Scientific Research Publishing

Journal of Software Engineering and Applications, 2017, 10, 288-309
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

A Neuro-Based Software Fault Prediction with Box-Cox Power Transformation

Momotaz Begum, Tadashi Dohi

Department of Information Engineering, Hiroshima University, Hiroshima, Japan
Email: momotaz.2k3@gmail.com, dohi@rel.hiroshima-u.ac.jp

How to cite this paper: Begum, M. and Dohi, T. (2017) A Neuro-Based Software Fault Prediction with Box-Cox Power Transformation. *Journal of Software Engineering and Applications*, **10**, 288-309.
https://doi.org/10.4236/jsea.2017.103017

Received: February 3, 2017
Accepted: March 27, 2017
Published: March 30, 2017

Abstract

Software fault prediction is one of the most fundamental but significant management techniques in software dependability assessment. In this paper we concern the software fault prediction using a multilayer-perceptron neural network, where the underlying software fault count data are transformed to the Gaussian data, by means of the well-known Box-Cox power transformation. More specially, we investigate the long-term behavior of software fault counts by the neural network, and perform the multi-stage look ahead prediction of the cumulative number of software faults detected in the future software testing. In numerical examples with two actual software fault data sets, we compare our neural network approach with the existing software reliability growth models based on nonhomogeneous Poisson process, in terms of predictive performance with average relative error, and show that the data transformation employed in this paper leads to an improvement in prediction accuracy.

Keywords

Software Reliability, Artificial Neural Network, Box-Cox Power Transformation, Long-Term Prediction, Fault Count Data, Empirical Validation

1. Introduction

As software systems play an increasingly important role in our lives, their complexity and size continue growing. The increased complexity of software systems makes the assurance of software quality much difficult. In fact many of software applications require critical functionality because of their increasing size and complexity. Software reliability is an important facet of software quality, and is defined as the probability of failure-free software operation for a specified period of time in a specified environment. For the purpose of quantitative assessment, software reliability growth models (SRGMs) have been widely used during the last four decades [1] [2] [3] [4] [5]. It is worth mentioning that SRGMs specify

any parametric form of random processes that describe the behavior of software failure occurrence with respect to time. Since SRGMs are essentially stochastic models with abstractions, they must be built under several simplified mathematical assumptions, and, at the same time, their parameter estimation with the fault count data observed in software testing is not a trivial task. Because the maximum likelihood estimation, which is commonly used, is reduced to a multimodal nonlinear optimization problem with constraints, and requires more computation efforts. Another problem on SRGMs is the model selection from a great number of SRGM candidates. During the last four decades, over three hundred SRGMs have been proposed in the literature. The conclusion from the empirical research suggests that the best SRGM in terms of the goodness-of-fit performance depends on the kind of software fault count data. In other words, there does not exist the best SRGM which can fit every software fault count data. Sharma *et al.* [6] proposed a selection method of the best SRGM by using the concept of *distance*. Unfortunately, it is noted that the best SRGM which can fit the past observation experienced before does not always provide the best *prediction* model for the future (or remaining) testing period.

Apart from SRGMs based on stochastic modeling, artificial neural network (ANN) has gained much popularity to deal with non-linear phenomena arising in applications to time series forecasting, pattern recognition, function approximation, etc. Comparing with non-trivial stochastic models, it is easy to implement the ANN for the software fault prediction, since the feed-forward back-propagation (BP) type of learning algorithm can be widely used to estimate the internal parameters, such as connection weights. Software fault prediction using the ANN was proposed first by Karunanithi *et al.* [7], Karunanithi and Malaiya [8] [9]. They applied simple multi-layered perceptron (MLP) feed forward neural networks, which enjoy a universal approximation ability [10] to represent an arbitrary nonlinear mapping with any degree of accuracy, in the prediction of software fault-detection time in software testing. Since their seminal contributions, the ANN approach has been frequently applied to different estimation/prediction problems in software engineering. Khoshgoftaa *et al.* [11] [12], Khoshgoftaar and Szabo [13] considered different problems to identify fault-prone modules in software quality assessment. For more recent survey, see Vashisht *et al.* [14].

While, it should be pointed out that the ANN approach has some drawbacks in application to software fault prediction. First, there is no efficient way to determine the best neural network architecture in each application domain. Even though the number of input neurons and output neurons may be determined from physical requirements, the number of hidden layers and hidden neurons significantly influences the prediction performance in MLP feed forward neural networks. In many applications, these sizes must be determined through trial-and-error heuristics in pre-experiments. In other words, the predictive performance of software fault count strongly depends on the ANN architecture assumed for the analysis. Second, the ANN is a kind of nonlinear regression mod-

el, but can be regarded as a deterministic model to output the deterministic values as estimates or predictions. Dissimilar to the familiar SRGMs, it is impossible to quantify the software reliability as a probability by applying the common ANN approach. This feature penalizes us to use the ANN when quantifying the software reliability measures such as the software reliability, mean time to software failure, etc. On the other hand, though the ANN is a simple connectionist model depending on the architecture, it can be considered as a statistically nonparametric model without specific model assumptions. As mentioned above, a huge number of SRGMs have been developed in the literature [15]-[25], but almost all of them are based on some parametric assumptions which cannot be validated for every fault count data. In that sense, the ANN approach can be viewed as one of nonparametric models with no specific model assumptions. In fact, the ANN approach provides a data-driven modeling framework and can bridge several kinds of machine learning techniques. Yang *et al.* [26] applied a model mining technique to provide a generic data-driven SRGM. Xiao and Dohi [27] proposed a nonparametric wavelet method to estimate nonhomogeneous Poisson process (NHPP)-based SRGM. Cheng *et al.* [28] considered a multistep-ahead a time series prediction with the MLP approach. Park *et al.* [29] also compared several data-driven approaches with the existing SRGMs. Recently, Begum and Dohi [30] [31] applied an idea on multiple-input multiple-output (MIMO) neural computation by Park *et al.* [29], and proposed a refined neural network approach to predict the long-term behavior of software fault count with the grouped data. They impose an assumption that the underlying fault count process obeys the Poisson law with an unknown mean value function, and propose to utilize several data transform methods from the Poisson count data to the Gaussian data. However, it is worth noting that even the Poisson assumption can be regarded as a simplified assumption in SRGM modeling and has not been validated empirically.

In this paper we refine the existing MLP approach as a data-driven model from the view point of software fault prediction. In particular, we deal with the grouped data which consists of the number of software fault counts detected at each testing date, although the existing MLP approaches focus on only the software fault-detection time data which are not easily available in actual software testing. We apply the well-known data transformation technique called the Box-Cox power transformation [32], from an arbitrary probability law to the Gaussian law, and transform the underlying software fault prediction problem to a nonlinear Gaussian regression problem with the MLP. First, Tukey [33] introduced a family of power transformations such that the transformed values obey a monotonic function of the observations over some admissible range, where an arbitrary transformation parameter is involved in the power transformations. Box and Cox [32] proposed the maximum likelihood as well as the Bayesian methods for estimation of the parameter, and derived the asymptotic distribution of the likelihood ratio to test some hypotheses about the parameter. The main contributions by Box and Cox [32] are two-folds: The first one is to com-

pute the profile likelihood function and to obtain an approximate confidence interval from the asymptotic property of the maximum likelihood estimator. The second one is to ensure that the probability model is fully identifiable in the Bayesian approach. In neural computation contexts, Dashora *et al.* [34] used the Box-Cox power transformation for data driven prediction models with single output neuron in MLP and analyzed the intermittent stream flow for Narmada river basin. They compared the MLP with seasonal autoregressive integrated moving average and Thomas-Fiering models. Sennaroglu and Senvar [35] evaluated the process capability indices in industry and compared the Box and Cox power transformation with weighted variance methods in the Weibull distribution model in terms of the process variation with the product specifications. In this way, an applicability of the Box and Cox power transform can be recognized in many research fields.

