period, such as the number of failures per hour [82, 83].

Our proposed software reliability growth modeling approaches are flexible, which can take different types of data as input. Our approaches were originally intended for using time-domain data (actual failure time) as input to make predictions. If it is assumed that the data collected are interval-domain data, it is possible to develop new models by changing the input-output pair of the network.

One type of software failure data can be transformed into another type in order to meet the input data requirement for a specific model. Interval-domain data can be obtained by counting the number of failures occuring within a specified time period in time-domain data. However, if it is needed to transform interval-domain data to time-domain data, this conversion can be achieved by either randomly or uniformly allocating the failures for the specified time intervals, and then recording the individual times at which failure occurred. Some software reliability tools integrate the capability of data transformation between two data types, such as CASRE (Computer-Aided Software Reliability Estimation) [82].

Similar to what we have proposed in the previous chapters, we can model the inter-relationship among the number of software failures, if the interval-domain data are obtained. For example, suppose x_i is the number of failures in the first i specified time intervals, by using the proposed SVM approach, the input sequence and the

corresponding one-step-ahead desired output sequence can be defined as:

$$\begin{aligned} \textit{Input Sequence}: &\quad x_0, x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots \\ \textit{Output Sequence}: &\quad x_1, x_2, \cdots, x_i, x_{i+1}, x_{i+2}, \cdots \end{aligned}$$

Once the network is trained based on all the currently available history failure data, the one-step-ahead prediction will be obtained.

CASRE [82] software tool was used to obtain the interval-domain data based on the time-domain data. Since interval-domain data can be obtained by counting the number of failures occuring within a specified time period in time-domain data, we believe it is reasonable to take the average inter-failure time as the test interval when we transform time-domain data to interval-domain data. Specifically, the test intervals for the four data sets are 653 seconds (DATA-1), 2513 seconds (DATA-2), 3247 seconds (DATA-3), and 3653 seconds (DATA-4), respectively. We will experiment with both types of software failure data in the following section to illustrate the flexibility of our proposed approaches and their predictive performance.

6.3.2 Modeling Long-Term Behavior

The reason we focused on short-term prediction (one-step-ahead) in this research was to establish a baseline for comparison purposes with other known approaches. We also believe it is more meaningful to make one-step-ahead prediction in certain types of applications in order to make early stage preventive action and avoid catastrophic events.

Meanwhile, it is of great interest for modeling and predicting long-term behavior of software failure process as well. For example, suppose x_i is the number of failures in the first i specified time intervals, and we are using $x_0, x_1, \cdots, x_{i-1}$ to predict x_i. Once the predicted value of x_i, denoted by $\widehat{x}_i$, is obtained, it is then used as input

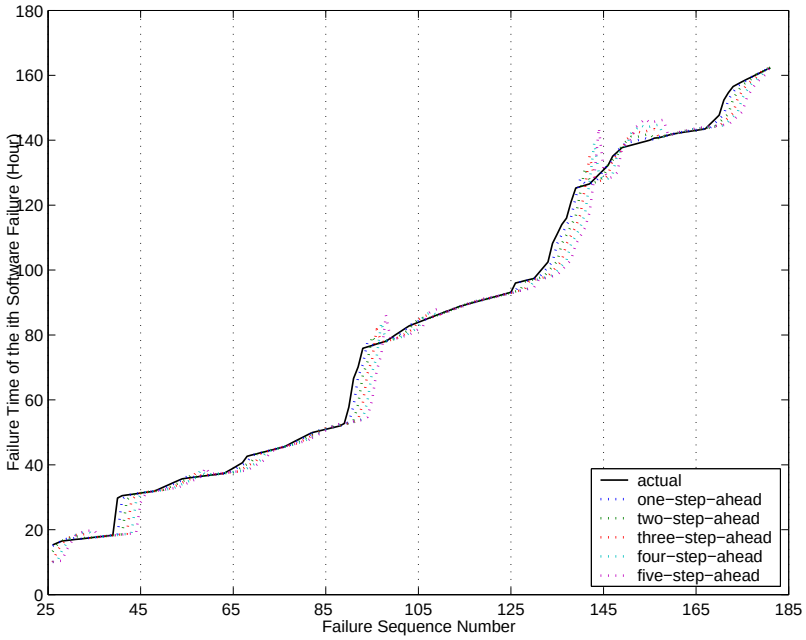

Figure 6.1: Long-term modeling performance using DATA-3 with time-domain data.

to the network to generate the predicted value of x_{i+1}, denoted by $\widehat{x}_{i+1}$. Further, $\widehat{x}_i$ and $\widehat{x}_{i+1}$ are used as input to obtain $\widehat{x}_{i+2}$, and so forth.

We specifically conducted experiments to study the long-term modeling behavior for all four data sets. As an example, Fig. 6.1 and Fig. 6.2 show the long-term (up to five-step-ahead) predicted and actual values of the software failure data for DATA SET # 3 in both time-domain and interval-domain situations using the proposed adaptive support vector machines approach. The solid line represents the actual value. The remaining group of five values on the same X-axis represent the long-term predicted values, respectively.

Accordingly, Table 6.4 and Table 6.5 summarizes the quantitative prediction results for four data sets using the proposed adaptive support vector machines approach. In order to alleviate the impact of different data size and scale, Mean Square Error (MSE) metric is used. The smaller the values of MSE, the closer are the predicted values to the actual values.

Table 6.4: Comparison of Mean Square Error - Failure Time

Data Sets	1 Step Ahead	2 Steps Ahead	3 Steps Ahead	4 Steps Ahead	5 Steps Ahead
DATA-1	0.1057	0.2808	0.5201	0.8218	1.1431
DATA-2	1.2256	3.5840	6.4490	9.8304	13.8229
DATA-3	2.6422	8.0898	16.0485	25.7545	36.8780
DATA-4	2.6861	6.8639	12.1496	18.5289	26.3541

Table 6.5: Comparison of Mean Square Error - Number of Failures

Data Sets	1 Step Ahead	2 Steps Ahead	3 Steps Ahead	4 Steps Ahead	5 Steps Ahead
DATA-1	1.0279	2.5755	4.7273	7.7912	12.0222
DATA-2	1.1960	3.6810	7.2798	12.0711	18.5207
DATA-3	1.5196	5.7466	13.3444	25.3908	42.2251
DATA-4	1.5441	4.7364	9.2970	14.9019	21.2751

Meanwhile, based on the long-term modeling behavior as shown in Fig. 6.1 and Fig. 6.2, we can easily obtain the corresponding inter-failure time and the number of failures in a specified time interval for further interpretation of reliability measures. For example, Once we have the prediction for the number of failures in the first j specified time intervals, y_j, we can obtain the number of failures in the jth specified time interval, z_j, by $(y_j - y_{j-1})$. Similar procedure can also be applied in order to obtain inter-failure time data.

