Baltic J. Modern Computing, Vol. 4 (2016), No. 4, pp. 1016–1037
http://dx.doi.org/10.22364/bjmc.2016.4.4.26

Software Reliability Assessment using Neural Networks of Computational Intelligence Based on Software Failure Data

Manmath Kumar BHUYAN[1], Durga Prasad MOHAPATRA[2], Srinivas SETHI[3]

[1] Utkal University, Vanivihar, IGIT, Sarang, India
[2] National Institute Technology, Rourkela, Odisha, India
[3] Indira Gandhi Institute of Technology, Sarang, Odisha, India

manmathr@gmail.com, durga@nitrkl.ac.in, srinivas_sethi@igitsarang.ac.in

Abstract. The computational intelligence approach using *Neural Network (NN)* has been known to be very useful in predicting software reliability. Software reliability plays a key role in software quality. In order to improve accuracy and consistency of software reliability prediction, we propose the applicability of *Feed Forward Back-Propagation Network* (*FFBPN*) as a model to predict software reliability. The model has been applied on data sets collected across several standard software projects during system testing phase with fault removal. Unlike most connectionist models, our model attempt to compute *average error* (*AE*), the *root mean square error* (*RMSE*), *normalized root mean square error* (*NRMSE*), *mean absolute error* (*MAE*) simultaneously. A comparative study among the proposed feed-forward neural network with some traditional parametric software reliability growth model's performance is carried out. The results indicated in this work suggest that *FFBPN* model exhibit an accurate and consistent behavior in reliability prediction.

Keywords: Computational Intelligence; Neural Networks; Software Reliability; Data set; Predictive Measures; Feed Forward; Back-propagation; Average Error; Root Mean Square Error; Normalized Root Mean Square Error; Mean Absolute Error.

1 Introduction

According to demand of the modern digital age, it is a big challenge for software developers to quickly design, implement, test, and maintain complex software systems. Also it is a difficult task for software companies to deliver good quality software in appropriate time (Bhuyan et al., 2014). The last two decades have witnessed a paradigm shift in the field of software engineering (Benala, 2012).

A typical definition of software reliability is "the probability of the failure free operation of a computer program for a specified exposure period of time in a specified

use environment" (IEEE, 1991; Boland, 2002; Khatatneh and Mustafa, 2009; Musa et al., 1984; Goel et al., 1985). Another definition of software reliability is "the probability of the product working correctly over a given period of time" (Mall, 2005). Over the years, many numbers of parametric models and non-parametric growth models have been proposed; there is no such model that can predict reliability across all types of data sets in any environment and in any phase of software development. In present scenario, connectionist approach is showing future success approaches to software reliability prediction and modeling.

The major benefits of software reliability measurement are planning and controlling the resources during software development process for developing high quality software. It gives confidence about software correctness. Also it minimizes the additional cost of testing and improves software reliability (Goel, 1985). At the time of development of any product or system like commercial, military, or any other application, we need to ensure its reliability and consistency in its performance, because a system's reliability has a major impact on maintenance, repair costs, continuity of service, and customer satisfaction (Bhuyan et al., 2015). At the end, the project manager needs to ensure that the software is reliable enough to be released into the market. Keeping the above motivation in mind, in the next paragraph we define our contribution.

The main contribution of this paper is to investigate feed forward back-propagation network (*FFBPN*) model architecture to predict software reliability using failure data. Unlike the traditional neural network modeling approach, we first explain some useful mathematical expressions to our proposed network model. Our propose model's prediction is based on failure data collected in the process of software system testing or operation. The input to this network is designed as the order pair of cumulative execution time (CPU time) and the desired output value (i.e. number of failures). The network is trained to minimize the error between desired and predicted output. An algorithm named *FFBPRP* is described in Section 3.1.2 to calculate the predictive measures. The model predicts the reliability using various prediction criteria that are described in Section 4. The model compute the predictive performance using common predictability measure such as: the Average Error (*AE*), the Root Mean Square Error (*RMSE*), Normalized Root Mean Square Error (*NRMSE*), and common data set. The model performance is further improved by measuring the Mean Absolute Error (*MAE*) along with the above predictability measure. We forecasted short-term prediction and long-term prediction for two numbers of data sets sample. The comparison is performed with the criteria *Short-Term Prediction (STP)* and *Long-Term Prediction (LTP)* with different research work proposed before. The whole idea is to support the project manager to monitor testing, estimating the project schedule, and helping the researchers to evaluate the reliability of the model.

The rest of the paper is organized as follows: Section 2 describes some related work proposed so far in the area of reliability prediction. Section 3 presents the concept feed forward networks model. We describe the proposed model framework, the basic terminologies, application, training of the network, and step-by-step procedure for reliability prediction in this section. In this section, an algorithm is used to compute various predictive criteria. An analysis on software failure data from a medium-sized command and control system, and experimental observation are presented in Section 4. Finally,

in Section 4.2, some comparative studies of various analytical and connectionist models on software failures is carried out. In Section 5, the conclusion and future work are given.