In this paper we transform the discrete integer-valued data which denote the cumulative number of software faults detected in software system testing to the Gaussian data by the Box-Cox transform. Next we input the transformed data into the MIMO MLP to make the long-term prediction of the software fault count. Note that almost all papers in past, [7] [8] [9] [11] [12] [13] [36] [37] [38] [39] [40] just considered the one-stage look ahead prediction. Hence, our challenge with MIMO and the Box-Cox power transformation overcomes the limitation for the existing neuro-based approaches. Recently, the same authors [30] considered a different transformation technique for the software fault prediction, where only the one-stage look ahead prediction is made in terms of prediction interval. Also, though they implicitly assume the Poisson law for the underlying fault count data, we do not restrict the Poisson law for the software fault count data, because the Box-Cox transform is a general data transformation scheme. The present paper is organized as follows. In Section 2, we overview the SRGMs based on nonhomogeneous Poisson process (NHPP). Section 3 introduces a refined ANN for the purpose of prediction, where the Box-Cox power transformation is applied in the pre-processing of input data. In Section 4, we analyze two actual software fault count data sets and compare our refined neural network approach with eleven NHPP-based SRGMs [41] from the view point of predictive performance with relative average error. We show here that our ANN approach affords a more appropriate prediction device and tends to have an enhanced performance from the standpoint of predictability. Finally the paper is concluded with remarks in Section 5.

2. NHPP-Based Software Reliability Modeling

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

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

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

$$\Pr\{X(t)=x\}=\frac{\{\Lambda(t;\theta)\}^x}{x!}\exp\{-\Lambda(t;\theta)\}, \tag{1}$$

where

$$\Lambda(t;\theta)=\int_0^t \varnothing(x;\theta)\mathrm{d}x \tag{2}$$

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

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

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

$$\mathrm{LLF}(\theta)=\sum_i^n\left((x_i-x_{i-1})\log\{\Lambda(t_i;\theta)-\Lambda(t_{i-1};\theta)\}-\log\{(x_i-x_{i-1})!\}\right)-\Lambda(t_n;\theta), \tag{3}$$

where $\Lambda(0;\theta)=0$, $x_0=0$ and $t=t_n$ for simplification. The maximum likelihood estimate of model parameter $\hat{\theta}$, can be obtained by maximizing Equation (3) with respect to the model parameter θ. Once the model parameter is estimated, our next concern is to predict the future value of the intensity function or the mean value function at an arbitrary time $t_{n+l}(l=1,2,\cdots)$, where l denotes the prediction length [42]. In parametric modeling, the prediction at time t_{n+l} is easily done by substituting estimated model parameter $\hat{\theta}$ into the time evolution $\Lambda(t;\theta)$, where the unconditional and conditional mean value functions at an arbitrary future time t_{n+l} are given by

Table 1. NHPP-based SRGMs.

Model (Abbr.)	Mean value function
Exponential (exp) [17]	$\Lambda(t)=aF(t)$ $F(t)=a\{1-\exp(-bt)\}$
Gamma (gamma) [24]	$\Lambda(t)=aF(t)$ $F(t)=\int_0^t \frac{c^b s^{b-1}\exp(-cs)}{\Gamma(b)}\mathrm{d}s$
Pareto (pareto) [15] [20]	$\Lambda(t)=aF(t)$ $F(t)=1-\left(\frac{c}{t+c}\right)^b$
Truncated normal (tnorm) [23]	$\Lambda(t)=a\frac{F(t)-F(0)}{1-F(0)}$ $F(t)=\frac{1}{\sqrt{2\pi b}}\int_{-\infty}^{t}\exp\left(-\frac{(s-c)^2}{2b^2}\right)\mathrm{d}s$
Log normal (lnorm) [16] [23]	$\Lambda(t)=aF(\log t)$ $F(t)=\frac{1}{\sqrt{2\pi b}}\int_{-\infty}^{t}\exp\left(-\frac{(s-c)^2}{2b^2}\right)\mathrm{d}s$
Truncated logistic (tlogist) [21]	$\Lambda(t)=a\frac{F(t)-F(0)}{1-F(0)}$ $F(t)=\frac{1}{1+\exp\left(-\frac{t-c}{b}\right)}$
Log logistic (llogist) [19]	$\Lambda(t)=aF(\log t)$ $F(t)=\frac{1}{1+\exp\left(-\frac{t-c}{b}\right)}$
Truncated extreme value maximum (txvmax) [22]	$\Lambda(t)=a\frac{F(t)-F(0)}{1-F(0)}$ $F(t)=\exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$
Log extreme value maximum (lxvmax) [22]	$\Lambda(t)=aF(\log t)$ $F(t)=\exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$
Truncated extreme value minimum(txvmin) [22]	$\Lambda(t)=a\frac{F(0)-F(t)}{F(0)}$ $F(t)=\exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$
Log extreme value minimum (lxvmin) [18] [22]	$\Lambda(t)=a\left(1-F(-\log t)\right)$ $F(t)=\exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$

$$\Lambda\left(t_{n+l};\hat{\theta}\right)=\int_0^{t_{n+l}}\varnothing\left(x;\hat{\theta}\right)\mathrm{d}x, \tag{4}$$

$$\begin{aligned}\Lambda\left(t_{n+l}\middle|X\left(t_n\right)=x_n;\hat{\theta}\right)&=x_n+\int_{t_n}^{t_{n+l}}\varnothing\left(x;\hat{\theta}\right)\mathrm{d}x\\&=x_n+\Lambda\left(t_{n+l};\hat{\theta}\right)-\Lambda\left(t_n;\hat{\theta}\right).\end{aligned} \tag{5}$$

When the mean value function is unknown, a few nonparametric approaches have been developed [43] [44]. However, it should be noted that those approaches can deal with the fault-detection time data, but do not work for prediction in future. The wavelet-based method in [27] can treat the grouped data, but fails to make the long-term prediction in nature. In the following section, we use an elementary MLP for the purpose of the long-term software fault prediction.

3. A Refined MLP Architecture

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

3.1. Preliminary Set-Up

In the common neural computation, it is noted that the neural network including the simplest MLP with only one output neuron is regarded as a nonlinear regression model, where the explanatory variables are randomized by the Gaussian white nose. In other words, the output data in the MLP is implicitly assumed to be a realization of a nonlinear Gaussian model. On the other hand, since one handles the fault count data as integer values in the software fault prediction, the underlying data shall be transformed to the Gaussian data in advance. Such a pre-data processing is common in the wavelet shrinkage estimation [27] although it specifies the underlying data as the Poisson data. According to the idea in the literature [27], we apply the Box-Cox power transformation technique [32] from an arbitrary random data to the Gaussian data. As mentioned in Section 1, Box and Cox [32] developed a procedure to identify an appropriate exponent λ to transform data into a "normal shape". **Table 2** presents the Box-Cox power transformation and its inverse transform formula. In this table, x_i denotes the cumulative number of software faults detected at $i\left(1,2,\cdots,n\right)$-th testing day. Then, we have the transformed data $\tilde{x}_i$ by means of the Box-Cox power transformation. The transformation parameter λ indicates

Table 2. Box-Cox power transformation formulae.