We also experimented with inter-failure time and the number of failures in a specified time interval to study the long-term modeling behavior for all four data sets. As an example, Fig. 6.3 and Fig. 6.4 show the next-step-predicted and actual values of inter-failure time and the number of failures in a specified time interval for DATA SET # 3 using the proposed adaptive support vector machines approach. The solid line represents the actual value. The dotted line represents the next-step-predicted

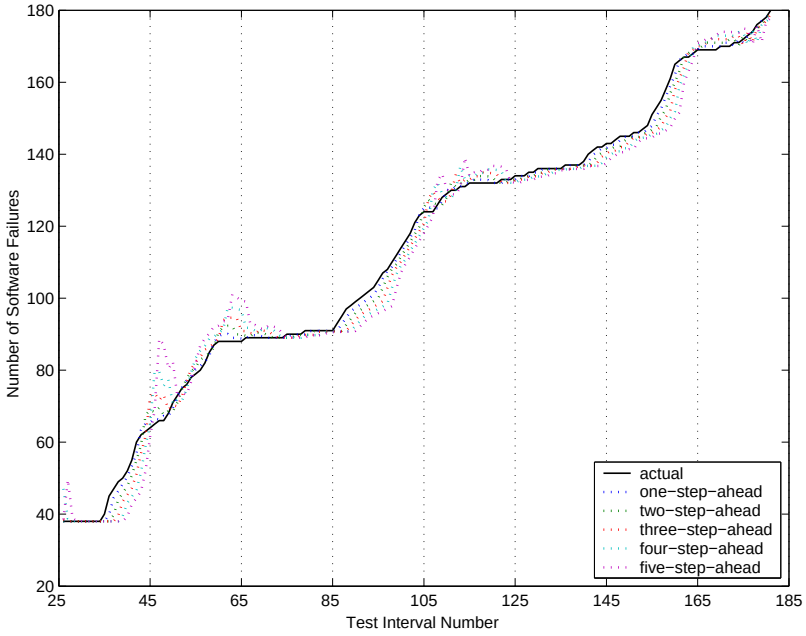

Figure 6.2: Long-term modeling performance using DATA-3 with interval-domain data.

value, respectively.

Accordingly, Table 6.6 and Table 6.7 summarizes the quantitative prediction results for four data sets using the proposed adaptive support vector machines approach. In order to alleviate the impact of different data size and scale, Mean Square Error (MSE) metric is used. The smaller the values of MSE, the closer are the predicted values to the actual values.

Table 6.6: Comparison of Mean Square Error - Inter-Failure Time

Data Sets	1 Step Ahead	2 Steps Ahead	3 Steps Ahead	4 Steps Ahead	5 Steps Ahead
DATA-1	0.1377	0.1647	0.1721	0.1739	0.1822
DATA-2	1.4272	2.3175	2.6395	2.9942	3.4509
DATA-3	2.9516	4.1425	5.7775	7.4656	9.4901
DATA-4	4.0592	4.6417	4.7899	4.6889	5.1295

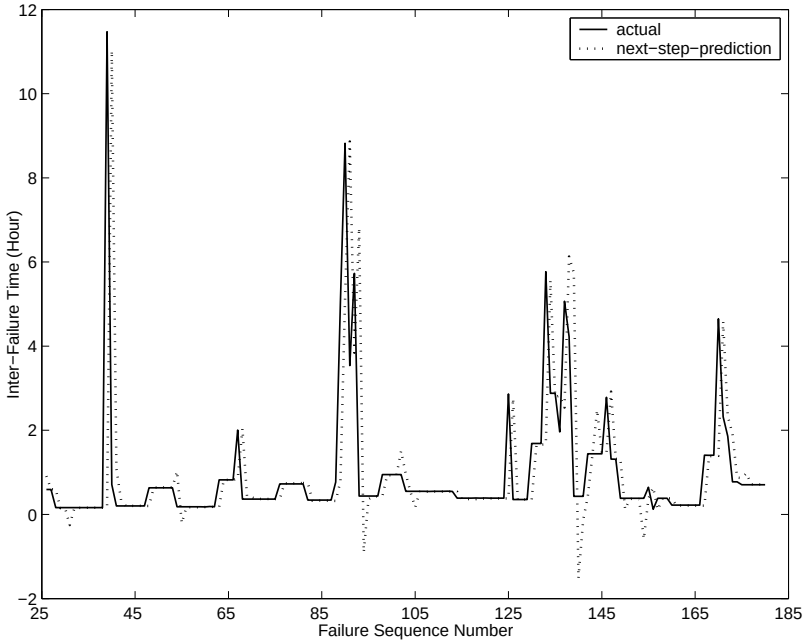

Figure 6.3: Inter-failure time modeling performance using DATA-3.