2 Related Work

Artificial Neural Network (ANN) is a powerful technique for *Software Reliability Prediction (SRP)*. Werbose (1988) proposed back-propagation learning and regression technique to identify sources of forecast in uncertainty in a recent gas market model. Though, the author discussed about various methods and techniques, not implemented using his proposed technique on software failure data. Shadmehr et al. (1990) estimated model parameters of pharmacokinetics system using feed-forward multilayered network and predicted the noise residing in the measured data sample. The authors not considered the back-propagation learning for training the network. The authors compared their results with that of the optimal Bayesian estimator and found that their performance was better than the maximum likelihood estimator. The feed- forward network with back propagation learning of artificial neural networks models are applied for software reliability and quality prediction (Khoshgoftaar, 1992; Singh and Kumar, 2010b; Thwin and Quah, 2002). The data sets construction procedure is not explicitly defined. Khoshgoftaar et al. (1992), Singh et al. (2010b), and Thwin et al. (2002), have developed a connectionist model and taken failure data set as input to predict the software reliability, where as the predictive parameter taken is not sufficient for prediction. These work discussed network architecture, method of data representation and some unrealistic assumptions associated with software reliability models.

Pan et al. (2014) made reliability analysis on high *Light-Emitting Diode* (LED). The authors used simulation method to carried out stress accelerated testing and life prediction. Vijay Kumar et al. (2016) proposed an approach to classify the stone textures based on the patterns occurrence on each sub window. The proposed method is tested on Mayang texture images, Brodatz textures, Paul Bourke color images, VisTex database, Google color stone texture images and also original photo images taken by digital camera. Qiuying et al. (2013) computed the number of test cases in the discrete software reliability demonstration testing. Sun (2012) constructed software reliability *Gene Expression Programming* (GEP) model based on usage profile. Maizir et al. (2013) used *ANN* to predict the axial bearing capacity from high strain dynamic testing (i.e. *Pile Driving Analyzer* (PDA)) data. The authors used regression mean and *Mean Square Error* (MSE) as predictable measures for predicting the axial bearing capacity. The authors excluded the strong predictable measures such as *AE*, *RMSE*, *MAE* etc. in their work.

Mark et al. (2015) used directional coding and back-propagation neural network to recognize palm vein. The *Mean Absolute Deviation* (MAD) is implemented as feature vector that were used in the form of input to the beck-propagation network.

Karunanithi et al. (1992) predicted software reliability using feed forward network and recurrent network. The authors compared their result with 14 different data sets and shown that *NN* produced better predictive accuracy compared to analytical models in *end-point predictions*. The authors considered only *average error* (*AE*), *average*

bias (AB), *normalized average error* (NAE) for predictability measure, but not considered *root mean square error* (*RMSE*), *normalized root mean square error*(*NRMSE*). Sitte (1999), analyzed two models for software reliability prediction: 1) neural network models and 2) parametric recalibration models. The author used Kolmogorov distance as the common prediction measure in the comparison experiment. The comparison was performed for next-step[4] prediction and end-point predictions. These approaches differentiate the neural networks and parametric recalibration models in the context of software reliability prediction and conclude that neural networks are much simpler and better predictors. Tian et al. (2005) proposed an useful model for online adaptive software reliability prediction using evolutionary neural network. The prediction is based on software cumulative failure time prediction on multiple-delayed-input single-output architecture. The challenge for this approach is to predetermine the network architecture such as the numbers of the layers and the numbers of neurons in each layer.

RajKiran et al. (2007) predicted software reliability by using *wavelet neural networks (WNN)*. In this paper, the authors employed two kinds of wavelets i.e. *Morlet wavelet* and *Gaussian wavelet* as transfer functions. They made a comparison on test data with various neural network and found that their proposed network performs better than others. In (2008), the authors introduced three linear ensembles and one nonlinear ensemble to compute software reliability. Various statistical and intelligent techniques were used such as *multiple linear regressions* (*MLR*), *multivariate adaptive regression splines* (*MARS*), and *back-propagation trained neural network* (*BPNN*) for software reliability prediction.

Lo (2009) designed a model for software reliability prediction using *ANN*. *Fuzzy Wavelet Neural Network (FWNN)* was used for phase space reconstruction technology and for *SRP* (Zhao, 2010). The authors noticed that the failure history is a vital factor for *SRP*. Ho et al. (2003) proposed connectionist models and used modified Elman recurrent neural network for modelling and predicting software failure. A comparative study was carried out on their proposed model, with the feed-forward neural network, the Jordan recurrent model, and some traditional software reliability growth models. Their experimented results show that the proposed model performed better than the other model.

Pai et al. (2006) used support vector machines and simulated annealing algorithms for reliability forecasting. They used lagged data in their analysis by dividing the 101 observations such as: 33 observations for training, 8 observations for validation and 60 observations for test. Since, it is not a standard method of splitting the data set for experimentation.

[4] short-term prediction and next-step prediction are used interchangeably

3 *FFBPN* Model Architecture and Reliability Prediction

In this section, we attempt to describe the *FFBPN* model's architecture. In order to frame the network, we first discuss the basic concepts of our proposed model *FFBPN* and its application to software reliability prediction.