Box-Cox	Formulae	
	Data transform	Inversion transform
$\lambda = 0$	$\tilde{x}_i = \log(x_i)$	$\exp(\tilde{x}_{n+l})$
$\lambda \neq 0$	$\tilde{x}_i = \frac{x_i^{\lambda} - 1}{\lambda}$	$(\lambda \tilde{x}_{n+l} + 1)^{1/\lambda}$

the power to which all data should be raised, where the parameter λ has to be adjusted in the Box-Cox power transformation. It is common to determine the optimal λ in the pre-experiments before time series prediction.

Let $\tilde{x}_i (i = 1, 2, \cdots, n)$ and $\tilde{x}_{n+l} (l = 1, 2, \cdots)$ be the input and output for the MIMO type of MLP, respectively. Then, the prediction of the cumulative number of software faults is given by the inversion of the data transform. **Figure 1** depicts the architecture of back propagation type MIMO, where n is the number of software fault count data experienced before the observation point t_n and l is the prediction length. We suppose that there is only one hidden layer with $k(=1, 2, \cdots)$ hidden neurons in our MIMO type of MLP.

3.2. Training Phase

Suppose that all the connection weights (nk weights from input to hidden layer, kl weights from hidden to output layer in **Figure 1**) are first given by the uniformly distributed pseudo random varieties. In the MIMO, if these weights are completely known, then it is possible to calculate $(\tilde{x}_{n+1}, \cdots, \tilde{x}_{n+l})$ from the input $(\tilde{x}_1, \cdots, \tilde{x}_n)$ directly. However, since it is impossible to train all the weights including $k(n+l)$ unknown patterns in principle via the common BP algorithm, it is needed to develop a new long-term prediction scheme for the MIMO. In short, we briefly introduce the long-term prediction scheme developed by Begum and Dohi [31]. Suppose that $n > l$ without any loss of generality. In **Figure 2**, we illustrate the configuration of our prediction scheme. In order to predict the cumulative number of software faults for l testing days from the observation point t_n, the prediction has to be made at the point t_{n-l}. This implies that only $(n-l)k + kl = nl$ weights can be estimated with the training data experienced for the period $(t_{n-l}, t_n]$ and that the remaining $k(n+l) - nl$ weights are not trained at time t_n. We call these $k(n+l) - nl$ weights the *non-estimable* weights in this paper. As the prediction length is longer, the number of non-estimable weights becomes greater and the prediction uncertainty also increases more. In this scheme, the Box-Cox transformed data $(\tilde{x}_1, \cdots, \tilde{x}_{n-l})$ with given λ are used for the input in the MIMO, and the remaining data $(\tilde{x}_{n-l+1}, \cdots, \tilde{x}_n)$ are used for the teaching signals in the training phase.

The BP algorithm is the well-known gradient descent method to update the connection weights, so as to minimize the squared error between the network output values and the teaching signals. For the value coming out an input neuron, $\tilde{x}_i (i = 1, 2, \cdots, n-l)$, it is common to add two special inputs; bias units

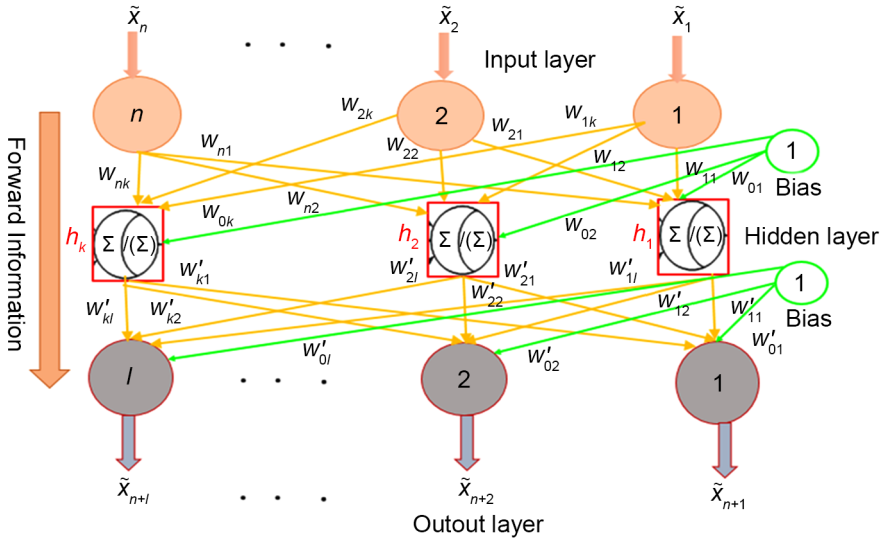

Figure 1. Architecture of back propagation type MIMO.

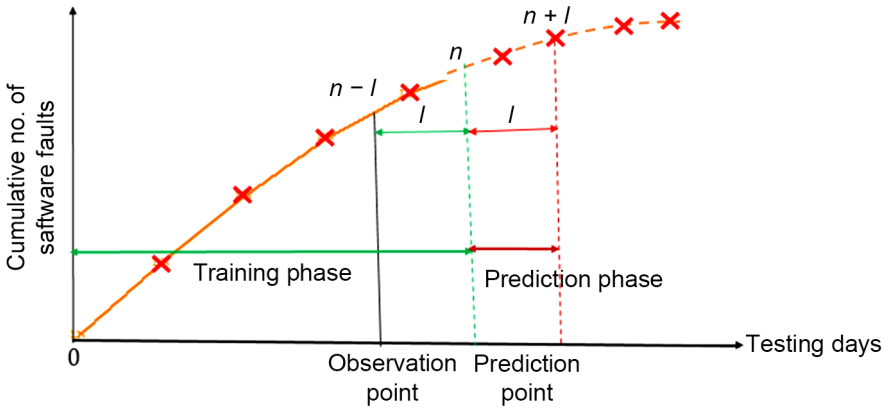

Figure 2. Configuration of prediction scheme via MIMO.

which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. Let $w_{ij} \in [-1,1]$ be the connection weights from i-th input neuron to j-th hidden neuron, where w_{0j} and w'_{0s} denote the bias weights for j-th hidden neuron and s-th output neuron, respectively, for the training phase with $i = 0,1,\cdots,n-l$, $j = 0,1,\cdots,k$ and $s = n-l+1, n-l+2,\cdots,n$. Each hidden neuron calculates the weighted sum of the input neuron, h_j, in the following equation:

$$h_j = \sum_{i=1}^{n-l} \tilde{x}_{ij} w_{ij} + w_{0j}. \tag{6}$$

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

$$\tilde{x}_s = \sum_{j=1}^{k} f(h_j) w'_{js} + w'_{0s}. \tag{7}$$

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

$$\mathrm{SSE}=\frac{\sum_{s=n-l+1}^{n}\left(\tilde{x}_s^o-\tilde{x}_s\right)^2}{(l-1)} \tag{8}$$

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

Next we overview the BP algorithm. It updates the weight parameters so as to minimize SSE between the network output values $\tilde{x}_s\,(s=n-l+1,n-l+2,\cdots,n)$ and the teaching signals $\tilde{x}_s^o$ where each connection weight is adjusted using the gradient descent algorithm according to the contribution to SSE in Equation (8). The momentum, α, and the learning rate, η, are controlled to adjust the weights and the convergence speed in the BP algorithm, respectively. Since these are the most important tuning parameters in the BP algorithm, we carefully examine these parameters in pre-experiments. In this paper we set $\alpha = 0.25 \sim 0.90$ and $\eta = 0.001 \sim 0.500$. Then, the connection weights are updated in the following:

$$w_{ij(\mathrm{new})}=w_{ij}+\alpha w_{ij}+\eta\delta h_j,\,(i=1,2,\cdots,n-l,\,j=1,\cdots,k), \tag{9}$$

$$w'_{js(\mathrm{new})}=w'_{js}+\alpha w'_{js}+\eta\delta\tilde{x}_s,\,(j=1,2,\cdots,k,s=n-l+1,\cdots,n), \tag{10}$$