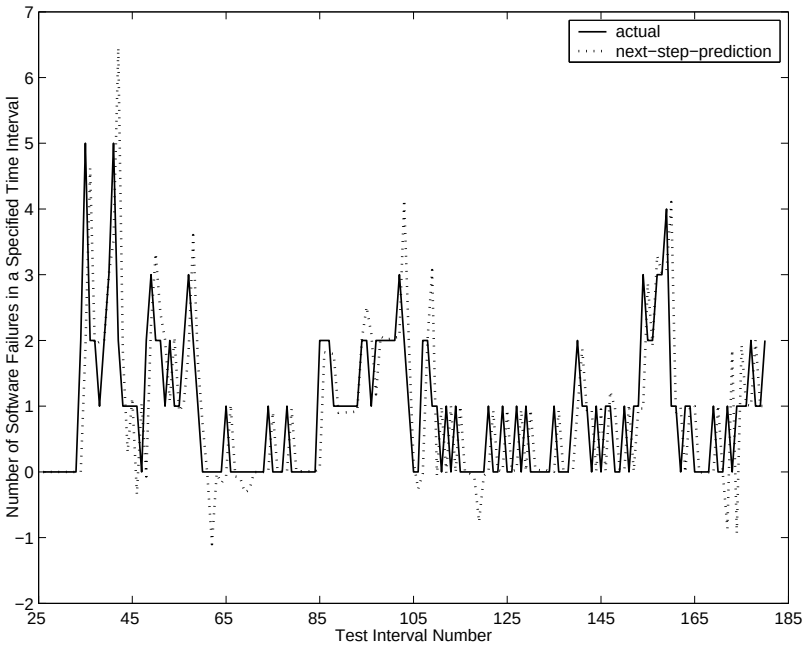

Figure 6.4: Number of failure modeling performance using DATA-3.

Table 6.7: Comparison of Mean Square Error - Number of Failures in Specified Intervals

Data Sets	1 Step Ahead	2 Steps Ahead	3 Steps Ahead	4 Steps Ahead	5 Steps Ahead
DATA-1	1.7671	2.1582	3.0106	4.4908	6.3827
DATA-2	1.0850	1.5328	2.8409	5.0395	8.5478
DATA-3	0.9955	1.8696	3.5994	6.3709	10.5746
DATA-4	1.7266	2.3597	3.0474	3.8957	4.1188

From the experimental results presented in the figures and tables, we can obtain the following observations.

Using the proposed adaptive support vector machines approach as an example, we illustrated that our modeling approaches are flexible to model software failure process with both time-domain data and interval-domain data.

In short-term prediction situations (next-step-prediction), the results exhibit consistently good prediction performance with both time-domain data and interval-domain data independent of different characteristics of data sets.

The prediction performance is compromised with the increasing prediction steps (long-term prediction). This is reasonable because of lacking input of the most recent data patterns. Meanwhile, it is also shown that the effectiveness of long-term prediction depends on the nature of the data set. For instance, it generates satisfactory results for 'smooth' segments in the data sets. However, when it comes near the 'turning point' in the data sets, the effectiveness of long-term prediction is limited.

6.3.3 Comparison with Analytical Software Reliability Models

Software reliability models must cover two different types of situations. One is finding faults and fixing them, and the other is referring to "no fault removal". "No fault

removal" actually means "deferred fault removal". When the failures are identified, the underlying faults will not be removed until the next release [17, 21]. This situation is simple and usually occurs during validation test and operation phase. Most of software reliability models deal with the process of finding and fixing faults that usually occur during software verification process. Thus, if it is assumed that fault removal process does not introduce new faults, the software reliability will increase with the progress of debugging. A software reliability model describing such fault detection and removal phenomenon is called a software reliability growth model [22, 23, 24].

Debugging and testing will reduce the error content but, at the same time, increase development costs. Thus, software reliability assessment is important with respect to determine optimal time to stop testing and also make sure the reliability requirement has been met based on various software reliability measurement metrics [83].

In this section, we first summarize a category of stochastic reliability models for the software failure process based on a Non-homogeneous Poisson Process (NHPP). The category of NHPP software reliability models are realistic models for predicting software reliability and have a very interesting and useful interpretation in debugging and testing the software. Then, we apply those NHPP software reliability models to the four data sets to test their performance and also compare with our proposed modeling approaches.

Notation

$m(t)$	expected number of errors detected by time t
$a(t)$	error content function, i.e., total number of errors in the software including the initial and introduced errors at time t
$b(t)$	error detection rate per error at time t

- Goel-Okumoto Model. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= a(1-e^{-bt}) \\ a(t) &= a \\ b(t) &= b \end{aligned} \tag{6.1}$$

- NHPP Delayed S-Shaped Model. It is the modification of Goel-Okumoto model to make it S-shaped. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= a(1-(1+bt)e^{-bt}) \\ a(t) &= a \\ b(t) &= \frac{b^2 t}{1+bt} \end{aligned} \tag{6.2}$$

- NHPP Inflection S-Shaped Model. It is the same as Goel-Okumoto if $\beta = 0$. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= \frac{a(1-e^{-bt})}{1+\beta e^{-bt}} \\ a(t) &= a \\ b(t) &= \frac{b}{1+\beta e^{-bt}} \end{aligned} \tag{6.3}$$

- Weibull Model. It is the same as Goel-Okumoto model when $c = 1$. $m(t)$ is defined as:

$$m(t) = a(1-e^{-bt^c}) \tag{6.4}$$

- Yamada Exponential Model. It attempts to account for testing-effort. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= a(1-e^{-r\alpha(1-e^{-\beta t})}) \\ a(t) &= a \\ b(t) &= r\alpha\beta e^{-\beta t} \end{aligned} \tag{6.5}$$

- Yamada Rayleigh Model. It attempts to account for testing-effort. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= a(1 - e^{-r\alpha(1-e^{-\beta t^2/2})}) \\ a(t) &= a \\ b(t) &= r\alpha\beta t e^{-\beta t^2/2} \end{aligned} \tag{6.6}$$

- Yamada Imperfect Debugging Model - I . It assumes exponential fault content function and constant error detection rate. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt}) \\ a(t) &= ae^{\alpha t} \\ b(t) &= b \end{aligned} \tag{6.7}$$

- Yamada Imperfect Debugging Model - II. It assumes constant introduction rate α and the error detection rate. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha at \\ a(t) &= a(1 + \alpha t) \\ b(t) &= b \end{aligned} \tag{6.8}$$

- Pham-Nordmann Model. It assumes introduction rate is a linear function of testing time, and the error detection rate function is non-decreasing with an inflection S-shaped model. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha at}{1 + \beta^{-bt}} \\ a(t) &= a(1 + \alpha t) \\ b(t) &= \frac{b}{1 + \beta e^{-bt}} \end{aligned} \tag{6.9}$$