Here feed-forward neural network is a static network as it is a network with no output feed-back (Chiang et al., 2004). The single-layer feed-forward neural network connected with several distinct layers to form a multilayered feed-forward network is shown in Figure 1. The learning rule applied in this model is parameter learning (Lin and Lee, 1996). The layer that receives input is called the input layer and typically performs no function other than buffering the input signal (Bhuyan et al., 2014). The input layer is not used for computation, so each node of input layer transmits input values to the hidden layer directly. Any layer between the input and output layers are called a hidden layer, because it is internal to the network and has no direct contact with the external environment.

The network reliability prediction process consisted of two steps: a) the training phase b) prediction phase. In the training phase, connecting weights are adjusted to reduce the errors of the network outputs as compared to the actual outputs. At the end of the training phase, the connecting weights between layers are fixed, so that state of any neuron is solely determined by the input-output pattern and not on the initial and past states of the neuron, that is, there is no dynamics involved (Chiang et al., 2004). Normally static-feed forward network is trained using two methods: a) *back-propagation algorithm* b) *conjugate gradient algorithm*. In this paper, the *back-propagation training* algorithm is applied to get back-propagated error using supervised learning. Initially an arbitrary pattern is applied in feed-forward back-propagation network between the units[5]. The error in back-propagation process is propagated towards the hidden layer to minimize the error.

In this paper, we have implemented an algorithm to train the proposed *FFBPN* model and compute the output (i.e. number of failures). This *FFBPN* is work in two phases: 1) *forward propagation* and 2) *back propagation*.

A simple feed forward network assimilates the current activation in memory and generates the output. For each iteration, the errors are back propagated with a set of connection weights. The iteration process may not continue for large epochs[6], because for each layer's error, the error is back propagated, gets smaller and smaller until it converges to zero (i.e. desired output). In this section, an algorithm $FFBPRP$ is applied to compute all predictability measures. The input data structure and its pattern for reliability prediction are also described in this section.

3.1 *FFBPN* architecture construction and training

The proposed model *FFBPN* consists of three-layer network. The network comprises of two step mappings as some compound function, which can be rewritten as nested function that is given in Equation 2 , where x is the input vector to the input layer.

[5] In this paper the terms unit, neuron, and node are used interchangeably.

[6] One pass through all of the training observations (Training Phase) is called an epoch.

The hidden layer nodes are fully connected with input and output layer nodes. The feed-forward network does not consider feedback from output nodes. The basic *feed forward neural network* architecture comprises of two phases: 1) feed forward NN, 2) back propagation with error from output layer. Here the input vector is propagated from input layer to hidden layer as shown in Figure 1. The *FFBPN* model is a simple supervised learning model. The error is calculated for each pair of input pattern and then it is back propagated for training. The error is calculated and weights are folded back to compute the new updated weights for every iteration.

3.1.1 *FFBPN* Architecture Construction: This section gives a brief discussion about *FFBPN* training using back-propagation learning. The operation in this paper is restricted to "hidden" and "output" layers. The input vector 'x' (n-continuous-valued input nodes) is propagated from input layer with weight matrix *V*. Here, the input nodes x_i receive external inputs (i.e. cumulative execution time) after $(i-1)^{th}$ failure interval. Let us consider x_i = 0 for non-input nodes and d_i is the desired output for the desired state of i^{th} unit. Then by taking net input, the output from hidden layer is computed using Equation 1.

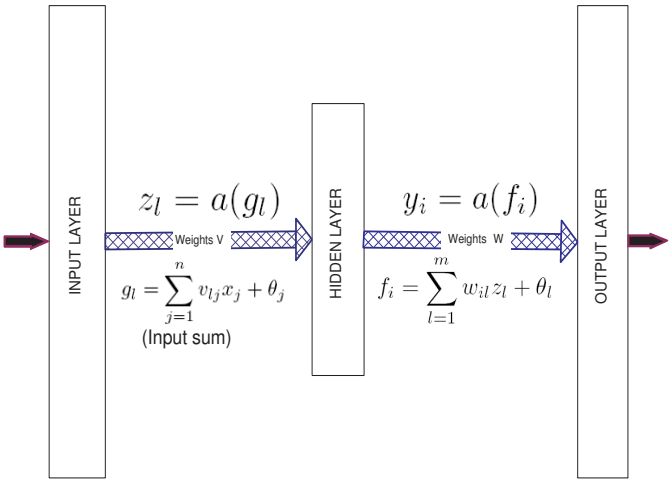


Fig. 1. A simple architecture of feed-forward neural network.

$$z_l = a(g_l), \text{ where, } g_l = \sum_{j=1}^{n} v_{lj} x_j + \theta_j \tag{1}$$

Here, n is the number of input nodes, θ_j is a threshold value, x_j is the input node(i.e. cumulative execution time), v_{lj} is the weight link from j^{th} node of input layer to l^{th}

node of hidden layer, and a is an activation function between the layers that need to be continuous, differentiable, and non-decreasing.