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

$$\delta h_j=f(h_j)\left(1-f(\delta h_j)\right), \tag{11}$$

$$\delta\tilde{x}_s=\tilde{x}_s(1-\tilde{x}_s)\left(\tilde{x}_s^o-\tilde{x}_s\right), \tag{12}$$

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

$$w_{0j(\mathrm{new})}=w_{0j}+\alpha w_{0j}+\eta\delta h_j, \tag{13}$$

$$w'_{0s(\mathrm{new})}=w'_{0s}+\alpha w'_{0s}+\eta\delta\tilde{x}_s. \tag{14}$$

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

3.3. Prediction Phase

Once the nl weights are estimated with the training data experienced for the period $(t_{n-l},t_n]$ through the BP algorithm, we need to obtain the remaining $k(n+l)-nl$ non-estimable weights for prediction. Unfortunately, since these cannot be trained with the information at time t_n, we need to give these values by the uniform pseudo random variates in the range $[-1,1]$. By giving the random connection weights, the output as the prediction of the cumulative number of software faults, $(\tilde{x}_{n+1},\cdots,\tilde{x}_{n+l})$, are calculated by replacing Equations (6) and

(7) by

$$h_{j(\text{new})} = \sum_{i=1}^{n} \tilde{x}_{ij} w_{ij(\text{new})} + w_{0j(\text{new})}, \tag{15}$$

$$\tilde{x}_{n+s} = \sum_{j=1}^{k} f\left(h_{j(\text{new})}\right) w'_{js(\text{new})} + w'_{0s(\text{new})}, \tag{16}$$

respectively, for $i = 1, 2, \cdots, n$, $j = 1, \cdots, k$ and $s = 1, 2, \cdots, l$. Note that the resulting output is based on one sample by generating a set of uniform pseudo random variates. In order to obtain the prediction of the expected cumulative number of software faults, we generate m sets of random varieties and take the arithmetic mean of the m predictions of $\left(\tilde{x}_{n+1}, \cdots, \tilde{x}_{n+l}\right)$, where m = 1000 is confirmed to be enough in our preliminary experiments. In other words, the prediction in the MIMO type of MLP is reduced to a combination of the BP learning and a Monte Carlo simulation on the connection weights.

4. Numerical Experiments

4.1. Data Sets

We use two real project data sets cited in the reference [2]; DS1 and DS2, which consist of the software fault count (grouped) data. In these data sets, the length of software testing and the total number of detected software faults are given by (62, 133) and (41, 351), respectively. To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent, where the initial guess of weights, w_{ij}, w'_{js}, w_{0j} and w'_{0s}, are given by the uniform random variates ranged in $[-1, +1]$, the number of total iterations in the BP algorithm run is 1000 and the convergence criteria on the minimum error is 0.001 which is same as our previous paper [30]. In **Figure 3** and **Figure 4**, we give two examples on how to determine the optimal transformation parameter λ^*. In our experiments, it is shown that the search range of λ should be $[-3, +2]$.

4.2. Predictive Performance Criterion

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

$$\text{AE}_l = \frac{\sum_{s=1}^{l} \text{RE}_s}{l}, \tag{17}$$

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

$$\text{RE}_s = \left| \frac{\left(\tilde{x}^o_{n+s} - \tilde{x}_{n+s}\right)}{\tilde{x}^o_{n+s}} \right| \left(s = 1, 2, \cdots, l\right). \tag{18}$$

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

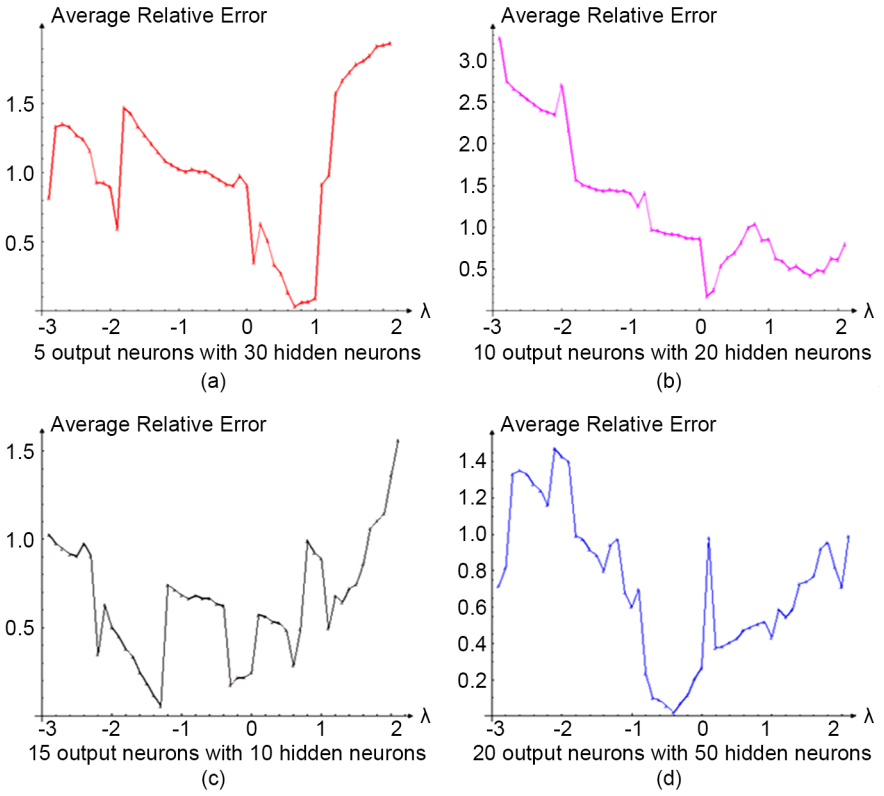

Figure 3. Determination of the transformation parameter λ (DS1 with 50% observation point).

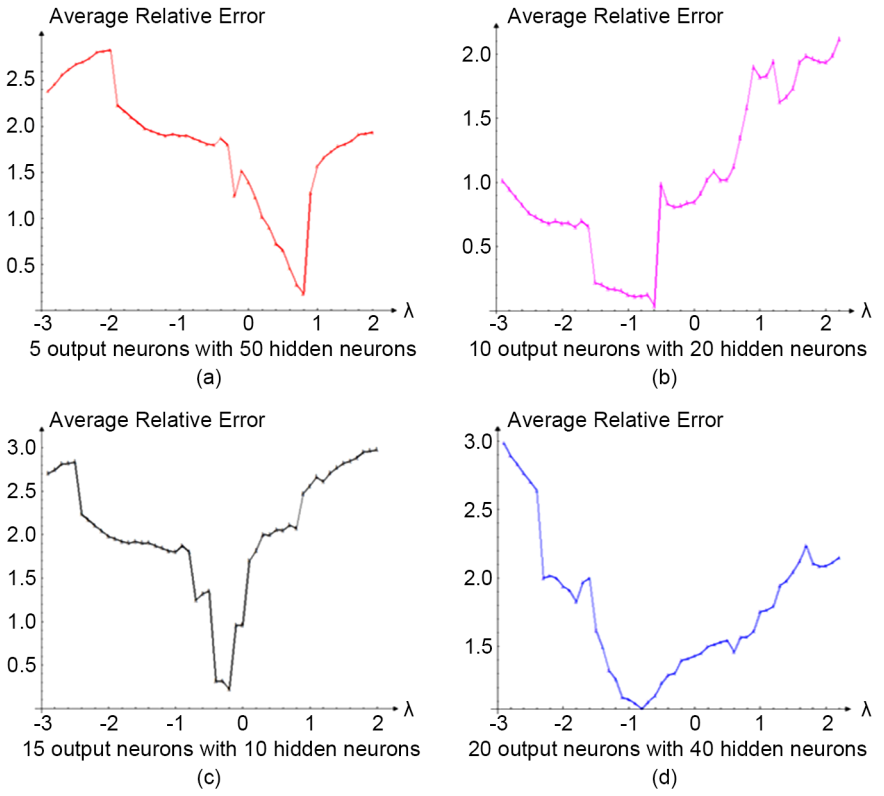

Figure 4. Determination of the transformation parameter λ (DS2 with 50% observation point).