- Pham-Zhang NHPP Model. It assumes introduction rate is exponential function of the testing time, and the error detection rate is non-decreasing with an inflection S-shaped model. $m(t)$ is defined as:

$$\begin{aligned} m(t) &= \frac{1}{1+\beta e^{-bt}}[(c+a)(1-e^{-bt}) - \frac{a}{b-\alpha}(e^{-\alpha t} - e^{-bt})] \\ a(t) &= c + a(1-e^{-\alpha t}) \\ b(t) &= \frac{b}{1+\beta e^{-bt}} \end{aligned} \tag{6.10}$$

Parameter Estimation

Parameter (such as a, b) estimation is of primary importance in software reliability prediction. Once the analytical solutions for $m(t)$ is known for a given model, the parameters in the solution need to be determined. Parameter estimation is achieved by applying a technique of Maximum Likelihood Estimation (MLE), the most important and widely used estimation technique.

Performance Comparison

These NHPP software reliability growth models [84, 85, 86, 87, 88, 89, 90, 17, 91, 92, 93, 94] are applied to all four data sets with interval-domain data (number of failures by time t). All the data points available are used for parameter estimation. Parameter estimation using Maximum Likelihood Estimation (MLE) is done via NHPP Software [83]. Each group of estimated parameters (such as a and b) are then fed back into the corresponding models in order to investigate the capability of each model for describing the software failure process. This procedure is extensively used in Pham's papers [86, 85], book [83], and other researcher's works for checking the 'model fit' performance.

In order to alleviate the impact of different data size and scale, Mean Square Error (MSE) metric is used to compare the number of failures in the first i specified time intervals according to the actual failure data and the predicted number of failures obtained from different models. The smaller the values of MSE, the better the

models describe the software failure process.

Then, we selected the top four to five models with the best performance in each data set and recorded the results in Table 6.8.

Table 6.8: Comparison of Mean Square Error (MSE) Among NHPP Models

Model Name	DATA-1	DATA-2	DATA-3	DATA-4
Delayed S-shaped SRGM [88]	173.6	75.6	171.4	356.4
Inflection S-shaped SRGM [92]	-	13.5	-	-
Weibull [17]	6.8	-	30.8	-
Yamada Imperfect Debugging-I [94]	13.9	11.1	25.1	12.5
Yamada Imperfect Debugging-II [94]	14.4	-	-	21.1
Pham-Nordmann [93]	6.9	-	27.7	19.2
Pham-Zhang [86]	-	23.6	28.0	-

Besides investigating the model describing capabilities of different models, it is also very important to have each model undergone real predictive power test. Traditionally, a portion of the data set will be used for estimating the necessary parameters. Then, the estimated parameters will be applied to the remaining unseen data for the real predictive power test [85].

However, in our case, the proposed approaches are iteratively and dynamically learning based on the continuously increasing data points. Then, the predictions are made based on the most current learned models. Therefore, we need to make some changes to the traditional predictive power test for analytical models. Specifically, for fair comparison, we design a new evolving parameter estimation mechanism for analytical software reliability models in order to be consistent with our proposed evolving approaches. For instance, parameter estimation is conducted iteratively after each newly arriving data point. Then, we make predictions based on the most current estimated parameters using analytical models.

In order to alleviate the impact of different data size and scale, we still use Mean Square Error (MSE) metric to compare the number of failures in the first i specified time intervals according to the actual failure data and the predicted number of failures obtained from different models. The smaller the values of MSE, the better the real prediction performance.

Table 6.9 summarizes the predictive performance comparisons between our proposed modeling approaches and NHPP software reliability models using all four data sets. For NHPP software reliability models, only those with MSE values around 300 or less are shown.

Table 6.9: (MSE) Comparison Between Our Approaches and NHPP Models

Model Name	DATA-1	DATA-2	DATA-3	DATA-4
Delayed S-shaped SRGM [88]	0.6680	0.8869	-	-
Yamada Imperfect Debugging-I [94]	107.3727	309.8692	-	-
Proposed D - ENN	1.8364	3.3113	1.6014	2.8404
Proposed RNN + BR	2.2762	1.8358	0.8891	0.8316
Proposed SVM	0.7625	0.7306	1.2866	1.2097

From the experimental results presented in Table 6.8 and Table 6.9, we can obtain the following observations.

Most of the existing analytical software reliability growth models depend on *a priori* assumptions about the nature of software faults and the stochastic behavior of software failure process. A model that fits well in DATA SET # 1 may not necessarily fit well in DATA SET # 2. Further, those underlying assumptions are often violated in practice. That is the reason why practitioners need to choose the best model as testing progresses for a specific project.

The prediction effectiveness for analytical software reliability models depend on the nature of the data set. For example, as shown in Table 6.9, Delayed S-shaped

software reliability model can achieve considerably better prediction performance in DATA SET # 1 and DATA SET # 2, than in DATA SET # 3 and DATA SET # 4.

Different from the analytical software reliability models, our proposed approaches are based on totally data-driven methods in order to learn from the data set itself and generalize a pattern embedded inside the data. This is corroborated by the facts that our proposed approaches exhibit consistently accurate prediction performance under multiple circumstances.

As also summarized in the previous section, the prediction performance is compromised with the increasing prediction steps. The explanation for this phenomenon is that our proposed approaches are trying to discover very detailed inter-relationship among each data set. They need the feed from the recent data in order to make accurate predictions. However, the emphasis of a analytical software reliability model is focused on the trend or 'the big picture' of the software failure process. Thus, the best fit model (if any), may also provide satisfactory results for extreme-long-term predictions with sparse data.

