International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Optimal Release Time Estimation of Software System using Box-Cox Transformation and Neural Network

Momotaz Begum, Tadashi Dohi
Department of Information Engineering
Hiroshima University
1-4-1 Kagamiyama, Higashi-Hiroshima 739–8527, Japan
*Corresponding author: momotaz.2k3@gmail.com

(Received June 27, 2017; Accepted September 27, 2017)

Abstract
The determination of the software release time for a new software product is the most critical issue for designing and controlling software development processes. This paper presents an innovative technique to predict the optimal software release time using a neural network. In our approach, a three-layer perceptron neural network with multiple outputs is used, where the underlying software fault count data are transformed into the Gaussian data by means of the well-known Box-Cox power transformation. Then the prediction of the optimal software release time, which minimizes the expected software cost, is carried out using the neural network. Numerical examples with four actual software fault count data sets are presented, where we compare our approach with conventional Non-Homogeneous Poisson Process (NHPP) -based Software Reliability Growth Models (SRGMs).

Keywords- Software cost model, Optimal software release time, Software reliability, Artificial neural network, Data transformation, Long-term prediction, Fault count data, Empirical validation.

1. Introduction

Software testing is an important feature in software development processes and plays an imperative role in defining the quality of software. Moreover, due to huge competitions in the market, software can be seldom survived during the long lifetime due to the competitions and always faces the version up. At the same time, newly added feature in software increases the complexity, which leads to the software faults if not evaluated appropriately. On the other hand, the main concern in software process management by practitioners is to determine when to stop software testing and release the software to the market or users. The determination of the optimal timing to stop software testing is called the optimal software release problem, which is directly related to software testing costs. The total testing cost is reduced successfully if the volume of software test work is less. The debugging cost is more significant in the operational phase than that in the testing phase after releasing software. On the contrary, the higher the software reliability desired, the longer testing period, which further accumulates higher testing cost. Hence, it is important to find as appropriate software release time taking account of the expected software cost. Many authors have discussed in the past on when to stop software testing and to release it for use. The recent trend shows that the quantitative software reliability assessment processes and the efficient software testing methodologies are becoming a firm prerequisite for software developers. Towards achieving the prerequisite, a great number of SRGMs have been developed in the literature (see Goel and Okumoto,1979; Yamada et al., 1983; Ohba, 1984; Littlewood, 1984; Goel, 1985; Abdel-Ghaly et al., 1986; Lyu, 1996; Achcar et al., 1998; Cai, 1998; Gokhale and Trivedi, 1998; Pham, 2000; Ohishi et al., 2009; Okamura et al., 2013).

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Many researchers have also studied the optimal software release policies as well. Okumoto and Goel (1980) derived the optimal software released time for their NHPP-based software reliability model (SRGM) such that the software reliability attains a certain requirement level. Pham and Zhang (1999) proposed a software cost model with warranty and risk costs. The influential involvement of Leung (1992), Dalal and McIntosh (1994), Zhang and Pham (1998), Pham and Zhang (1999), Pham (2003) has been seen in a number of optimization problems under different modeling assumptions. They focused on the completely known stochastic point process such as NHPP-based SRGM to estimate the optimal software testing time. However, it is obvious that there is a need for a better SRGM to fit every software fault count data, addressing the fact that the formulation of the optimal software release problems strongly depends on the accurate statistical estimation of software fault count process. Kaneishi and Dohi (2013), Xiao and Dohi (2013), and Saito and Dohi (2015) introduced new non-parametric assessment methods for software fault count process in testing. However, the disadvantages of these methods rises on the failure to predict the long-term behavior of software fault count process; therefore, these methods are not applicable to the actual software release problems. Recently, a non-parametric method was employed to predict approximately the optimal software release time by minimizing the upper or lower bound of the expected software cost by Saito et al. (2016).

In this paper, a neural network approach with fault count data is used to estimate the optimal software release time, which reduces the relevant software cost. The well-known data transformation technique called the Box-Cox power transform (see Box and Cox, 1964), is applied to transform the data with an arbitrary probability law into the Gaussian data, and to transform the underlying software fault prediction problem into a nonlinear Gaussian regression problem with the Multilayer Perceptron (MLP). An MLP is a feed forward artificial neural network model where all of the input data are mapped onto a set of appropriate outputs. The MLP exploits a supervised learning technique called back propagation. The back propagation is a common learning technique for training Artificial Neural Networks (ANNs) and is used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two-phase cycle; even propagation and weight update. The problems of ANN exist in many applications including software fault prediction, so that there is no efficient way to determine the best neural network architecture. The number of input neurons and output neurons may be determined from physical requirements in many cases. On the other hand, the number of hidden layers and hidden neurons significantly influence the prediction performance in the MLP feed forward neural networks. Nevertheless, the MLP is widely applicable to the statistical identification, function approximation, and time series forecasting.

The idea on data transformation is rather simple, fit useful to handle the non-Gaussian data. Tukey (1957) introduced a family of power transformations where an arbitrary transformation parameter is involved in the power transformations. Box and Cox (1964) proposed a method to calculate 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 contribution by Box and Cox (1964) are two-folds: the first one is to calculate the profile likelihood function and to acquire 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. Dashora et.al. (2015) 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 in the context of neural computation. It compared against the MLP with seasonal autoregressive integrated

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

moving average and Thomas-Fiering models. Sennaroglu and Senvar (2015) estimated 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 on the product specifications. In these ways, the potential applicability of the Box and Cox power transform is acknowledged in several research fields.