4.3. Results

Tables 3-6 summarize the results on AE for the underlying data set DS1 at 50% ~ 90% observation points of the whole data for the prediction length l = 5, 10, 15 and 20 days, where λ^* denotes the optimal transformation parameter in the sense of minimum AE, and the bold number implies the best prediction model in the same category. For instance, Table 3 gives the prediction results on the cumulative number of software faults for 5 days prediction at respective observation points, when the number of hidden neurons changes from k = 10 to 50. In the MIMO type of MLP neural network, we compare the Box-Cox power transformation with the non-transformed case (Normal) and the best SRGM in Table 1. In the column of SRGM, we denote the best SRGMs in terms of predictive performance (in the sense of minimum AE) and estimation performance (in the sense of minimum AIC) by P and E, respectively.

It is seen that our MIMO-based approaches provide smaller AEs than the common SRGMs in almost all cases when the observation points are 50% and 70%. In the 60% observation point, the best prediction model is the non-transformed MIMO (Normal) with k = 50. On the other hand, in the latter phase of software testing, *i.e.*, 80% ~ 90% observation points, SRGMs, such as txvmax and txvmin, offer less AEs than the MIMO type of MLPs. Even in these cases, it should be noted the best SRGM with the minimum AIC is not always equivalent to the best SRGM with the minimum AE. This fact tells us that one cannot know exactly the best SRGM in advance in terms of predictive performance. Comparing the data transform methods with the non-transformed one, we can find only one case where Normal provides the best prediction result in Table 3 for DS1 and Table 8 for DS2. However, in the other early prediction phases, it is seen that the data transform can work well to give more accurate prediction results in the MIMO type of MLPs.

Tables 7-10 present the prediction results on AE for DS2. Similar to DS1, the MIMO type of MLPs can predict the cumulative number of software faults more accurately in the early testing phase, say, 50% ~ 60% observation points, than SRGMs. Focusing on the number of hidden neurons in the MIMO type of MLPs, we expected first that the larger k may lead to the better predictive performance. However, it is not true from the results in Tables 4-6. In the MIMO-based approach, it is essential to determine feasible k and λ values, because the number of hidden neurons results the expensive computation cost with different prediction length l. In the original paper by Box and Cox [32] they suggest that "fix one, or possibly a small number, of λ's and go ahead with the detailed estimation". In their examples, they use what is usually called "snap to the grid" method to choose the transformation parameter. Unfortunately, no universal method to determine the optimal λ has not been reported yet in the literature. Hence, it is needed to give an appropriate λ even though it is not optimal. From Figure 3 and Figure 4, it can be recognized that the adjustment of λ is quite sensitive to the predictive performance and has to be done through the try-and-error heuristics. However, for an arbitrary λ, we can know that the multi-stages look ahead

Table 3. Comparison of average relative errors for five days prediction with DS1 ($l = 5$).

k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
50% observation ($t_n = 31$)									
10	0.832	1.115	0.970	1.452	0.950	0.826	0.157 (−0.5)	0.224	
20	0.985	1.763	0.989	1.306	0.313	0.384	0.115 (1.8)	0.137	P: pareto (1.280)
30	0.814	0.592	1.006	0.345	0.913	1.938	**0.028 (0.7)**	1.078	E:txvmax (2.286)
40	1.617	1.384	1.491	3.835	0.408	0.787	0.294 (1.9)	0.136	
50	2.154	1.158	0.972	0.488	0.516	1.543	0.435 (1.3)	0.078	
60% observation ($t_n = 37$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	3.188	2.172	0.789	0.931	0.867	1.277	0.052 (0.2)	0.056	
20	2.203	2.203	1.419	1.632	0.929	0.823	0.116 (1.6)	0.038	P:lnorm (0.023)
30	0.876	1.510	4.025	1.210	0.293	1.862	0.072 (1.1)	0.043	E:txvmax (0.764)
40	2.151	0.473	1.331	0.881	0.978	1.523	0.126 (0.8)	0.029	
50	2.185	3.521	1.240	1.529	1.272	0.211	0.099 (−0.3)	**0.018**	
70% observation ($t_n = 43$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	2.234	0.677	0.821	1.540	3.125	1.548	0.245 (0.6)	0.268	
20	1.723	1.113	3.521	3.127	1.415	1.365	0.151 (0.3)	0.172	P:txvmax (0.017)
30	2.262	3.296	1.833	1.928	1.835	1.211	1.105 (1.3)	0.083	E:txvmax (0.017)
40	2.204	4.089	2.405	1.210	0.829	1.541	**0.016 (0.8)**	0.098	
50	1.707	1.331	0.232	0.914	0.637	0.971	0.122 (0.9)	0.056	
80% observation ($t_n = 50$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	1.669	1.245	0.864	0.912	1.183	0.872	0.031 (1.7)	0.017	
20	2.626	0.924	1.436	1.293	0.791	0.379	0.044 (−0.4)	0.059	P:txvmin **(0.007)**
30	1.676	1.472	0.265	2.053	0.529	0.080	0.019 (0.2)	0.027	E:lxvmin (0.022)
40	2.185	1.861	2.547	1.997	1.380	0.509	0.029 (0.9)	0.018	
50	2.785	0.925	3.237	0.590	1.455	2.551	0.119 (0.8)	0.127	
90% observation ($t_n = 56$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	2.652	1.695	0.478	0.717	0.985	1.243	0.145 (0.4)	0.291	
20	1.658	0.424	0.581	0.985	0.068	1.717	0.068 (1.0)	0.043	P:txvmax **(0.020)**
30	2.202	1.751	1.318	1.351	1.598	3.761	0.184 (0.6)	0.186	E:lxvmin (0.030)
40	1.625	1.243	0.558	0.869	1.662	1.120	0.023 (1.7)	0.034	
50	2.915	0.501	0.956	2.127	1.273	0.831	0.134 (1.3)	0.161	

Table 4. Comparison of average relative errors for ten days prediction with DS1 ($I = 10$).