6.4 Results Summary in Short-Term Load Forecasting

Table 6.10 summarizes the results of daily average load forecasting using our proposed three approaches based on the commonly used statistical metrics Mean Square Error (MSE) and Root Mean Square Error (RMSE). Karayiannis *et al.* [42] applied both feed-forward neural network (FFNN) and cosine radial basis function neural network (RBFNN) approaches for daily average load forecasting based on input variables of past load, temperature and humidity. These results are also summarized in Table 6.10. For example, using our proposed ENN approach with the same testing data set, the Root Mean Square Error (RMSE) is 0.0187. RMSE is 0.0286 using proposed RNN +

BR approach, and is 0.0512 using proposed SVM approach. These errors are lower than the results obtained by RBFNN approach (0.1120) and FFNN approach (0.1702) in Karayiannis *et al.* [42] that use multiple input variables. The results show that our proposed approaches yield better generalization capability and lower prediction error compared to other neural network approaches.

Table 6.10: Performance Comparisons

	Performance Metrics	Proposed D - ENN	Proposed RNN + BR	Proposed SVM	RBFNN [42]	FFNN [42]
Training Data	MSE	5.1942×10^5	5.4832×10^5	1.8840×10^6	3.2296×10^7	3.0120×10^7
04/10 – 06/09, 1999	$RMSE^*$	0.0274	0.0282	0.0522	0.2160	0.2086
Testing Data	MSE	2.4129×10^5	5.6354×10^5	1.8114×10^6	8.6792×10^6	2.0034×10^7
01/25 – 03/26, 2001	$RMSE^*$	0.0187	0.0286	0.0512	0.1120	0.1702
Input Variables Used		Load only	Load only	Load only	Load, Temperature, Humidity	

* scaled by the mean value of load data.

6.5 Discussions in Short-Term Load Forecasting

For comparison purposes, we also study the performance when multiple input variables such as temperature and humidity are considered. More specifically, using the proposed SVM approach, the inputs to the model are the average daily load, temperature, and humidity of the previous days. The corresponding one-day-ahead output is the daily average load for the next day.

Table 6.11 summarizes the results of daily average load forecasting using our proposed approach based on the commonly used statistical metrics Mean Square Error

(MSE) and Root Mean Square Error (RMSE). For example, using our proposed SVM approach with the same testing data set, the Root Mean Square Error (RMSE) is 0.0512 by using load as the only input variable and is 0.0542 by using multiple input variables. These errors are lower than the results obtained by RBFNN approach (0.1120) and FFNN approach (0.1702) in Karayiannis *et al.* [42] that use multiple input variables.

Table 6.11: Performance Comparisons

	Performance Metrics	Proposed SVM Approach	Proposed SVM Approach	RBFNN [42]	FFNN [42]
Training Data	MSE	1.8840×10^6	1.8773×10^6	3.2296×10^7	3.0120×10^7
04/10 – 06/09, 1999	$RMSE^*$	0.0522	0.0521	0.2160	0.2086
Testing Data	MSE	1.8114×10^6	2.0336×10^6	8.6792×10^6	2.0034×10^7
01/25 – 03/26, 2001	$RMSE^*$	0.0512	0.0542	0.1120	0.1702
Input Variables Used		Load only	Load, Temperature, Humidity	Load, Temperature, Humidity	

* scaled by the mean value of load data.

Chapter 7

Conclusion and Future Work

7.1 Contributions and Conclusion

Artificial neural networks are powerful methods for classification and function approximation. Neural networks have better capabilities of fault tolerance, robustness, and adaptability compared to traditional analytical models. However, neural networks have some limitations such as experimental network parameter selection, danger of overfitting, and convergence to local minima instead of global minima.

Optimization of neural network structure design to improve forecasting performance is still a problem. Although researchers have attempted to address these related issues, there is no standard method of designing the neural network structure to solve a specific problem efficiently. Most of the existing neural network approaches use a static structure with a predetermined number of input neurons and a predetermined number of hidden neurons that are established during training. In certain applications, the number of available data increases over time. The fixed network structure does not address the effect on the performance of prediction as the number of data increases, and thus may not provide the best results.

Our research gave three solutions with respect to the above-mentioned limitations

of supervised learning using neural networks. First, we proposed a dynamic learning model using evolutionary connectionist. In certain dynamic applications, the number of available data increases over time. The optimization process determines the number of the input neurons and the number of neurons in the hidden layer. The corresponding globally optimized neural network structure will be iteratively and dynamically reconfigured and updated as new data arrive to improve the prediction accuracy.

Second, we propose improving generalization capability using recurrent neural network and Bayesian regularization. Recurrent neural network has the inherent capability of developing an internal memory, which may naturally extend beyond the externally provided lag spaces. Moreover, by adding a penalty term of sum of connection weights, Bayesian regularization approach is applied to the network training scheme to improve the generalization performance and lower the susceptibility of overfitting.

Third, we proposed an adaptive prediction model using support vector machines. The learning process of support vector machines is focused on minimizing an upper bound of the generalization error that includes the sum of the empirical training error and a regularized confidence interval, which eventually results in better generalization performance. Further, this learning process is iteratively and dynamically updated after every occurrence of new data in order to capture the most current feature hidden inside the data sequence.

All the proposed approaches have been successfully applied and validated on applications related to software reliability prediction and electric power load forecasting. Numerical results show that the proposed approaches have improved existing performance. For software reliability prediction, we obtain statistically higher prediction accuracy compared to the existing neural network based models and NHPP software reliability models. For short-term load forecasting, the proposed approaches

yield lower prediction error using minimal number of input variables compared to the existing approaches that use multiple input variables.

7.2 Future Work

Recent studies show that using testing time as the only influencing factor may not be appropriate for predicting software reliability [95, 85]. Some environmental factors should be integrated. Examples of related environmental factors are program complexity, programmer skills, testing coverage, level of test-team members, and reuse of existing code [95, 96]. Our proposed modeling approach is flexible to incorporate the related environmental factors by changing the input variables. As part of our on-going study, we plan to continue further research in this area.

Also, I intend to expand the research by investigating the applicability of applying the proposed approaches to other application areas, and explore other possibilities of improving dynamic learning and optimization of existing learning techniques.

Bibliography

[1] G. A. Rovithakis, I. Chalkiadakis, and M. E. Zervakis. High-order neural network structure selection for function approximation applications using genetic algorithms. *IEEE Trans. Systems, Man, and Cybernetics - Part B: Cybernetics*, 34(1):150–158, Feb. 2004.