Begum and Dohi (2016a) proposed a one-stage look-ahead prediction method with an MLP where the software fault count data are used. The method is adequate to predict the cumulative number of software faults in practice. Dohi et al. (1999) considered an optimal software release problem by means of the MLP-based software fault prediction, and was able to estimate the optimal software release time. They proposed an interesting graphical method to find directly an optimal software testing time, which minimizes the relevant software cost. However, the model dealt with only software fault-detection time data, and failed to make the multi-stage look-ahead prediction. An idea to use Multiple-Input Multiple-Output (MIMO) came from Park et al. (2014) who proposed a refined ANN approach to predict the long-term behavior of a software fault count with the grouped data. Begum and Dohi (2016b, 2016c) imposed a plausible assumption that the underlying fault count process obeys the Poisson law with an unknown mean value function, and proposed to utilize three data transform methods from the Poisson count data to the Gaussian data; Bartlett transform (1936), Anscombe transform (1948) and Fisz transform (1955). They also proposed the idea to apply an MIMO type of MLP to an optimal software release problem with the grouped data. Recently, Begum and Dohi (2017) proposed another MLP-based software fault prediction method with Box-Cox power transformation.

2. NHPP-Based Software Reliability Modeling

Assume that software system test starts at time $t = 0$. Let $X(t)$ denote the cumulative number of software faults detected by time t. Then the counting process $\{X(t), t \geq 0\}$ is said to be a nonhomogeneous Poisson process (NHPP) with intensity function $\emptyset(t;\theta)$, if the following conditions hold:

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

where $o(h)$ is the higher term of infinitesimal time h, and $\emptyset(t;\theta)$ 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 mass function (p.m.f.) of the NHPP is given by

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

where

$$\Lambda(t;\theta) = \int_0^t \emptyset(x;\theta)dx \quad (2)$$

is called the mean value function and indicates the expected cumulative number of software faults

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

up to time t, say, $\Lambda(t;\theta) = E[X(t)]$.

The identification problem of the NHPP is reduced to a statistical estimation problem of the unknown model parameter θ, if the mean value function $\Lambda(t;\theta)$ or the intensity function $\emptyset(t;\theta)$ is known. The parametric statistical estimation methods can then be applied when the parametric form of the mean value function or the intensity function is given. The maximum likelihood method and/or the EM (Expectation-Maximization) algorithm are commonly used to estimate the mean value function. Okamura and Dohi (2013) summarized eleven parametric NHPP-based SRGMs (see Table 1) and developed a parameter estimation tool, SRATS. The best SRGM with smallest AIC (Akaike Information Criterion) is automatically selected in SRATS, where the best SRGM fits the past observation data on software fault counts among the eleven models.

Suppose that n realizations of $X(t_i)$, x_i $(i = 1,2,\dots,n)$, 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 fault count data (t_i, x_i) $(i = 1,2,\dots,n)$ is given by

$$LLF(\theta) = \sum_i^n \big((x_i - x_{i-1}) \log\{\Lambda(t_i;\theta) - \Lambda(t_{i-1};\theta)\} - \log\{(x_i - x_{i-1})!\}\big) - \Lambda(t_n;\theta) \quad (3)$$

where $\Lambda(0;\theta) = 0$, $x_0 = 0$ and $t = t_n$ for simplification. By maximizing Equation (3) with respect to the model parameter θ, we can predict the future value of the intensity function or the mean value function at an arbitrary time t_{n+l} $(l = 1,2,\dots)$, where l denotes the prediction length. In parametric modeling, the prediction at time t_{n+l} is easily done by substituting estimated model parameter $\hat{\theta}$ 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

$$\Lambda(t_{n+l};\ \hat{\theta}) = \int_0^{t_{n+l}} \emptyset(x;\hat{\theta})dx, \quad (4)$$

$$\Lambda(t_{n+l} \big|\ X(t_n) = x_n;\hat{\theta}\big) = x_n + \int_{t_n}^{t_{n+l}} \emptyset(x;\hat{\theta})dx\ = x_n + \Lambda(t_{n+l};\ \hat{\theta}) - \Lambda(t_n;\ \hat{\theta}). \quad (5)$$

If the parametric form of the mean value function or the intensity function is unknown, the identification problem of an NHPP becomes much more difficult. A limited number of nonparametric approaches have been developed by Kaneishi and Dohi (2013), Saito and Dohi (2015), etc. However, those approaches can deal with the fault-detection time data, but will not work for long-term prediction with the grouped fault count data. In addition, the wavelet-based method in Xiao and Dohi (2013) can handle the grouped data, but is unsuccessful to make the long-term prediction. To overcome the above problem we use a fundamental MLP for long-term software fault prediction.

3. MLP Architecture

An Artificial Neuron Network (ANN) is a computational model inspired by the structure and functions of biological neural networks. An ANN has several advantages but one of the most significant ones is that it can be trained from past observation. The simplest ANN has three layers that are interconnected. The first layer consists of input neurons. Those neurons send data to the second layer i.e. hidden layer, which sends the outputs to the third layer. Subsequently, the hidden neurons have no communication with the external world, so that the output layer of neurons sends

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

the final output to the external world. The problem is how to get an appropriate number of hidden neurons in the neural computation. Here, we study an MIMO type of MLP with only one hidden layer. Similar to Section 2, suppose that n software fault count data (t_i, x_i) $(i = 1,2,\dots,n)$ are observed at the observation point t $(= t_n)$. Our concern is about the future prediction of the cumulative number of software faults at time t_{n+l} $(l = 1,2,\dots)$.