50% observation ($t_n = 31$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	3.253	1.804	1.037	0.508	0.051	0.414	**0.051 (1.0)**	0.958	P:lxvmin (1.120) E:txvmax (3.480)
20	3.172	2.144	1.242	0.166	0.513	0.789	0.166 (0)	0.479	
30	1.123	0.767	0.862	1.241	0.708	0.961	0.341(0.9)	0.841	
40	2.078	0.818	0.762	0.831	1.129	0.639	0.069 (0.6)	0.057	
50	1.201	0.539	0.603	1.811	0.814	0.791	0.479 (1.3)	1.023	
60% observation ($t_n = 37$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	1.273	1.438	1.645	0.098	1.029	0.953	0.043 (0.7)	1.087	P:tlogist **(0.025)** E:txvmax (1.287)
20	1.985	1.383	0.946	1.261	1.698	1.048	0.946 (−1.0)	1.106	
30	2.012	2.924	0.594	1.128	0.537	1.715	0.537 (1.0)	1.746	
40	1.025	1.187	1.137	0.991	1.263	0.961	0.035 (1.6)	0.850	
50	1.876	1.236	0.785	0.864	1.019	1.149	0.623 (1.1)	0.513	
70% observation ($t_n = 43$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	1.193	0.689	2.647	1.043	1.088	0.088	0.088 (2.0)	1.088	P:txvmax **(0.077)** E:txvmax **(0.077)**
20	4.983	0.937	1.723	1.066	0.219	0.613	0.219 (1.0)	0.079	
30	1.133	4.089	3.345	1.549	1.790	3.519	0.265 (1.4)	0.871	
40	1.254	1.237	1.304	1.139	3.129	1.145	0.359 (0.9)	1.058	
50	3.173	1.096	1.129	0.387	2.187	2.153	0.106 (1.2)	1.099	
80% observation ($t_n = 50$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	2.160	0.183	1.058	1.235	1.443	2.198	0.183 (−2.0)	0.382	P:lxvmax **(0.018)** E:lxvmin (0.056)
20	3.121	0.897	0.636	1.074	0.058	0.582	0.058 (1.0)	3.039	
30	1.386	0.541	1.903	0.474	1.031	1.865	0.452 (1.5)	1.052	
40	3.189	1.259	1.126	0.635	0.962	1.813	0.198 (1.9)	0.907	
50	2.175	2.067	1.048	1.269	1.458	1.617	0.339 (1.2)	0.839	

Table 5. Comparison of average relative errors for fifteen days prediction with DS1 ($l = 15$).

k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
50% observation ($t_n = 31$)									
10	1.025	0.448	0.664	0.573	0.491	1.033	**0.056 (−1.4)**	0.348	
20	4.259	1.713	0.531	1.391	1.723	1.632	0.339 (0.3)	1.728	P:lxvmin (1.550) E:txvmax (4.981)
30	3.585	0.296	0.846	0.931	0.635	1.461	0.087 (1.2)	1.298	
40	1.313	1.331	1.234	1.381	1.208	1.220	0.641 (−1.1)	1.461	
50	2.932	1.062	1.432	0.775	1.041	2.140	0.331 (0.2)	1.073	
60% observation ($t_n = 37$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	2.405	1.451	0.041	1.249	1.329	1.049	0.041 (−1.0)	1.028	
20	1.844	2.480	1.682	1.105	0.279	0.347	0.279 (1.0)	1.047	P:tlogist **(0.042)** E:txvmax (0.049)
30	3.696	2.265	0.906	0.936	1.141	0.639	0.198 (0.7)	2.076	
40	1.904	1.289	1.113	2.076	1.985	1.481	0.792 (0.6)	1.037	
50	2.326	1.076	1.234	1.176	1.028	1.149	0.427 (0.5)	0.076	
70% observation ($t_n = 43$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	2.394	1.439	0.864	0.955	1.194	1.402	0.524 (0.6)	2.394	
20	2.175	1.226	2.367	1.061	0.187	0.255	0.187 (1.0)	0.087	P:txvmax **(0.028)** E:txvmax **(0.028)**
30	2.321	1.247	1.369	0.653	0.519	1.527	0.151 (−1.8)	0.519	
40	2.061	1.058	1.152	1.516	1.658	0.932	0.719 (0.8)	1.129	
50	0.987	1.256	1.321	0.293	1.183	0.681	0.137 (0.7)	0.072	

Table 6. Comparison of average relative errors for twenty days prediction with DS1 ($l = 20$).

k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
50% observation ($t_n = 31$)									
10	1.218	1.105	0.692	1.962	0.677	1.450	**0.018 (−0.6)**	1.240	
20	0.910	1.031	1.629	0.169	0.683	0.573	0.169 (0)	0.390	P:lxvmin (1.088) E:txvmax (6.103)
30	0.599	1.611	1.309	0.825	0.483	2.062	0.421 (0.6)	0.634	
40	1.155	1.004	1.194	1.016	0.924	1.407	0.293 (0.2)	3.226	
50	0.712	1.389	0.631	0.980	1.906	0.952	0.591 (−1.1)	0.451	
60% observation ($t_n = 37$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	1.345	0.071	0.384	0.744	1.632	1.722	0.071 (−2.0)	0.524	
20	0.987	0.184	0.519	0.663	1.510	1.523	0.241 (1.3)	2.072	P:tlogist **(0.033)** E:txvmax (0.042)
30	1.277	1.078	0.240	0.864	0.331	0.819	0.034 (−0.7)	0.630	
40	1.080	1.812	1.199	1.573	1.183	0.720	0.641 (0.8)	2.041	
50	1.397	1.748	1.094	1.613	0.265	0.056	0.056 (2.0)	0.379	

Table 7. Comparison of average relative errors for five days prediction with DS2 ($l = 5$).

k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
50% observation ($t_n = 21$)									
10	1.920	1.289	2.398	0.876	0.955	0.902	0.055 (0.4)	0.073	
20	1.395	0.572	1.072	1.079	1.892	1.005	**0.046 (1.3)**	1.207	P:lxvmax (0.211) E: lxvmin (2.011)
30	1.390	1.841	1.572	1.229	0.902	1.981	0.555 (1.1)	1.390	
40	1.908	1.978	1.870	1.105	1.171	0.927	0.279 (1.7)	0.253	
50	2.387	2.167	1.870	1.012	1.198	0.538	0.180 (−0.9)	0.165	
60% observation ($t_n = 25$)									
10	2.562	1.694	1.751	1.598	0.472	0.657	0.472 (1.0)	1.174	
20	1.997	1.817	1.684	0.058	0.289	0.327	0.058 (0)	0.152	P:tlogist (0.039) E:lxvmin (0.075)
30	2.106	2.463	1.617	0.474	0.657	0.829	0.657 (1.0)	0.173	
40	1.901	2.658	1.312	0.489	0.463	0.401	**0.031 (1.9)**	0.332	
50	2.678	1.348	2.470	2.559	0.921	0.966	0.374 (0.4)	1.015	
70% observation ($t_n = 29$)									
10	1.895	0.745	0.926	1.506	1.006	0.828	0.476 (0.6)	1.159	
20	2.116	1.379	1.981	0.380	0.475	0.491	0.380 (0)	0.466	P:lxvmin **(0.021)** E:lxvmin **(0.021)**
30	2.153	2.829	2.147	1.927	1.246	1.061	0.899 (−0.7)	0.395	
40	0.976	1.537	2.340	0.853	1.554	1.556	0.206 (0.9)	1.055	
50	1.389	1.478	1.421	1.246	1.829	1.391	0.331 (0.7)	0.786	
80% observation ($t_n = 33$)									
10	2.463	2.816	0.841	0.396	0.817	0.719	0.394 (0.5)	1.819	
20	1.713	0.673	1.817	1.452	0.541	0.312	0.312 (2.0)	0.988	P:lxvmin **(0.026)** E:lxvmin **(0.026)**
30	2.876	1.922	0.592	1.392	0.339	1.393	0.339 (1.0)	1.195	
40	1.993	1.037	1.325	0.471	0.723	0.835	0.471 (0)	1.023	
50	2.106	1.317	0.927	1.612	2.743	0.916	0.107 (−0.3)	0.885	
90% observation ($t_n = 56$)									
10	2.874	4.027	1.029	0.984	1.489	0.647	0.876 (1.3)	1.152	
20	1.312	1.198	1.217	1.716	0.916	0.186	0.149 (0.6)	1.182	P:txvmax **(0.002)** E:lxvmin (0.039)
30	3.016	1.912	0.949	0.967	0.427	0.416	0.297 (1.5)	1.050	
40	2.956	2.847	1.813	0.229	0.647	0.017	0.017 (2.0)	1.133	
50	2.053	1.113	1.972	1.284	0.263	0.229	0.063 (1.8)	0.603	