[2] S. H. Ling, Frank H. F. Leung, H. K. Lam, Yim-Shu Lee, and Peter K. S. Tam. A novel genetic-algorithm-based neural network for short-term load forecasting. *IEEE Trans. Industrial Electronics*, 50(4):793–799, Aug. 2003.

[3] Tetsuyuki Takahama and Setsuko Sakai. Structural optimization of neural network by genetic algorithm with damaged genes. In *Proceedings of the 9th International Conference on Neural Information Processing*, pages 1211–1215, Singapore, Nov. 2002.

[4] Abdul Rahman Bohari and Naoki Mizuno. Learning and structural design of feedforward neural networks by employing genetic algorithms. In *Proceedings of the 35th SICE Annual Conference*, pages 1377–1382, Tottori, Japan, July 1996.

[5] K. F. Leung, H. K. Lam, F. H. F. Leung, and P. K. S. Tam. Graffiti commands interpretation for ebooks using a self-structured neural network and genetic algorithm. In *Proceedings of 2002 International Joint Conference on Neural Networks*, pages 2487–2492, Honolulu, HI, May 2002.

[6] Dipankar Dasgupta and Douglas R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96, Baltimore, MD, June 1992.

[7] Zhijun Liu and Masanori Sugisaka. A genetic algorithm approach used to gen-

erate the neural network structures. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 763–768, Kyongju, South Korea, Oct. 1999.

[8] Frank H. F. Leung, H. K. Lam, S. H. Ling, and Peter K. S. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Trans. Neural Networks*, 14(1):79–88, Jan. 2003.

[9] Hirotaka Nakayama, Tadashi Iwata, and Toshiyuki Yamauchi. Learning and structuring of neural networks using genetic algorithm and linear programming. In *Proceedings of 1993 International Joint Conference on Neural Networks*, pages 2702–2705, Nagoya, Japan, Oct. 1993.

[10] Seung-Soo Han and Gary S. May. Optimization of neural network structure and learning parameters using genetic algorithms. In *Proceedings of 8th International Conference on Tools with Artificial Intelligence*, pages 200–206, Toulouse, France, Nov. 1996.

[11] Lefteri H. Tsoukalas and Robert E. Uhrig. *Fuzzy and Neural Approaches in Engineering*. John Wiley & Sons, Inc., New York, NY, 1996.

[12] Vojislav Kecman. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. Complex Adaptive Systems. The MIT Press, 2001.

[13] John Yen and Reza Langari. *Fuzzy Logic, Intelligence, Control, and Information*. Prentice Hall, 1998.

[14] Robert Hochman, Taghi M. Khoshgoftaar, Edward B. Allen, and John P. Hudepohl. Using the genetic algorithm to build optimal neural networks for fault-prone module detection. In *Proceedings of the 7th International Symposium on*

Software Reliability Engineering, pages 152–162, White Plains, NY, Oct. 30–Nov. 2 1996.

[15] Robert Hochman, Taghi M. Khoshgoftaar, Edward B. Allen, and John P. Hudepohl. Evolutionary neural networks: A robust approach to software reliability problems. In *Proceedings of the 8th International Symposium on Software Reliability Engineering*, pages 13–26, Albuquerque, NM, Nov. 1997.

[16] Hai-Feng Liang, Guang-Yu Tu, and Hong-Wei Tang. Application of genetic algorithm neural network for short-term load forecasting of power system. *Power System Technology*, 25(1):49–53, Jan. 2001.

[17] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill Series in Software Engineering and Technology. McGraw-Hill Book Company, 1987.

[18] Amrit L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Trans. Software Eng.*, SE-11(12):1411–1423, Dec. 1985.

[19] M. Xie. *Software Reliability Modelling*. World Scientific Publishing Co., Singapore, 1991.

[20] J. G. Shanthikumar. Software reliability models: A review. *Microelectronics and Reliability*, 23(5):903–943, 1983.

[21] John D. Musa. *Software Reliability Engineering*. McGraw-Hill Osborne Media, New York, NY, 1998.

[22] C. V. Ramamoorthy and F. B. Bastani. Software reliability-status and perspectives. *IEEE Trans. Software Eng.*, SE-8(4):354–370, July 1982.

[23] Shigeru Yamada and Shunji Osaki. Software reliability growth modeling: Models and applications. *IEEE Trans. Software Eng.*, SE-11(12):1431–1437, Dec. 1985.

[24] Nozer D. Singpurwalla and Refik Soyer. Assessing (software) reliability growth using a random coefficient autoregressive process and its ramifications. *IEEE Trans. Software Eng.*, SE-11(12):1456–1464, Dec. 1985.

[25] Kai-Yuan Cai, Chuan-Yuan Wen, and Ming-Lian Zhang. A critical review on software reliability modeling. *Reliability Engineering and System Safety*, 32(3):357–371, 1991.

[26] Kai-Yuan Cai, Lin Cai, Wei-Dong Wang, Zhou-Yi Yu, and David Zhang. On the neural network approach in software reliability modeling. *The Journal of Systems and Software*, 58(1):47–62, 2001.

[27] Nachimuthu Karunanithi, Darrell Whitley, and Yashwant K. Malaiya. Prediction of software reliability using connectionist models. *IEEE Trans. Software Eng.*, 18(7):563–574, July 1992.

[28] N. Karunanithi, D. Whitley, and Y. K. Malaiya. Using neural networks in reliability prediction. *IEEE Software*, 9(4):53–59, July 1992.

[29] Joong-Yang Park, Sang-Un Lee, and Jae-Heung Park. Neural network modeling for software reliability prediction from failure time data. *Journal of Electrical Engineering and Information Science*, 4(4):533–538, August 1999.

[30] Lev V. Utkin, Sergey V. Gurov, and Maxim I. Shubinsky. A fuzzy software reliability model with multiple-error introduction and removal. *International Journal of Reliability, Quality and Safety Engineering*, 9(3):215–227, 2002.

[31] N. Karunanithi and Y. K. Malaiya. The scaling problem in neural networks for software reliability prediction. In *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, pages 76–82, Research Triangle Park, NC, Oct. 1992.