3.1 First Phase: Data Transformation

The simplest MLP with only one output neuron is considered as a nonlinear regression model, where the explanatory variables are randomized by the Gaussian white noise. In particular, the output data in the MLP are indirectly assumed to be realizations of a nonlinear Gaussian model. By contrast, the fault count data are integer values. Hence, the original data shall be transformed to the Gaussian data in advance. Xiao and Dohi (2013) used a pre-data processing which is common in the wavelet shrinkage estimation, where the underlying data follows the Poisson data. In the same way, we apply the Box-Cox power transformation technique proposed by Box and Cox (1964), from an arbitrary random data to the Gaussian data. As mentioned in the above, they developed a method to find an appropriate exponent λ to transform data into a "normal shape". Table 2 presents the Box-Cox power transformation and its inverse transform formula where x_i denotes the cumulative number of software faults detected at i $(1,2,\dots,n)$-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 the power to which all data should be raised, where the parameter λ has to be adjusted in the Box-Cox power transformation. Generally, the transformation parameter λ should be optimal that's why before time series prediction pre-experiments is essential. Let $\tilde{x}_i$ $(i = 1,2,\dots,n)$ and $\tilde{x}_{n+l}$ $(l = 1,2,\dots)$ be the input and output for the MIMO type of MLP, respectively. Then, the prediction of the cumulative number of software faults are given by the inversion of the data transform. Fig 1 depicts the architecture of back propagation type MIMO, where n is the number of software fault count data and l is the prediction length. We suppose that there is only one hidden layer with k $(= 1,2,\dots)$ hidden neurons in our MIMO type of MLP.

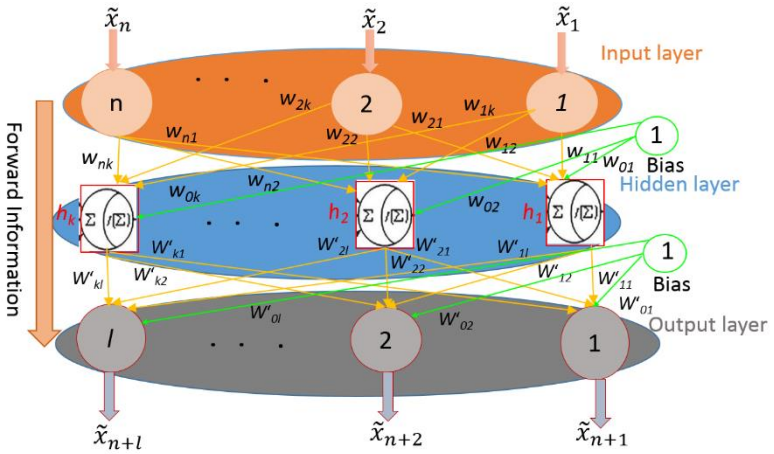

Fig. 1. Architecture of back propagation type MIMO

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

3.2 Second Phase: Training the Neural Network

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

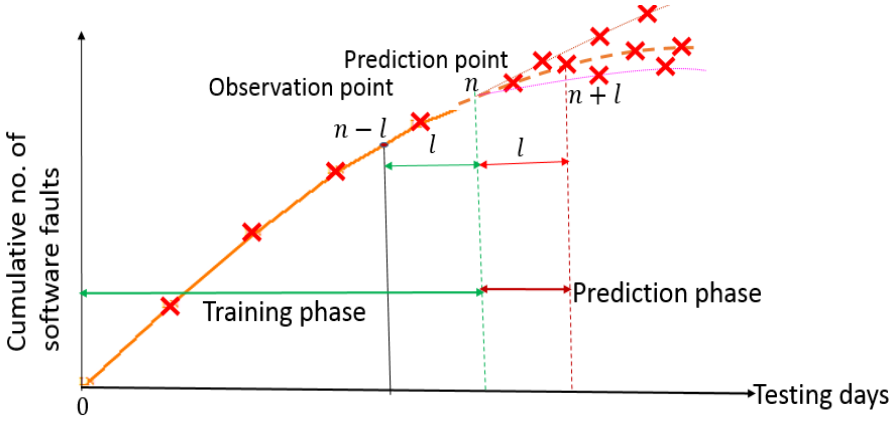

Fig. 2. Configuration of prediction scheme via MIMO

(i) Long-Term Prediction Scheme

In Fig 2, we show the structure of our prediction scheme. For the long-term prediction, we assume that $n > l$ without any loss of generality. In order to predict the cumulative number of software faults for l testing days from the observation point t_n, the prediction has to be made at the point t_{n-l}. This indicates 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. In addition, the prediction improbability also rises more when the prediction length is longer, and the number of non-estimable weights becomes greater. In this scheme, the Box-Cox transformed data $(\tilde{x}_1, \ldots, \tilde{x}_{n-l})$ with given λ are used for the input in the MIMO, and the remaining data $(\tilde{x}_{n-l+1}, \ldots, \tilde{x}_n)$ are used for the teaching signals in the training phase.

(ii) Common BP Algorithm

Back Propagation (BP) is a common method for training a neural network. The BP algorithm is the well-known gradient descent method to update the connection weights, to minimize the squared error between the network output values and the teaching signals. There are two special

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

inputs; bias units which always have the unit values. These inputs are used to evaluate the bias to the hidden neurons and output neurons, respectively. For the value coming out an input neuron, $\tilde{x}_i$ $(i = 1,2,\dots,n-l)$, 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 $i = 0,1,\dots,n-l$, $j = 0,1,\dots,k$ and $s = n-l+1, n-l+2,\dots,n$. 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 $(s = n-l+1, n-l+2,\dots,n)$ 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

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

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 $\tilde{x}_s$ $(s = n-l+1, n-l+2,\dots,n)$ 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 α=0.25~0.90 and η= 0.001~0.500. Then, the connection weights are updated in the following:

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

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

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

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Table 1. NHPP-based SRGMs

Model (Abbr.)	Mean value function
Exponential, (exp) [Goel and Okumoto (1979)]	$\Lambda(t) = aF(t),\ F(t) = a\{1-\exp(-bt)\}$
Gamma, (gamma) [Yamada et.al. (1983)]	$\Lambda(t) = aF(t),\ F(t) = \int_0^t \frac{c^b s^{b-1}\exp(-cs)}{\Gamma(b)} ds$
Pareto, (pareto) [Abdel-Ghaly et.al. (1986), Littlewood (1984)]	$\Lambda(t) = aF(t),\ F(t) = 1-\left(\frac{c}{t+c}\right)^b$
Truncated normal, (tnorm) [Okamura, et.al (2013a)]	$\Lambda(t) = a\frac{F(t)-F(0)}{1-F(0)},\ F(t) = \frac{1}{\sqrt{2\pi b}}\int_{-\infty}^{t}\exp\left(-\frac{(s-c)^2}{2b^2}\right) ds$
Log normal, (lnorm) [Achcar, et.al. (1998), Okamura, et.al (2013a)]	$\Lambda(t) = aF(\log t),\ F(t) = \frac{1}{\sqrt{2\pi b}}\int_{-\infty}^{t}\exp\left(-\frac{(s-c)^2}{2b^2}\right) ds$
Truncated logistic, (tlogist) [Ohba (1984)]	$\Lambda(t) = a\frac{F(t)-F(0)}{1-F(0)}, F(t) = \frac{1}{1+\exp\left(-\frac{t-c}{b}\right)}$
Log logistic, (llogist) [Gokhale and Trivedi (1998)]	$\Lambda(t) = aF(\log t),\ F(t) = \frac{1}{1+\exp\left(-\frac{t-c}{b}\right)}$
Truncated extreme value maximum, (txvmax) [Goel (1985)]	$\Lambda(t) = a\frac{F(t)-F(0)}{1-F(0)},\ F(t) = \exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$
Log extreme value maximum, (lxvmax) [Ohishi, et.al. (2009)]	$\Lambda(t) = aF(\log t),\ F(t) = \exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$
Truncated extreme value minimum, (txvmin) [Ohishi, et.al. (2009)]	$\Lambda(t) = a\frac{F(0)-F(t)}{F(0)},\ F(t) = \exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$
Log extreme value minimum, (lxvmin) [Goel (1985), Ohishi, et.al. (2009)]	$\Lambda(t) = a(1-F(-\log t)),$ $F(t) = \exp\left(-\exp\left\{\left(-\frac{t-c}{b}\right)\right\}\right)$

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

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

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

Table 2. Box-Cox power transform formulae

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

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

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

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

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

3.3 Last Phase: Long-Term Prediction

Once the nl weights are estimated with the training data experienced for the period $(t_{n-l+1,}t_n]$, we need to obtain the remaining $k(n+l)-nl$ non-estimable weights for prediction through the BP algorithm. 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, $(\tilde{x}_{n+1},\dots,\tilde{x}_{n+l})$, are calculated by replacing Equations (6) and (7) by

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

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

respectively, for $i = 1,2,\dots,n, j = 1,\dots,k$ and $s = 1,2,\dots,l$. 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 variaties and take the arithmetic mean of m predictions of $(\tilde{x}_{n+1},\dots,\tilde{x}_{n+l})$, where $m = 1{,}000$ is confirmed to be enough in our preliminary experiments. In other words, the prediction in the MIMO type of MLP is reduced to a combination of the BP learning and a Monte Carlo simulation on the connection weights.

4. Optimal Software Release Decision

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

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

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

$$C(t_0;t_n,\hat{\theta}) = c_0 t_0 + c_1\{x_n + \Lambda(t_0;\hat{\theta}) - \Lambda(t_0|X(t_n) = x_n;\hat{\theta})\} + c_2\{\Lambda(T_L;\hat{\theta}) - \Lambda(t_0;\hat{\theta})\}. \quad (17)$$

Note that $\Lambda(T_L;\hat{\theta})$ is independent of t_0 and that $\Lambda(t_n;\hat{\theta}) = x_n$from Eq.(5). When $t_0 = t_{n+l}$, l corresponds to the remaining testing length. Hence, our optimization problem is essentially reduced to

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

$$\min_{t_n \le t_0 \le T_L} C(t_0; T_L; \hat{\theta}) \Leftrightarrow \min_{t_n \le t_0 \le T_L} \{c_0 t_0 - (c_2 - c_1)\Lambda(t_0; \hat{\theta})\} \tag{18}$$

$$\Leftrightarrow \max_{t_n \le t_0 \le T_L} \left\{ \Lambda(t_0; \hat{\theta}) - \left(\frac{c_0}{c_2 - c_1} \right) t_0 \right\}.$$

When SRGMs are assumed, the mean value function $\Lambda(t_0; \hat{\theta})$ can be estimated from the underlying data. On the other hand, in the context of neural computation, the output of MLP in Eq. (16) is regarded as an estimate of $\Lambda(t_0; \hat{\theta})$. Hence, output sequence $(\tilde{x}_{n+1}, \tilde{x}_{n+2}, \dots, \tilde{x}_{n+l})$ denotes $\left(\Lambda(t_{n+1}; \hat{\theta}), \Lambda(t_{n+2}; \hat{\theta}), \dots, \Lambda(t_{n+l}; \hat{\theta})\right)$ in the MIMO type MLP. Dohi et al. (1999) gave the essentially similar but somewhat different dual problem to Eq. (18). Consequently, the optimal release time $\hat{t}_0$ is equivalent to the point with the maximum vertical distance between the predicted curve $\Lambda(t_0; \hat{\theta})$ and a straight line $(c_2 - c_1)t_0/c_0$ in the two-dimensional plane. Then, it can be graphically checked that there exists a finite optimal software release time. In fact, when $\hat{t}_0$ is equal to the observation point t_n, then it is optimal not to continue testing the software product any more. In other words, the optimization problem considered here is a prediction problem of the function $\Lambda(t_0; \hat{\theta})$.