The output y_i represents the numbers of cumulative failures and is calculated using Equation 2. The updated weight w_{il} is computed for each network copy that is summed up with inputs before individual weights are refined. Output y_i can be computed as

$$y_i = a(f_i), \text{where, } f_i = \sum_{l=1}^{m} w_{il} z_l + \theta_l. \tag{2}$$

Here, m is the number of 'hidden' nodes. The weight matrix w_{il} (connected from l^{th} node to i^{th} node) and θ_l is the bias.

3.1.2 Tailoring Neural Network for *FFBPN* Model Prediction Performance: The *FFBPN* is trained by using computed output data and desired output data as it belongs to supervised learning. Assume that there are i numbers of data points are available to train the networks. The first set of analysis is carried out starting with the observations in time to predict the $(i+1)^{th}$ data (Tian and Noore, 2005). This is the order that is used to generate the software failure time data for training purposes. The (i.e. *Mean Square Error (MSE)*) L is calculated by using the cost function using Equation 3.

$$L = \frac{1}{m} \sum_{k=1}^{m} (d_k - y_k)^2 = \frac{1}{2} \sum_{k=1}^{m} L_k^2 \tag{3}$$

Here,

$$L_k = \begin{cases} d_k - y_k, \text{ for } k^{th} \text{ output node} \\ 0, \text{ otherwise} \end{cases}$$

.

L_k is the summation ranges over all the output units, m is the total number of output nodes and it is an index over training sequence matrix. d_k is the actual desired cumulative failures, y_k is the predicted cumulative failures.

Using Steepest-descent method, *FFBPN* network gives a weight update rule that requires a matrix inversion at each step. The network is trained by minimizing the total error which is given in Equation 3.
We accumulate the values of the gradient of the weights changes Δw_{il} and Δv_{lj} which are represented by Equations 4 and 5 respectively.

$$\Delta w_{il} = \eta (d_i - y_i) a'(f_i) z_l \tag{4}$$

$$\Delta v_{lj} = \eta \sum_{i=1}^{m} \left[(d_i - y_i) a'(f_i) w_{il} \right] a'(g_l) x_j \tag{5}$$

where η is a scalar parameter that is used to set the rate of adjustment, referred as learning rate and a' is the derivative of a. We have proposed an algorithm for software reliability prediction. We have named our algorithm *Feed-Forward Back-Propagation Reliability Prediction (FFBPRP)* algorithm. Our model *FFBPN* is trained and the final output is predicted using *FFBPRP* algorithm. Below, we explain the steps of our *FFBPRP* algorithm.

Algorithm *FFBPRP*

Step-1 Initialization
Initialize the weight matrices W, V and the thresholds of the neurons with values in range $(0 \ldots 0.5)$. set $k = 1, L = 0$, where k is the number of pattern, L is the error.

Step-2 Set Tolerable Error
Set the maximum tolerable error E_{max}. Consider the error precision as E_{max}=0.005.

Step-3 Supply Input
Feed the input to the network, the inputs are taken in pairs $\{x^{(1)}, d^{(1)}\}, \{x^{(2)}, d^{(2)}\}, ...$ $\{x^{(p)}, d^{(p)}\}$. Input $x^{(i)}$ represents cumulative execution time, $d^{(i)}$ represents the desired failure number.

Step-4 Training Loop
Apply k^{th} input pattern to the input layer $x^{(k)}$.

Step-5 Forward Propagation
Compute the estimated output $y_1,\ y_2,\ ...,\ y_m$ (i.e. next failure number) using Equation. (2).

Step-6 Compute Error
Calculate the errors L_i for each output node y_i, using Equation. (3)

Step-7 Backward-Error Propagation and Weight Updation
Update the weights W, V using Equation. (4) and (5).

Step-8 Handle Training Conditions
If $k \geqslant$ p, go to Step-9
else update $k = k + 1$, go to Step-4.

Step-9 Error Comparison
If $L \leqslant E_{max}$ go to Step-10
else set $L = 0, k = 1$, go to Step-4.

Step-10 Print Final Weights
Print the values of W and V

Step-11 Compute the Output
Compute the next failures using Equation. (2).

Step-12 Predictive Measures Computation
Calculate the various prediction criterions mentioned in Section 4.

Now, we briefly explain our *FFBPRP* algorithm. First, our algorithm initializes the network parameters along with the weights associated with the layers in Step-1. The maximum tolerable *MSE* value is fixed in Step-2. The input to this network is structured in Step-3. In Step-4, the network is trained and in Step-5, the output of the network is computed. The errors between the predicted and actual values are calculated and compared with the *MSE* which is predefined in Step-6. The computation process and weight adjustment are continued until the *MSE* found falls below a minimum threshold value. Based on this stopping criterion, the network is back propagated towards hidden layer. Then the network is trained using back-propagation learning and weights are adjusted accordingly in Step-7. This process is continued and the maximum tolerable error is compared with *MSE* after each epoch in Step-8 and 9. The final weights are calculated

in Step-10. At the end, the final weights are fixed and the output is recorded for the next cumulative failure in Step-11. In Step-12, the algorithm computes all prediction criteria using formulas that are mentioned in Section 4.