[32] Wan Azizun Adnan and Mashkuri Hj. Yaacob. An integrated neural-fuzzy system of software reliability prediction. In *Proceedings of the 1st International Conference on Software Testing, Reliability and Quality Assurance*, pages 154–158, New Delhi, India, Dec. 1994.

[33] W. A. Adnan, M. Yaakob, R. Anas, and M. R. Tamjis. Artificial neural network for software reliability assessment. In *2000 TENCON Proceedings of Intelligent Systems and Technologies for the New Millennium*, pages 446–451, Kuala Lumpur, Malaysia, Sep. 2000.

[34] Sultan H. Aljahdali, Alaa Sheta, and David Rine. Prediction of software reliability: A comparison between regression and neural network non-parametric models. In *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications*, pages 470–473, Beirut, Lebanon, June 2001.

[35] Sultan H. Aljahdali, Alaa Sheta, and David Rine. Predicting accumulated faults in software testing process using radial basis function network models. In *Proceedings of the ISCA 17th International Conference on Computers and their Applications*, pages 26–29, San Francisco, CA, April 2002.

[36] S. L. Ho, M. Xie, and T. N. Goh. A study of the connectionist models for software reliability prediction. *Computers and Mathematics with Applications*, 46(7):1037–1045, Oct. 2003.

[37] Renate Sitte. Comparison of software-reliability-growth predictions: Neural networks vs parametric-recalibration. *IEEE Trans. Reliability*, 48(3):285–291, Sep. 1999.

[38] K. Metaxiotis, A. Kagiannas, D. Askounis, and J. Psarras. Artificial intelligence in short term electric load forecasting: A state-of-the-art survey for the researcher. *Energy Conversion and Management*, 44(9):1525–1534, June 2003.

[39] Hesham K. Alfares and Mohammad Nazeeruddin. Electric load forecasting: Literature survey and classification of methods. *International Journal of Systems Science*, 33(1):23–34, Jan. 2002.

[40] Henrique Steinherz Hippert, Carlos Eduardo Pedreira, and Reinaldo Castro Souza. Neural networks for short-term load forecasting: A review and evaluation. *IEEE Trans. Power Systems*, 16(1):44–55, Feb. 2001.

[41] Hong Chen, Claudio A. Canizares, and Ajit Singh. Ann-based short-term load forecasting in electricity markets. In *Proceedings of IEEE Power Engineering Society 2001 Winter Meeting*, pages 411–415, Columbus, OH, Jan. 28 - Feb. 1 2001.

[42] Nicolaos B. Karayiannis, Mahesh Balasubramanian, and Heidar A. Malki. Evaluation of cosine radial basis function neural networks on electric power load forecasting. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2100–2105, Portland, OR, July 2003.

[43] Yasuhiko Dote and Seppo J. Ovaska. Industrial applications of soft computing: A review. *Proceedings of the IEEE*, 89(9):1243–1265, Sep. 2001.

[44] S. Barghinia, P. Ansarimehr, H. Habibi, and N. Vafadar. Short term load forecasting of iran national power system using artificial neural network. In *Proceedings of the 2001 Power Tech*, Porto, Portugal, Sep. 2001.

[45] W. Charytoniuk and M. S. Chen. Neural network design for short-term load forecasting. In *Proceedings of the International Conference on Electric Utility Deregulation and Restructuring and Power Technologies*, pages 554–561, London, UK, April 2000.

[46] H. C. Huang, R. C. Hwang, and J. G. Hsieh. Short-term power load forecasting by

non-fixed neural network model with fuzzy bp learning algorithm. *International Journal of Power and Energy System*, 22(1):50–57, 2002.

[47] Yan-Xi Yang, Ding Liu, Qi Li, and Gang Zheng. Short term load forecasting using a multilayer neural network with bp-ga mixed algorithm. *Information and Control*, 21(3):284–288, June 2002.

[48] Worawit Tayati and Wanchai Chankaipol. Substation short term load forecasting using neural network with genetic algorithm. In *Proceedings of the 2002 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, pages 1787–1790, Beijing, China, Oct. 2002.

[49] Dipti Srinivasan. Evolving artificial neural networks for short term load forecasting. *Neurocomputing*, 23(1–3):265–276, Dec. 1998.

[50] Ta-Peng Tsao, Gwo-Ching Liao, and Shi-Hsien Chen. Short-term load forecasting using neural networks and evolutionary programming. In *Proceedings of Fifth International Power Engineering Conference*, pages 743–748, Singapore, May 2001.

[51] Lefteri H. Tsoukalas and Robert E. Uhrig. *Fuzzy and Neural Approaches in Engineering*, chapter 11. Practical Aspects of Using Neural Networks, pages 385–405. John Wiley & Sons, Inc., New York, NY, 1996.

[52] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. *Software Reliability: Measurement, Prediction, Application*, chapter 13. Comparison of Software Reliability Models, pages 383–407. McGraw-Hill Series in Software Engineering and Technology. McGraw-Hill Book Company, 1987.

[53] John D. Musa. *Software Reliability Engineering*, chapter 8. Software Reliability Models, pages 267–290. McGraw-Hill Osborne Media, New York, NY, 1998.

[54] N. Karunanithi. A neural network approach for software reliability growth modeling in the presence of code churn. In *Proceedings of the 4th International Symposium on Software Reliability Engineering*, pages 310–317, Denver, CO, Nov. 1993.

[55] C. Stringfellow and A. Amschler Andrews. An empirical method for selecting software reliability growth models. *Empirical Software Engineering*, 7(4):319–343, Dec. 2002.

[56] Liang Tian and Afzel Noore. Evolutionary neural network modeling for software cumulative failure time prediction. *Reliability Engineering and System Safety*, 87(1):45–51, Jan. 2005.

[57] Liang Tian and Afzel Noore. On-line prediction of software reliability using an evolutionary connectionist model. *Journal of Systems and Software*, In press, available online Oct. 2004.

[58] Liang Tian and Afzel Noore. Short-term load forecasting using optimized neural network with genetic algorithm. In *Proceedings of the 8th International Conference on Probabilistic Methods Applied to Power Systems*, pages 135–140, Ames, IA, Sep. 2004.