5. Numerical Experiments

We give numerical examples to predict the optimal software release time based on our neural network approach, where four data sets, DS1∼DS4, are used for analysis (see Lyu (1996)); 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), (41, 351), (46,266) and (109,535) respectively. To find out the desired output via the BP algorithm, we need much computation cost to calculate the gradient descent. 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 $1{,}000$ and the convergence criterion on the minimum error is 0.001 which is same as our previous paper by Begum and Dohi (2016a). In our experiments, it is shown that the search range of the transformation parameter λ should be $[-3, +2]$. We define the error criterion in the following to evaluate the prediction performance of the optimal software release time via MIMO type of MLP or SRGM,

$$\text{Prediction Error} = \left| \frac{\hat{t}_0 - t_0^*}{t_0^*} \right| \times 100, \tag{19}$$

where $\hat{t}_0$ is a posteriori optimal testing time and is calculated by finding the point l $(= 0,1,2, \dots, q)$ which maximizes $\Lambda(t_{n+l}; \hat{\theta}) - (c_2 - c_1)t_{n+l}/c_0$, and q satisfies $t_{n+q} = T_L$. This error criterion is used for the predictive performance when all the data $x_1, x_2, \dots, x_n, \ x_{n+1}, x_{n+2}, \dots, x_{n+q}$ are given. In Table 3, we present posteriori optimal release times for DS1∼DS4, when $c_0 = 4$, $c_1 = 1$ = 1 and c_2 varies in the range of 2∼10.

Meanwhile the cost parameter c_2 is not sensitive to the posteriori optimal release time in Table 3, we focus on only two cases; c_2 = 2 and c_2 = 10 hereafter. In Tables 4 and 5 we give the prediction results on the optimal software release time for DS1 with c_2 = 2 and c_2 = 10, respectively, where the prediction of optimal release time and its associated prediction error are

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

calculated at each observation point (50%~90% points of the whole data). In these tables, the bold number implies the best prediction model by MIMO type of MLP in comparison with the best SRGM among eleven models in Table 1. In the MIMO type of MLP, we compare Box-Cox power transformation results with the non-transformed case (Normal) and the best SRGM. In the column of SRGM, we denote the best SRGMs in terms of prediction performance (in the sense of minimum average relative error; AE) and estimation performance (in the sense of minimum Akaike Information Criterion; AIC), denoted by P and E, respectively, where the capability of the prediction model is measured by the Average Error (AE);

$$AE_l = \frac{\sum_{s=1}^{l} RE_s}{l}, \tag{20}$$

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

$$RE_s = \left|\frac{(\tilde{x}^o_{n+s} - \tilde{x}_{n+s})}{\tilde{x}^o_{n+s}}\right| \quad (s = 1,2,\dots,l). \tag{21}$$

So we regard the prediction model with smaller AE as a better prediction model. In Table 4 and 5, it is seen that our approaches could provide smaller prediction error than the common SRGM in almost all observation points (50%~90%).

Table 3. Posteriori optimal software release time for four datasets

c_2	t_0^*			
	DS1	DS2	DS3	DS4
2	50	28	33	59
3	51	35	37	89
4	51	35	44	89
5	54	36	44	89
6	56	36	44	89
7	62	36	44	89
8	62	36	44	89
9	62	36	44	89
10	62	36	44	89

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Table 4. Prediction results of optimal software release time with DS1 with $c_2 = 2$

50% Observation ($t_n = 31$)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P: pareto, 40 (21.56%) E: txvmax, 35 (31.37%)
37 (27.45%)	42 (17.64%)	49 (3.92%)	35 (31.37%)	42 (17.64%)	40 (21.56%)	**51(λ =0.7) (0%)**	42 (17.46%)	
60% observation ($t_n = 37$)								
38 (25.49%)	40 (21.56%)	41 (19.6%)	46 (9.8%)	38 (25.49%)	47 (7.84%)	**49(λ =1.8) (3.92%)**	39 (23.53%)	P: lnorm, 47 (7.84%) E: txvmax, 38 (25.49%)
70% observation ($t_n = 43$)								
45 (11.76%)	48 (5.88%)	49 (3.92%)	47 (7.84%)	47 (7.84%)	50 (1.96%)	**51(λ =1.5) (0%)**	49 (3.92%)	**P: txvmax, 51 (0%) E: txvmax, 51 (0%)**
80% observation ($t_n = 50$)								
52 (1.96%)	53 (3.92%)	57 (11.76%)	60 (17.64%)	53 (3.92%)	51 (0%)	**51(λ =2.0) (0%)**	52 (1.96%)	P: txvmin, 61 (19.6%) E: lxvmin, 52 (1.96%)
90% observation ($t_n = 56$)								
60 (17.64%)	61 (19.6%)	58 (13.72%)	57 (11.76%)	58 (13.72%)	59 (15.69%)	**56(λ =0.9) (9.81%)**	59 (15.69%)	P: txvmax, 61 (19.6%) E: lxvmin, 61 (19.6%)

Table 5. Prediction results of optimal software release time with DS1 with $c_2 = 10$

50% Observation ($t_n = 31$)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P: pareto, 40 (35.48%) E:txvmax, 5 (43.55%)
33 (46.77%)	35 (43.55%)	32 (51.62%)	38 (38.71%)	46 (25.81%)	41 (56.45%)	**53(λ =1.8) (14.51%)**	39 (37.10%)	
60% observation ($t_n = 37$)								
41 (56.45%)	38 (38.71%)	38 (38.71%)	46 (25.81%)	41 (56.45%)	46 (25.81%)	**54(λ =1.2) (12.90%)**	46 (25.81%)	P: lnorm, 49 (20.96%) E: txvmax, 38 (38.71%)
70% observation ($t_n = 43$)								
49 (20.96%)	50 (19.35%)	49 (20.96%)	49 (20.96%)	50 (19.35%)	54 (12.90%)	**59(λ =1.3) (4.84%)**	49 (20.96%)	P: txvmax, 51 (17.74%) E: txvmax, 51 (17.74%)
80% observation ($t_n = 50$)								
53 (14.51%)	53 (14.51%)	54 (12.90%)	57 (8.06%)	57 (8.06%)	56 (9.67%)	59(λ =0.9) (4.84%)	53 (14.51%)	**P: txvmin, 61 (1.61%)** E: lxvmin, 52 (16.13%)
90% observation ($t_n = 56$)								
58 (6.45%)	57 (8.06%)	59 (4.84%)	59 (4.84%)	60 (1.61%)	57 (8.06%)	60(λ =1.0) (3.22%)	56 (9.67%)	**P:txvmax, 61 (1.61%) E: lxvmin, 61 (1.61%)**

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

In contrast, at 80% observation time the non-transformed case (Normal) showed small errors (see Table 5). Even in these cases, it should be noted that the best SRGM with the minimum AIC is not always equivalent to the best SRGM with the minimum AE. This fact tells that one cannot know exactly the best SRGM in terms of judgment on when to stop software testing.