We assume that there are multiple neurons present in the hidden layer. We have taken one hidden layer in our implementation. Although as many as hidden layers can be taken, but the result shows that there is no significant improvement in performance in considering more number of hidden layers. Rather the training performance becomes low. The input layer neurons are exempted from error computation. In the first epoch, the weights are typically initialized, next the set of weights are chosen at random and weights are adjusted in proportion to their contribution to error (Karunanithi et al., 1991). The error is computed in output layer and the difference between the actual output and target output values are calculated using Equation 3.

The cross-validation is carried out by the entire representative data set into two sets: a) a *training data set*, used to train the network, b) a *test data set* used to predict the reliability of the system. We split the data set as follows: 80% for training and 20% for testing. The data sets are pre-processed by normalizing with respect to their maximum values. The model work better in the close interval [0,1] for all data sets. The training pair consists of training inputs to network and target output. The training data is in ordered pair of two dimensional arrays (input-output): $(I_1, O_1), (I_2, O_2)$ ... (I_i, O_i) ..., (I_n, O_n), where I_i and O_i represent the input values (i.e. cumulative execution time) (CPU time) and desired output value (i.e. number of failure) respectively. We can interpret number of failures as a function of cumulative execution time. The weights are structured in two dimensional data arrays (i.e. input-to-hidden layer and hidden-to-output layer). In this work, the logistic function, binary sigmoidal $f(x) = 1/(1 + e^{-\lambda x})$ is used, where x is the cumulative execution time. The binary sigmoidal function is used to reduce the computational burden during training (Sivanandam and Deepa, 2007). The function $f(x)$ is continuous, differentiable, and monotonically increasing. Here λ is the steepness parameter at each step of learning. The range of this logistic function varies from 0.0 to 1.0.

4 Experimental Results and Comparison with Related Work

We have implemented our proposed approach using *MATLAB* Version 7.10. We initialize V & W weight matrices with small random values before first epoch is started. After the first epoch, weights are adjusted randomly. The *FFBPRP* algorithm then calculates a sum squared error between the desired output and actual output with help of error function. The target values are not present for hidden units. Here the error precision (i.e. the maximum tolerable (*MSE*) is fixed and is taken as E_{max}=0.005. The network model *FFBPN* is trained with initial weights and continues until the stopping criterion is satisfied and best weights are recorded. Here, we carried out two types of experimentation; (a) *next-step prediction* or *short-term*[7] *prediction* of the reliability and (b) the *end-point prediction*[8] is performed at the end of a future testing and debugging session (Karunanithi et al., 1992a).

[7] *STP* and *next-step prediction* are used interchangeably

[8] *LTP* and *end-point predictions* are used interchangeably

Normally, the data set used for reliability growth model are having defect severities[9] 2 and 3 as per *TANDEM* report (Wood, 1996). In our experiment, we considered the failure data during system testing phase of various projects collected at Bell Telephone Laboratories, *Cyber Security and Information Systems Information Analysis Center* by John D. Musa (Musa, 1980) and from research work (Mohanty et al., 2013). For our experimentation, we collected two numbers of software failure data sets from various literature and that are shown in Table 1. The actual data set[10] shown in Table 1 consists of a) Failure Number, b) Time Between Failures (*TBF*), c) Day of Failure of different medium-sized applications, such as *Real-Time Command & Control Software System*, *Commercial System*, *Military Application*, *Operating System*, *Time Sharing system*, and *Word Processing System*.

We predicted the software reliability in *Short Term Prediction (STP)* that is used to measure the current reliability of a system and *Long Term Prediction (LTP)* used for decision making about long-term test plans. In *STP*, we considered x_i as the time between the $(i-1)^{th}$ and i^{th} software failure and d_i is the number of failures. We can interpret number of failures as a function of cumulative execution time. So, it can be written as $f(x_i) = d_i$. The normalized values of the input to the network such as $f(x_1), f(x_2)...f(x_i)$ are used to predict the $\hat{d}_{i+1}$, where $\hat{d}_{i+1}$ is the computed value (i.e. next number of failures) . In other way, we can forecast $\hat{d}_{i+1}$ by using $\{x_1, d_1\}, \{x_2, d_2\}, ...\{x_i, d_i\}$, where d_{i+1} is the corresponding target value is known as *short term prediction* or *1-step ahead prediction* or *next-step prediction*. Similarly, suppose $f(x_1), f(x_2)...f(x_i)$ are used to predict the $\hat{d}_{i+1}$, where d_{i+1} is the corresponding target value and $f(x_2), f(x_3)...f(x_{i+1})$ are used to predict the $\hat{d}_{i+2}$, where d_{i+2} is corresponding target value, where $\hat{d}_{i+2}$ is the computed value (i.e. next number of failures). Continuing in this way up to n^{th} pattern is known as *long term prediction* or *n-step step prediction* or *end-point prediction*.