Table 8. Comparison of average relative errors for ten days prediction with DS2 ($I = 10$).

k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
50% observation ($t_n = 21$)									
10	3.680	2.078	1.398	0.978	1.701	1.098	0.067 (1.3)	0.541	
20	1.011	0.650	1.123	0.844	1.826	2.118	0.139 (−0.7)	0.216	P:txvmin (0.059) E:lxvmin (2.613)
30	2.203	0.984	0.947	2.497	1.791	0.791	0.791 (2.0)	**0.035**	
40	2.742	1.880	2.118	0.757	1.526	2.572	0.114 (1.5)	0.058	
50	3.641	2.393	0.852	0.821	0.383	0.941	0.383 (1.0)	0.065	
60% observation ($t_n = 25$)									
10	3.325	2.126	1.985	1.876	1.323	0.814	0.814 (2.0)	0.189	
20	2.053	2.971	0.878	0.755	0.943	0.928	0.755 (0)	0.359	P:llogist (0.102) E:lxvmin (0.134)
30	1.917	2.048	1.278	2.293	0.313	0.927	0.313 (1.0)	0.273	
40	1.993	1.901	2.861	0.983	1.386	1.147	0.983 (0)	0.704	
50	2.882	2.106	1.278	1.518	1.278	0.083	**0.083 (2.0)**	0.136	
70% observation ($t_n = 29$)									
10	2.559	2.072	1.517	1.656	1.773	0.613	0.585 (1.8)	1.051	
20	1.528	0.991	2.658	2.158	0.289	1.089	0.289 (1.0)	0.367	P:tnorm **(0.015)**
30	1.818	2.522	1.905	1.898	1.893	1.713	0.385 (0.8)	0.551	P:tnorm
40	2.615	1.528	2.703	0.845	2.819	1.028	0.291 (1.1)	0.341	**(0.015)**
50	1.333	1.929	2.974	2.114	0.950	1.616	0.173 (0.6)	0.284	

Table 9. Comparison of average relative errors for fifteen days prediction with DS2 ($I = 15$).

k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
50% observation ($t_n = 21$)									
10	2.702	1.949	1.870	1.69	2.659	2.974	**0.219 (−0.1)**	2.556	
20	2.971	1.844	1.378	2.166	1.486	0.476	0.476 (2.0)	0.756	P:tnorm (0.227) E:lxvmin (2.865)
30	1.912	1.416	1.999	2.848	0.116	1.213	0.116 (1.0)	1.569	
40	1.389	2.791	2.016	1.375	1.313	1.481	0.336 (1.7)	0.238	
50	1.121	1.017	2.085	1.316	1.126	1.391	0.215 (1.4)	0.533	
60% observation ($t_n = 25$)									
10	1.017	2.143	0.913	0.852	0.159	1.109	0.159 (1)	3.162	
20	2.345	1.831	1.041	2.127	1.563	1.515	0.705 (1.6)	1.158	P:llogist (0.161) E:lxvmin (0.184)
30	1.568	2.819	1.825	2.326	1.117	1.261	0.211 (1.3)	2.190	
40	3.839	1.792	2.834	1.019	1.179	0.839	0.839 (2.0)	1.097	
50	2.413	2.087	1.124	1.236	1.946	1.619	**0.039 (1.1)**	0.385	

Table 10. Comparison of average relative errors for twenty days prediction with DS2 ($l = 20$).

50% observation ($t_n = 21$)									
k	MIMO								SRGM Best Model
	−3.0	−2.0	−1.0	0	1.0	2.0	λ^*	Normal	
10	1.792	1.991	1.498	1.403	0.473	0.513	0.473 (1.0)	1.016	
20	3.107	1.984	1.443	1.101	1.458	0.672	0.672 (2.0)	1.023	P:lxvmax (0.310) E:lxvmin (2.960)
30	2.044	2.715	1.007	1.938	1.406	1.236	0.745 (0.5)	1.302	
40	2.983	1.907	2.043	1.513	1.764	1.149	1.036 (−0.9)	1.251	
50	1.155	2.085	1.713	0.818	1.376	0.305	**0.305 (2.0)**	0.392	

prediction of software fault count is possible with the MIMO type of MLP and that the Box-Cox power transformation improves the predictive accuracy, especially, in the early prediction phase.

5. Concluding Remarks

In this paper we have investigated an applicability of the Box-Cox power transformation to the neuro-based software fault prediction. The ANN employed in this paper is an MIMO type of MLP, and can handle the grouped data on software fault counts as well as make the long-term prediction. To our best knowledge, this paper is the primary challenge to treat the long-term prediction of software faults with the grouped data in the ANN approach. Throughout a comprehensive comparison with the existing SRGMs, it has been shown that our MIMO type of MLP could work well to predict the cumulative number of software faults in the early testing phase. In the future, we will apply the proposed neural network approach to the other software fault count data and conduct more comprehensive data analysis to validate our method with data transformation. Especially, a challenging issue is to develop the prediction interval of the cumulative number of software faults. Even if SRGMs are assumed, it is almost impossible to get the exact predictive intervals of the cumulative number of software faults without applying any approximation method. We will extend our prediction scheme based on the MIMO type of MLP to the interval prediction problem. We will also consider how to select the optimal transformation parameter in the Box-Cox transformation.

Acknowledgements

The first author (M.B.) was partially supported by the MEXT (Ministry of Education, Culture, Sports, Science, and Technology) Japan Government Scholarship.

References

[1] Cai, K.Y. (1998) Software Defect and Operational Profile Modeling. Kluwer Academic Publishers, Boston. https://doi.org/10.1007/978-1-4615-5593-3

[2] Lyu, M.R. (1996) Handbook of Software Reliability Engineering. McGraw-Hill, New

York.

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

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

[5] Xie, M. (1991) Software Reliability Modelling. World Scientific, Singapore. https://doi.org/10.1142/1390

[6] Sharma, K., Garg, R., Nagpal, C.K. and Garg, R.K. (2010) Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach. *IEEE Transactions on Reliability*, **59**, 266-276. https://doi.org/10.1109/TR.2010.2048657

[7] Karunanithi, N., Malaiya, Y.K. and Whitley, D. (1991) Prediction of Software Reliability Using Neural Networks. *Proceedings of the* 2*nd International Symposium on Software Reliability Engineering*, Austin, 17-18 May 1991, 124-130.

[8] Karunanithi, N. and Malaiya, Y.K. (1992) The Scaling Problem in Neural Networks for Software Reliability Prediction. *Proceedings of the* 3*rd International Symposium of Software Reliability Engineering*, Research Triangle Park, 7-10 October 1992, 76-82. https://doi.org/10.1109/issre.1992.285856

[9] Karunanithi, N. and Malaiya, Y.K. (1996) Neural Networks for Software Reliability Engineering. In: Lyu, M.R., Ed., *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 699-728.

[10] Blum, E.K. and Li, L.K. (1991) Approximation Theory and Feed forward Networks. *Neural Networks*, **4**, 511-515.

[11] Khoshgoftaar, T.M., Lanning, D.L. and Pandya, A.S. (1993) A Neural Network Modeling for Detection of High-Risk Program. *Proceedings of the* 4*th International Symposium on Software Reliability Engineering*, Denver, 3-6 November 1993, 302-309.