Tables 6 and 7 give the optimal software release time for c_2 = 2 and c_2 = 10 with DS2. In the latter phase of software testing, *i.e.*, 80%~ 90% observation points, SRGMs, such as txvmin, txvmax and lxvmin, could offer less error than our MIMO type of MLP (see Table 6). Similarly Tables 8~11 provide the optimal software release time for c_2 = 2 and c_2 = 10 with DS3 and DS4 respectively. In most of the cases our MIMO type of MLP could give the best timing of software release for the all testing phase. Throughout our numerical experiments, we can conclude that when the cost parameter c_2 increases, the resulting software release time also increases.

Table 6. Prediction results of optimal software release time with DS2 with $c_2 = 2$

50% observation (t_n = 21)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P: **lxvmax, 26 (7.14%)** E: lxvmin, 22 (21.43%)
22 (21.43%)	23 (17.85%)	22 (21.43%)	22 (21.43%)	24 (14.28%)	26 (7.14%)	**26(λ =2.0) (7.14%)**	24 (14.28%)	
60% observation (t_n = 25)								
31 (10.14%)	31 (10.14%)	30 (7.14%)	28 (0%)	27 (3.57%)	26 (7.14%)	**28(λ =0.0) (0%)**	29 (3.57%)	P: tlogist, 30 (7.14%) E: lxvmin, 35 (25%)
70% observation (t_n = 29)								
32 (14.28%)	33 (17.85%)	31 (10.14%)	31 (10.14%)	30 (7.14%)	35 (25%)	**29(λ =1.5) (3.57%)**	33 (17.85%)	P: lxvmin, 35 (25%) E: lxvmin, 35 (25%)
80% observation (t_n = 34)								
37 (32.14%)	39 (39.28%)	37 (32.14%)	39 (17.85%)	37 (32.14%)	35 (25%)	**35(λ =2.0) (25%)**	**35 (25%)**	**P: lxvmin, 35 (25%) E: lxvmin, 35 (25%)**
90% observation (t_n = 37)								
39 (39.28%)	40 (42.85%)	40 (42.85%)	39 (39.28%)	37 (32.14%)	40 (42.85%)	**37(λ =0.0) (32.14%)**	40 (42.85%)	**P: txvmax, 37 (32.14%)** E: lxvmin, 39 (39.28%)

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Table 7. Prediction results of optimal software release time with DS2 with $c_2 = 10$

50% observation ($t_n = 21$)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P: lxvmax, 26 (27.78%)
25 (30.55%)	23 (36.11%)	25 (30.55%)	25 (30.55%)	22 (38.89%)	25 (30.55%)	**32(λ =1.7) (11.11%)**	25 (30.55%)	E: lxvmin, 22 (38.89%)
60% observation ($t_n = 25$)								
27 (25%)	32 (11.11%)	29 (19.44%)	29 (19.44%)	31 (13.89%)	31 (13.89%)	**35(λ =0.8) (2.78%)**	33 (8.33%)	P: tlogist, 30 (16.67%) E: lxvmin, 27 (25%)
70% observation ($t_n = 29$)								
31 (13.89%)	33 (8.33%)	31 (13.89%)	31 (13.89%)	30 (16.67%)	35 (2.78%)	**35(λ =2.0) (2.78%)**	33 (8.33%)	**P: lxvmin, 35 (2.78%) E: lxvmin, 35 (2.78%)**
80% observation ($t_n = 34$)								
39 (8.33%)	39 (8.33%)	39 (8.33%)	37 (2.78%)	35 (2.78%)	36 (0%)	**36(λ =2.0) (0%)**	39 (8.33%)	P: lxvmin, 35 (2.78%) E: lxvmin, 35 (2.78%)
90% observation ($t_n = 37$)								
39 (8.33%)	38 (5.55%)	37 (2.78%)	41 (13.89%)	39 (8.33%)	38 (5.55%)	**37(λ =-1.0) (2.78%)**	39 (8.33%)	**P: txvmax, 37 (2.78%)** E: lxvmin, 39 (8.33%)

Table 8. Prediction results of optimal software release time with DS3 with $c_2 = 2$

50% observation ($t_n = 23$)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P: lnorm, 39 (18.18%)
36 (9.09%)	38 (15.15%)	37 (10.81%)	39 (18.18%)	39 (18.18%)	42 (27.27%)	**33(λ =-0.3) (0%)**	44 (33.33%)	E: exp, 40 (17.5%)
60% observation ($t_n = 28$)								
38 (15.15%)	39 (18.18%)	42 (27.27%)	38 (15.15%)	41 (24.24%)	38 (15.15%)	**35(λ =1.9) (6.06%)**	42 (27.27%)	P: lnorm, 46 (39.39%) E: pareto, 46 (39.39%)
70% observation ($t_n = 32$)								
44 (33.33%)	39 (18.18%)	41 (24.24%)	35 (6.06%)	34 (3.03%)	41 (24.24%)	**34(λ =-1.0) (3.03%)**	40 (21.21%)	P: gamma, 35 (6.06%) E: txvmax, 39 (18.18%)
80% observation ($t_n = 37$)								
39 (18.18%)	40 (21.21%)	39 (18.18%)	40 (21.21%)	41 (24.24%)	41 (24.24%)	**38(λ =-1.3) (15.15%)**	39 (18.18%)	P:lxvmin, 39 (18.18%) E: exp, 42 (27.27%)
90% observation ($t_n = 41$)								
46 (39.39%)	45 (36.36%)	43 (30.30%)	43 (30.30%)	44 (33.33%)	46 (39.39%)	**41(λ =1.2) (24.24%)**	46 (39.39%)	P:lxvmin 42 (27.27%) E: txvmax, 44 (33.33%)