Some prediction criteria are listed below:

- **The Average Error (*AE*)**, computes how adequately a model predicts all over the system testing phase (Karunanithi et al., 1992b). *AE*, measures how well a model predicts throughout the testing phase (Karunanithi et al., 1992). *AE* is used to compare our model with other models to test the predictive accuracy.
  Relative Error(%) $RE_i = (|(F_i - D_i)/D_i|) * 100$
  Average Error(%) $AE = 1/n \sum_1^n RE_i$
- **The Root Mean Square Error (*RMSE*)**: *RMSE* is used to compute how far on average the error (i.e. between actual and target value) is from 0. The lower is *RMSE*, the higher is prediction accuracy.
  $RMSE = \left[\sqrt{\sum_1^n (F_i - D_i)^2}\right] / n$

[9] The severity levels defined as per the urgency of the customer needs. Severity 0 is No Impact: Can tolerate the situation indefinitely. 1 is Minor: Can tolerate the situation, but expect solution eventually. 2 is Major: Can tolerate the situation, but not for long, Solution needed. 3 is Critical: Intolerable situation. Solution urgently needed.

[10] Collection of data set from large projects is a difficult task because software industry considered their failure history as classified record (Karunanithi et al., 1992b).

Normalized Root Mean Square Error (*NRMSE*)
$NRMSE = \left[\sqrt{\sum_1^n (F_i - D_i)^2}\right] / \sum_1^n {F_i}^2$

– **Mean Absolute Error (*MAE*)** is an average of an absolute error that computes how close predictions are to the final result. The *MAE* and *RMSE* are used together to analyze the variation in the errors on data set.
$MAE = \left[\sum_1^n |(F_i - D_i)|\right] / n$

Where, F_i is the computed output and D_i is the target output. The lesser computed value of *AE*, *RMSE*, *NRMSE*, and *MAE* indicates the higher is the accuracy in prediction.

Table 1. Data set with number of failures used in the experiments from Musa (1980) & Iyer and Lee (1996).

Project Code	Project Name	Number of Failures	Development Phases
DBS-1	Real-Time Command & Control System	136	System Test Operations
DBS-2	(Iyer et al., 1996)	191	System Test

Table 2. Data set by Musa (Musa, 1980) for DBS-1.

Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time
1	3	36	5389	71	16229	106	46653
2	33	37	5565	72	16358	107	47596
3	146	38	5623	73	17168	108	48296
4	227	39	6080	74	17458	109	49171
5	342	40	6380	75	17758	110	49416
6	351	41	6477	76	18287	111	50145
7	353	42	6740	77	18568	112	52042
8	444	43	7192	78	18728	113	52489
9	556	44	7447	79	19556	114	52875
10	571	45	7644	80	20567	115	53321
11	709	46	7837	81	21012	116	53443
12	759	47	7843	82	21308	117	54433
13	836	48	7922	83	23063	118	55381
14	860	49	8738	84	24127	119	56463
15	968	50	10089	85	25910	120	56485
16	1056	51	10237	86	26770	121	56560
17	1726	52	10258	87	27753	122	57042
18	1846	53	10491	88	28460	123	62551
19	1872	54	10625	89	28493	124	62651
20	1986	55	10982	90	29361	125	62661
21	2311	56	11175	91	30085	126	63732
22	2366	57	11411	92	32408	127	64103
23	2608	58	11442	93	35338	128	64893
24	2676	59	11811	94	36799	129	71043
25	3098	60	12559	95	37642	130	74364
26	3278	61	12559	96	37654	131	75409
27	3288	62	12791	97	37915	132	76057
28	4434	63	13121	98	39715	133	81542
29	5034	64	13486	99	40580	134	82702
30	5049	65	14708	100	42015	135	84566
31	5085	66	15251	101	42045	136	88682
32	5089	67	15261	102	42188		
33	5089	68	15277	103	42296		
34	5097	69	15806	104	42296		
35	5324	70	16185	105	45406		

The data sets DBS-1 and DBS-2 are considered for analysis and observation in *next-step predictions* and *end-point predictions* using *FFBPN* model. We summarize these computed values in Table 3 for next-step prediction and Table 4 for long-term prediction.

Table 3. Predictive results in *STP* using two data sets.

Data set	AE	RMSE	NRMSE	MAE	Mean Error
DBS-1	3.0018	0.0042	0.0334	0.0243	-0.0019
DBS-2	3.3540	0.0061	0.0721	0.0353	-0.0092

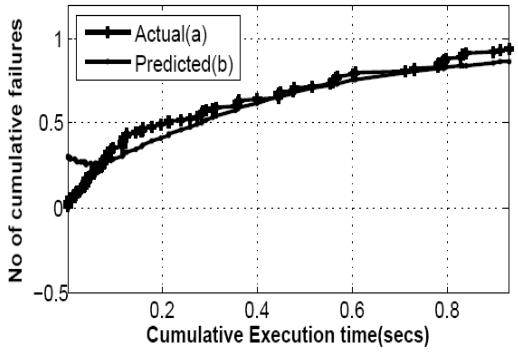
Table 4. Predictive results in *LTP* using two data sets.