[59] Morten With Pedersen. *Optimization of Recurrent Neural Networks for Time Series Modeling*. PhD thesis, Technical University of Denmark, Denmark, 1997.

[60] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, April-June 1990.

[61] Aninda Bhattacharya, Alexander G. Parlos, and Amir F. Atiya. Prediction of mpeg-coded video source traffic using recurrent neural networks. *IEEE Trans. Signal Processing*, 51(8):2177–2190, Aug. 2003.

[62] Ramazan Gencay and Min Qi. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE Trans. Neural Networks*, 12(4):726–734, July 2001.

[63] David J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, May 1992.

[64] F. Dan Foresee and Martin T. Hagan. Gauss-newton approximation to bayesian learning. In *Proceedings of the 1997 IEEE International Conference on Neural Networks*, pages 1930–1935, Houston, TX, June 1997.

[65] Tin-Yau Kwok and Dit-Yan Yeung. Bayesian regularization in constructive neural networks. In *Proceedings of the 1996 International Conference on Artificial Neural Networks*, pages 557–562, Bochum, Germany, July 1996.

[66] M. Ishikawa. Structural learning and bayesian regularization in neural networks. In *Proceedings of 1996 International Conference on Neural Information Processing*, pages 1377–1380, Hong Kong, China, Sep. 1996.

[67] C. G. Chua and A. T. C. Goh. A hybrid bayesian back-propagation neural network approach to multivariate modeling. *International Journal for Numerical and Analytical Methods in Geomechanics*, 27(8):651–667, July 2003.

[68] Liang Tian and Afzel Noore. Software reliability prediction using recurrent neural network with bayesian regularization. *International Journal of Neural Systems*, 14(3):165–174, June 2004.

[69] Vladimir Vapnik, Steven E. Golowich, and Alex Smola. Support vector method for function approximation, regression estimation and signal processing. In *Advances in Neural Information Processing Systems 9. Proceedings of the 1996 Conference*, pages 281–287, Denver, CO, Dec. 1996.

[70] L.J. Cao and Francis E.H. Tay. Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Trans. Neural Networks*, 14(6):1506–1518, Nov. 2003.

[71] Jonathan Robinson and Vojislav Kecman. Combining support vector machine learning with the discrete cosine transform in image compression. *IEEE Trans. Neural Networks*, 14(4):950–958, July 2003.

[72] Lijuan Cao and Francis E.H. Tay. Financial forecasting using support vector machines. *Neural Computing & Applications*, 10(2):184–192, 2001.

[73] Francis E.H. Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

[74] Francis E.H. Tay and L.J. Cao. Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48:847–861, 2002.

[75] Vladimir N. Vapnik. An overview of statistical learning theory. *IEEE Trans. Neural Networks*, 10(5):988–999, Sep. 1999.

[76] Kyoung jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

[77] Vladimir N. Vapnik. *Statistical Learning Theory*, chapter 10.8. Examples of SV Machines for Pattern Recognition. Adaptive and Learning Systems for Signal Processing, Communication, and Control. John Wiley & Sons, Inc., New York, NY, 1998.

[78] Francis E.H. Tay and L.J. Cao. epsilon -descending support vector machines for financial time series forecasting. *Neural Processing Letters*, 15(2):179–195, 2002.

[79] Lijuan Cao. Support vector machines experts for time series forecasting. *Neurocomputing*, 51:321–339, 2003.

[80] Liang Tian and Afzel Noore. Dynamic software reliability prediction: An approach based on support vector machines. *International Journal of Reliability, Quality and Safety Engineering*, 12(4), Aug. 2005.

[81] Liang Tian and Afzel Noore. A novel approach for short-term load forecasting using support vector machines. *International Journal of Neural Systems*, 14(5):329–335, Oct. 2004.

[82] Michael R. Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, NY, 1996.

[83] Hoang Pham. *Software Reliability*. Springer, 2000.

[84] Xuemei Zhang and Hoang Pham. Comparisons of nonhomogeneous poisson process software reliability models and its applications. *International Journal of Systems Science*, 31(9):1115–1123, 2000.

[85] Hoang Pham. Software reliability and cost models: Perspectives, comparison, and practice. *European Journal of Operational Research*, 149(3):475–489, Sep. 2003.

[86] Hoang Pham and Xuemei Zhang. An nhpp software reliability model and its comparison. *International Journal of Reliability, Quality and Safety Engineering*, 4(3):269–282, 1997.

[87] Amrit L. Goel and K. Okumoto. Time-dependent error-detection rate model for software and other performance measures. *IEEE Trans. Reliability*, 28:206–211, 1979.

[88] S. Yamada, M. Onha, and S. Osaki. S-shaped reliability growth modeling for software error detection. *IEEE Trans. Reliability*, 12:475–484, 1983.

[89] D. Kececioglu. *Reliability Engineering Handbook*. Prentice-Hall, 1991.

[90] B. Littlewood. Stochastic reliability growth: A model for fault removal in computer programs and hardware design. *IEEE Trans. Reliability*, 12:313–320, 1981.

[91] S. Yamada, H. Ohtera, and H. Narihisa. Software reliability growth models with testing effort. *IEEE Trans. Reliability*, 4:19–23, 1986.

[92] S. Yamada. Software quality/reliability measurement and assessment: software reliability growth models and data analysis. *Journal of Information Processing*, 14(3):254–266, 1991.

[93] H. Pham and L. Nordmann. A generalized nhpp software reliability model. In *Proceedings of the 3rd ISSAT International Conference on Reliability & Quality in Design*, pages 116–120, 1997.

[94] S. Yamada, K. Tokuno, and S. Osaki. Imperfect debugging models with fault introduction rate for software reliability assessment. *International Journal of Systems Science*, 23(12), 1992.

[95] Hoang Pham. *Software Reliability*, chapter 8. Software Reliability Models with Environmental Factors, pages 199–220. Springer, 2000.

[96] Xuemei Zhang, Mi-Young Shin, and Hoang Pham. Exploratory analysis of environmental factors for enhancing the software reliability assessment. *Journal of Systems and Software*, 57(1):73–78, April 2001.