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Table 9. Prediction results of optimal software release time with DS3 with $c_2 = 10$

50% observation (t_n = 23)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P: lnorm, 44 (0%)
41 (6.82%)	39 (11.36%)	35 (20.45%)	38 (13.64%)	39 (11.36%)	41 (6.82%)	**44(λ=1.7) (0%)**	40 (9.09%)	E:exp, 40 (9.09%)
60% observation (t_n = 28)								
39 (11.36%)	38 (13.64%)	38 (13.64%)	41 (6.82%)	42 (4.54%)	36 (36.36%)	**42(λ=1.0) (4.54%)**	40 (9.09%)	P:lnorm, 46 (4.54%) E: pareto, 46 (4.54%)
70% observation (t_n = 32)								
39 (11.36%)	40 (9.09%)	41 (6.82%)	39 (11.36%)	36 (18.18%)	41 (6.82%)	**44(λ=-2.7) (0%)**	39 (11.36%)	P:gamma, 42 (4.54%) E:txvmax, 39 (11.36%)
80% observation (t_n = 37)								
39 (11.36%)	39 (11.36%)	42 (4.54%)	41 (6.82%)	39 (11.36%)	42 (4.54%)	43(λ=0.9) (2.27%)	40 (9.09%)	**P: lxvmin, 44 (0%)** E: exp, 42 (4.54%)
90% observation (t_n = 41)								
46 (4.54%)	42 (4.54%)	46 (4.54%)	41 (6.82%)	44 (0%)	46 (4.54%)	**44(λ=1.0) (0%)**	45 (2.27%)	**P: lxvmin, 44 (0%)** E:txvmax, 46 (4.54%)

Table 10. Prediction results of optimal software release time with DS4 with $c_2 = 2$

50% observation (t_n = 55)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	P:txvmax, 56 (5.08%)
57 (3.39%)	62 (5.09%)	59 (0%)	65 (10.17%)	62 (5.09%)	60 (1.69%)	**59(λ =-1.0) (0%)**	60 (1.69%)	E:lxvmin, 109 (84.74%)
60% observation (t_n = 64)								
70 (18.64%)	79 (33.90%)	109 (84.74%)	90 (52.54%)	105 (77.79%)	90 (52.54%)	68(λ =1.7) (15.25%)	99 (67.80%)	**P:txvmax, 65 (1.69%)** E:lxvmin, 109 (84.74%)
70% observation (t_n = 76)								
89 (50.85%)	102 (72.88%)	109 (84.74%)	105 (77.79%)	89 (50.85%)	85 (44.07%)	**77(λ =1.3) (30.50%)**	105 (77.79%)	**P:lxvmax,** 77 **(30.50%)** E:lxvmin, 109 (84.74%)
80% observation (t_n = 87)								
105 (77.79%)	102 (72.88%)	109 (84.74%)	105 (77.79%)	88 (49.15%)	89 (50.85%)	**87(λ =0.9) (47.45%)**	105 (77.79%)	P:lxvmax, 88 (49.15%) E:lxvmin, 109 (84.74%)
90% observation (t_n = 98)								
109 (84.74%)	109 (84.74%)	109 (84.74%)	109 (77.79%)	99 (67.80%)	105 (77.79%)	**99(λ =1.0) (67.80%)**	109 (84.74%)	**P:lxvmax, 99 (67.80%)** E:lxvmin, 109 (84.74%)

International Journal of Mathematical, Engineering and Management Sciences
Vol. 3, No. 2, 177–194, 2018
https://dx.doi.org/10.33889/IJMEMS.2018.3.2-014

Table 11. Prediction results of optimal software release time with DS4 with $c_2 = 10$

50% observation ($t_n = 55$)								
MIMO ($\hat{t}_0$)								Best SRGM ($\hat{t}_0$)
-3.0	-2.0	-1.0	0.0	1.0	2.0	Best λ	Normal	
62 (30.33%)	68 (23.59%)	85 (4.49%)	61 (31.46%)	59 (33.71%)	69 (22.47%)	**85(λ =-1.0) (4.49%)**	101 (13.47%)	P:txvmax, 56 (37.07%) E:lxvmin, 109 (22.47%)
60% observation ($t_n = 64$)								
90 (1.12%)	105 (17.97%)	100 (12.36%)	104 (16.85%)	89 (0%)	105 (17.97%)	**89(λ =1.0) (0%)**	105 (17.97%)	P:txvmax, 65 (26.67%) E:lxvmin, 109 (22.47%)
70% observation ($t_n = 76$)								
105 (17.97%)	109 (22.47%)	99 (11.23%)	89 (0%)	109 (22.47%)	100 (12.36%)	**89(λ =0.0) (0%)**	99 (11.23%)	P:lxvmax, 77 (13.48%) E:lxvmin, 109 (22.47%)
80% observation ($t_n = 87$)								
109 (22.47%)	105 (17.97%)	106 (19.10%)	99 (11.23%)	99 (11.23%)	109 (22.47%)	**89(λ =1.7) (0%)**	100 (12.36%)	P:lxvmax, 88 (1.12%) E:lxvmin, 109 (22.47%)
90% observation ($t_n = 98$)								
102 (14.61%)	102 (14.61%)	102 (14.61%)	109 (22.47%)	109 (22.47%)	109 (22.47%)	**98(λ =1.0) (10.11%)**	109 (22.47%)	P:lxvmax, 99 (11.23%) E:lxvmin, 109 (22.47%)