[12] Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P. and Aud, S.J. (1997) Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunication System. *IEEE Transactions on Neural Networks*, **8**, 902-909. https://doi.org/10.1109/72.595888

[13] Khoshgoftaa, T.M. and Szabo, R.M. (1994) Predicting Software Quality during Testing Using Neural Network Models: A Comparative Study. *International Journal of Reliability, Quality and Safety Engineering*, **1**, 303-319. https://doi.org/10.1142/S0218539394000222

[14] Vashisht, V., Lal, M. and Sureshchandar, G.S. (2015) A Framework for Software Defect Prediction Using Neural Networks. *Journal of Software Engineering and Applications*, **8**, 384-394. https://doi.org/10.4236/jsea.2015.88038

[15] Abdel-Ghaly, A.A., Chan, P.Y. and Littlewood, B. (1986) Evaluation of Competing Software Reliability Predictions. *IEEE Transactions on Software Engineering*, **12**, 950-967. https://doi.org/10.1109/TSE.1986.6313050

[16] Achcar, J.A., Dey, D.K. and Niverthi, M. (1998) A Bayesian Approach Using Non-homogeneous Poisson Processes for Software Reliability Models. In: Basu, A.P., Basu, K.S. and Mukhopadhyay, S., Eds., *Frontiers in Reliability*, World Scientific, Singapore, 1-8.

[17] Goel, A.L. and Okumoto, K. (1979) Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. *IEEE Transactions on Reliability*, **28**, 206-211. https://doi.org/10.1109/TR.1979.5220566

[18] Goel, A.L. (1985) Software Reliability Models: Assumptions, Limitations and Applicability. *IEEE Transactions on Software Engineering*, **11**, 1411-1423. https://doi.org/10.1109/TSE.1985.232177

[19] Gokhale, S.S. and Trivedi, K.S. (1998) Log-Logistic Software Reliability Growth Model. *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*, Washington DC, 13-14 November 1998, 34-41. https://doi.org/10.1109/hase.1998.731593

[20] Littlewood, B. (1984) Rationale for a Modified Duane Model. *IEEE Transactions on Reliability*, **33**, 157-159. https://doi.org/10.1109/TR.1984.5221762

[21] Ohba, M. (1984) Infection S-Shaped Software Reliability Growth Model. In: Osaki, S. and Hatoyama, Y., Eds., *Stochastic Models in Reliability Theory*, Springer-Verlag, Heidelberg, 144-165. https://doi.org/10.1007/978-3-642-45587-2_10

[22] Ohishi, K., Okamura, H. and Dohi, T. (2009) Gompertz Software Reliability Model: Estimation Algorithm and Empirical Validation. *Journal of Systems and Software*, **82**, 535-543.

[23] Okamura, H., Dohi, T. and Osaki, S. (2013) Software Reliability Growth Models with Normal Failure Time Distributions. *Reliability Engineering & System Safety*, **116**, 135-141.

[24] Yamada, S., Ohba, M. and Osaki, S. (1983) S-Shaped Reliability Growth Modeling for Software Error Detection. *IEEE Transactions on Reliability*, **32**, 475-478. https://doi.org/10.1109/TR.1983.5221735

[25] Zhao, M. and Xie, M. (1996) On Maximum Likelihood Estimation for a General Non-Homogeneous Poisson Process. *Scandinavian Journal of Statistics*, **23**, 597-607.

[26] Yang, B., Li, X., Xie, M. and Tan, F. (2010) A Generic Data-Driven Software Reliability Model with Model Mining Technique. *Reliability Engineering & System Safety*, **95**, 671-678.

[27] Xiao, X. and Dohi, T. (2013) Wavelet Shrinkage Estimation for NHPP-Based Software Reliability Models. *IEEE Transactions on Reliability*, **62**, 211-225. https://doi.org/10.1109/TR.2013.2240897

[28] Cheng, H., Tan, P.-N., Gao, J. and Scripps, J. (2006) Multistep-Ahead Time Series Prediction. In: Ng, W.K., Kitsurewa, M. and Li, J., Eds., *Advances in Knowledge Discovery and Data Mining*, LNAI 3918, Springer-Verlag, New York, 765-774.

[29] Park, J., Lee, N. and Baik, J. (2014) On the Long-Term Predictive Capability of Data-Driven Software Reliability Model: An Empirical Evaluation. *Proceedings of the 25th International Symposium on Software Reliability Engineering*, Naples, 3-6 November 2014, 45-54.

[30] Begum, M. and Dohi, T. (2016) Prediction Interval of Cumulative Number of Software Faults Using Multi-Layer Perceptron. In: Lee, R., Ed., *Applied Computing & Information Technology*, Studies in Computational Intelligence, Vol. 619, Springer, Berlin, 43-58.

[31] Begum, M. and Dohi, T. (2016) A Refined Neural Network Approach for Software Fault Prediction with Grouped Data, under Submission.

[32] Box, G.E.P. and Cox, D.R. (1964) An Analysis of Transformations. *Journal of the Royal Statistical Society, Series B*, **26**, 211-252.

[33] Tukey, J.W. (1957) On the Comparative Anatomy of Transformations. *The Annals of Mathematical Statistics*, **28**, 602-632. https://doi.org/10.1214/aoms/1177706875

[34] Dashora, I., Singal, S.K. and Srivastav, D.K. (2015) Software Application for Data Driven Prediction Models for Intermittent Stream Flow for Narmada River Basin. *International Journal of Computer Applications*, **113**, 9-17.

[35] Sennaroglu, B. and Senvar, O. (2015) Performance Comparison of Box-Cox Transformation and Weighted Variance Methods with Weibull Distribution. *Journal of*

Aeronautics and Space Technologies, **8**, 49-55.

[36] Hu, Q.P., Xie, M., Ng, S.H. and Levitin, G. (2007) Robust Recurrent Neural Network Modeling for Software Fault Detection and Correction Prediction. *Reliability Engineering & System Safety*, **92**, 332-340.

[37] Mahajana, R., Guptab, S.K. and Bedib, R.K. (2015) Design of Software Fault Prediction Model Using BR Technique. *Procardia Computer Science*, **46**, 849-858.

[38] Noekhah, S., Hozhabri, A.A. and Rizi, H.S. (2013) Software Reliability Prediction Model Based on ICA Algorithm and MLP Neural Network. *Proceedings of the 7th International Conference on e-Commerce in Developing Countries*, Kish Island, 17-18 April 2013, 1-15. https://doi.org/10.1109/ecdc.2013.6556733

[39] Tian, L. and Noore, A. (2005) Evolutionary Neural Network Modeling for Software Cumulative Failure Time Prediction. *Reliability Engineering & System Safety*, **87**, 45-51.

[40] Tian, L. and Noore, A. (2005) On-Line Prediction of Software Reliability Using an Evolutionary Connectionist Model. *Journal of Systems and Software*, **77**, 173-180.

[41] Okamura, H. and Dohi, T. (2013) SRATS: Software Reliability Assessment Tool on Spreadsheet. *Proceedings of the 24th International Symposium on Software Reliability Engineering*, Pasadena, 4-7 November 2013, 100-117.

[42] Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M. and Torner, F. (2013) Evaluating Long-Term Predictive Power of Standard Reliability Growth Models on Automotive Systems. *Proceedings of the 24th International Symposium on Software Reliability Engineering*, Pasadena, 4-7 November 2013, 228-237.

[43] Kaneishi, T. and Dohi, T. (2013) Software Reliability Modeling and Evaluation under Incomplete Knowledge on Fault Distribution. *Proceedings of the 7th IEEE International Conference on Software Security and Reliability*, Gaithersburg, 18-20 June 2013, 3-12. https://doi.org/10.1109/sere.2013.28

[44] Saito, Y. and Dohi, T. (2015) Software Reliability Assessment via Non-Parametric Maximum Likelihood Estimation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **98**, 2042-2050.