Data set	AE	RMSE	NRMSE	MAE	Mean Error
DBS-1	3.3268	0.0062	0.0843	0.0233	0.0044
DBS-2	4.2441	0.0358	0.0984	0.0548	-0.0078

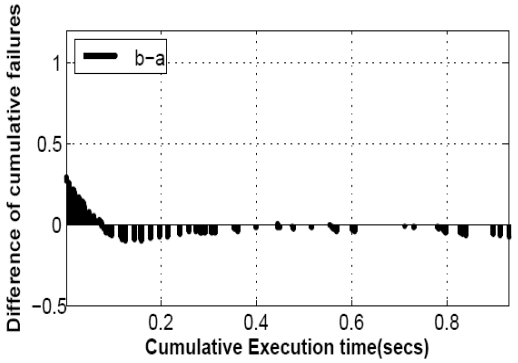
4.1 Analysis and Verification for DBS-1 and DBS-2

Using data set DBS-1, we carried out two analyses; 1) Results in *next-step* prediction, 2) Results in *end-point* prediction. Let us consider data set DBS-1 of Table 2 that contains 136 failures for analysis and verification purpose. After the network model is successfully trained, the weights are fixed and computations for measurement unit are performed. The predicted result for *next-step prediction* of various measurement units are shown in Table 3. Figure 2(a) shows the desired output and computed output against cumulative execution times. The Figure 2(a) shows the performance of the proposed model in terms of the number of cumulative failures w.r.t. the cumulative execution times, for the actual and predicted data points in *short-term predictions*. Figure 2(b) shows the deviation between the predicted value and actual target value in *short-term predictions*. The relative errors of Figure 2(a) are represented in Figure 2(b), which shows the closeness (accuracy) of prediction of the model in *short-term predictions*.

Then, the model under goes for *end-point predictions* using test data on data set sample *DBS-1*. The predicted result for *end-point prediction* of various measurement units are shown in Table 4. The prediction performance for *end-point predictions* are demonstrated in Figures 3(a)-3(c). Figure 3(c) is the plot between numbers of epochs and error rate in terms of *RMSE* during prediction. The observation on this predicted data and comparison is described in Section 4.2. Figure 3(c) shows the error rate (in terms of *RMSE*) w.r.t. the number of epochs. From Figure 3(c), it can be observed that, the error rate decreases as the number of epochs increases. This shows that our proposed model performs accurate prediction.



(a) *STP* of *FFBPN* network using data set DBS-1.

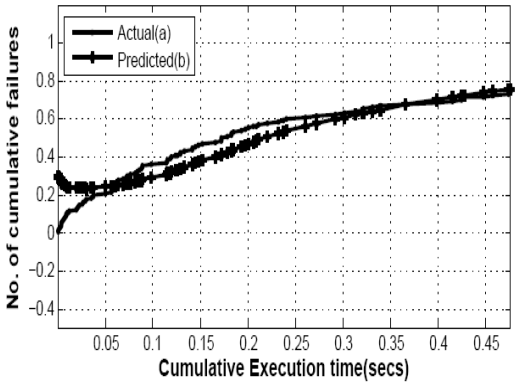


(b) Deviation between the *STP* value and the actual value using data set DBS-1.

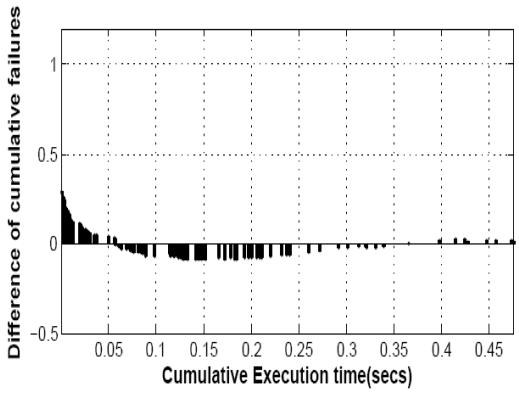
Fig. 2. Short Term Prediction and relative error of *FFBPN* network using data set DBS-1.

Analysis and Verification for DBS-2: Using data set DBS-2, we carried out two analyses; 1) Results in *next-step* prediction, 2) Results in *end-point* prediction. The various results for data set DBS-2 is given in Table 5 which contains 191 failures. The predicted result for *next-step* prediction of various measurement units are shown in Table 3. The relative errors of Figure 4(a) are represented in Figure 4(b), which shows the closeness (accuracy) of prediction of the model in *short-term* predictions.

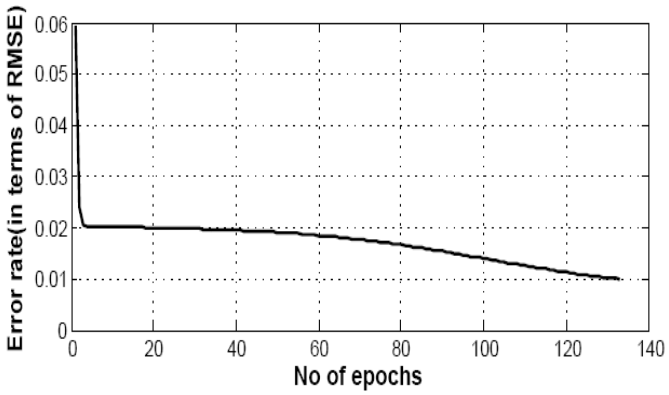
Then, the model under goes for *end-point* predictions using test data on data set sample *DBS-2*. The predicted result for *end-point* prediction of various measurement units are shown in Table 4. The prediction performance for *end-point predictions* are demonstrated in Figures 5(a)-5(c). Figure 5(c) is the plot between number of epochs and



(a) *LTP* of *FFBPN* network using data set DBS-1.