6. Conclusion

Almost all software is complex in nature, which has a huge testing scope. All defects in the software are possible to find out but testing will be endless and testing cycles will continue until a decision is made when to stop. Now it becomes even more complicated to come to a decision to stop testing which is being minimized the relevant expected cost. We have given illustrative examples with four real software fault data to derive estimates of the optimal software release time and performed a sensitivity analysis for each the observation point. In the numerical examples are that our MIMO type of MLP could give the accurate optimal software release time in the all testing phases. Additionally, the method can also help project managers to decide when to stop the software testing for market release at the earlier time. Future work will include to develop a tool to support the optimal software testing decision by automating the BP learning and the size determination of a hidden layer.

Acknowledgement

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

References

Abdel-Ghaly, A. A., Chan, P. Y., & Littlewood, B. (1986). Evaluation of competing software reliability predictions. *IEEE Transactions on Software Engineering*, *SE-12*(9), 950-967.

Achcar, J. A., Dey, D. K., & Niverthi, M. (1998). A Bayesian approach using nonhomogeneous Poisson processes for software reliability models. *Frontiers in Reliability* (A. P. Basu, K. S. Basu and S. Mukhopadhyay, eds.), 1-18, World Scientific.

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

Bartlett, M. (1936). The square root transformation in analysis of variance. *Supplement to the Journal of the Royal Statistical Society*, *3*(1), 68-78.

Begum, M., & Dohi, T. (2016a). Prediction interval of cumulative number of software faults using multilayer perceptron. In *Applied Computing & Information Technology* (pp. 43-58). Springer, New York.

Begum, M., & Dohi, T. (2016b). Optimal software release decision via artificial neural network approach with bug count data, in Jung, K. M., Kimura, K. and Cui, L.-R. (eds.), *Advanced Reliability and maintenance Modeling VII* (*Proceedings of 7th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modeling (APARM 2016)*), McGraw Hill, 17-24.

Begum, M., & Dohi, T. (2016c). Optimal stopping time of software system test via artificial neural network with fault count data. *Journal of Quality in Maintenance Engineering* (accepted).

Begum, M., & Dohi, T. (2017). A neuro-based software fault prediction with Box-Cox power transformation. *Journal of Software Engineering and Applications, 10,* 288-309.

Box, G. E. P., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society, Series B (Methodological)*, *26*(2), 211-252.

Cai, K. Y. (1998). *Software defect and operational profile modeling*. Kluwer Academic Publishers.

Dalal, S. R. & McIntosh, A. A. (1994). When to stop testing for large software systems with changing code. *IEEE Transactions on Software Engineering*, *20*(4), 318-323.

Dashora, I., Singal, S. K., & 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*(10), 9-17.

Dohi, T., Nishio, Y., & Osaki, S. (1999). Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering*, *8*, 167-185.

Fisz, M. (1955). The limiting distribution of a function of two independent random variables and its statistical application. *Colloquium Mathematicum*, *3*, 138-146.

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

Goel, A. L., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, *R-28*(3), 206-211.

Gokhale, S. S., & Trivedi, K. S. (1998). Log-logistic software reliability growth model. *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium (HASE-1998)*, IEEE CPS, 34-41.

Kaneishi. T., & 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 (SERE-2013)*, IEEE CPS, 3-12.

Leung, Y. W. (1992). Optimal software release time with a given cost budget. *Journal of Systems and Software*, *17*(3), 233–242.

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

Lyu, M. R. (1996). *Handbook of software reliability engineering*. McGraw-Hill.

Ohba, M. (1984). Inflection S-shaped software reliability growth model. In *Stochastic models in reliability theory* (pp. 144-162). Springer, Heidelberg.

Ohishi, K., Okamura, H., & Dohi, T. (2009). Gompertz software reliability model: estimation algorithm and empirical validation. *Journal of Systems and Software*, *82*(3), 535-543.

Okamura, H., & Dohi, T. (2013). SRATS: Software reliability assessment tool on spreadsheet. *Proceedings of The 24th International Symposium on Software Reliability Engineering (ISSRE- 2013)*, IEEE CPS, 100-107.

Okamura, H., Dohi, T., & Osaki, S. (2013). Software reliability growth models with normal failure time distributions. *Reliability Engineering* & *System Safety*, *116*, 135-141.

Okumoto, K., & Goel, A. L. (1980). Optimal release time for software systems based on reliability and cost criteria. *Journal of Systems and Software*, *1*, 315-318.

Park, J., Lee, N., & 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 (ISSRE-2014)*, IEEE CPS, 45-54.

Pham, H. (2000). *Software reliability*, Springer-Verlag.

Pham, H. (2003). Software reliability and cost models: perspectives, comparison, and practice. *European Journal of Operational Research*, *149*(3), 475–489.

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

Saito, Y., & Dohi, T. (2015). Software reliability assessment via non-parametric maximum likelihood estimation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences (A)*, *E98-A*(10), 2042–2050.

Saito, Y., Moroga, T., & Dohi, T. (2016). Optimal software release decision based on nonparametric inference approach. *Journal of the Japan Industrial Management Association*, *66*(4E), 396-405.

Sennaroglu, B., & Senvar, O. (2015). Performance comparison of Box-Cox transformation and weighted variance methods with Weibull distribution. *Journal of Aeronautics and Space Technologies*, *8*(2), 49-55.

Tukey, J. W. (1957). On the comparative anatomy of transformations. *The Annals of Mathematical Statistics*, *28*(3), 602–632.

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

Yamada, S., Ohba, M., & Osaki, S. (1983). S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability*, *R-32*(5), 475-478.

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