(b) Deviation between the *LTP* value and the real value using data set DBS-1.



(c) Performance result against *RMSE* using data set DBS-1.

Fig. 3. *LTP* and relative error of *FFBPN* network using data set DBS-1.

Table 5. Data set by Iyer and Lee (1996) for DBS-2.

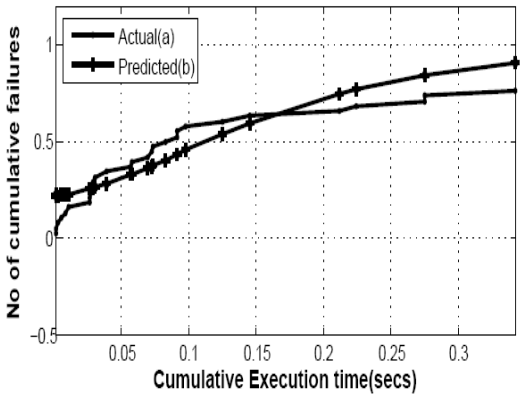
Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time	Failure No	Cumulative Execution Time
1	9.9898	40	369.61	79	858.68	118	1336.2	157	1792.4
2	18.747	41	379.51	80	872.67	119	1349.9	158	1806.6
3	28.962	42	391.11	81	879.07	120	1363.9	159	1820.8
4	40.719	43	403.37	82	885.46	121	1377.8	160	1835.1
5	52.872	44	417.38	83	889.07	122	1383	161	1847.8
6	61.037	45	431.38	84	902.73	123	1388.2	162	1861.5
7	70.447	46	445.39	85	916.75	124	1396.9	163	1875.7
8	80.03	47	453.99	86	930.94	125	1409.4	164	1890.3
9	88.819	48	462.8	87	945.24	126	1422.5	165	1904.9
10	100.3	49	472.18	88	959.53	127	1436.2	166	1916.9
11	110.31	50	483.2	89	973.83	128	1449.8	167	1930.2
12	117.3	51	494.25	90	988.13	129	1463.7	168	1943.5
13	124.36	52	505.39	91	993.81	130	1478.2	169	1957.9
14	130.85	53	516.55	92	1001.5	131	1485.7	170	1972.3
15	137.48	54	527.7	93	1009.5	132	1496.8	171	1986.7
16	143.67	55	539.81	94	1017.5	133	1509.7	172	2001.3
17	149.64	56	551.94	95	1025.9	134	1522.8	173	2015.8
18	154.47	57	563.97	96	1034.6	135	1536.9	174	2025.7
19	164.37	58	576.01	97	1048.3	136	1551.3	175	2038.1
20	177.25	59	588.08	98	1062	137	1565.9	176	2050.9
21	183.9	60	600.39	99	1075.8	138	1580.5	177	2062.3
22	191.83	61	612.71	100	1089.6	139	1595.2	178	2075.6
23	200.02	62	625.03	101	1103.3	140	1609.8	179	2088.9
24	208.79	63	637.37	102	1117.1	141	1620.8	180	2102.8
25	218.06	64	650.49	103	1130.9	142	1628.6	181	2113.5
26	227.6	65	664.14	104	1145.4	143	1639.5	182	2124.3
27	237.5	66	677.97	105	1159.9	144	1650.7	183	2135.6
28	247.47	67	691.79	106	1174.5	145	1661.8	184	2147.4
29	257.56	68	705.63	107	1189	146	1669.4	185	2160.1
30	267.7	69	719.47	108	1203.5	147	1677.5	186	2172.8
31	280.18	70	733.31	109	1218.3	148	1686.3	187	2186
32	292.96	71	747.19	110	1233.3	149	1695.5	188	2199.1
33	305.79	72	761.09	111	1248.2	150	1705.1	189	2212.3
34	319.6	73	775	112	1263.1	151	1715	190	2225.5
35	328.15	74	788.92	113	1278	152	1727.8	191	2238.7
36	336.82	75	802.83	114	1284.3	153	1740.6		
37	345.49	76	816.77	115	1296.5	154	1753.5		
38	354.17	77	830.72	116	1309.4	155	1766.3		
39	362.81	78	844.69	117	1322.4	156	1779.1		

error rate in terms of *RMSE* during prediction. The observation on this predicted data and comparison is described in Section 4.2. Figure 5(c) shows the error rate (in terms of *RMSE*) w.r.t. the number of epochs. From Figure 5(c), it can be observed that, the error rate decreases as the number of epochs increases. This shows that our proposed model performs accurate prediction.

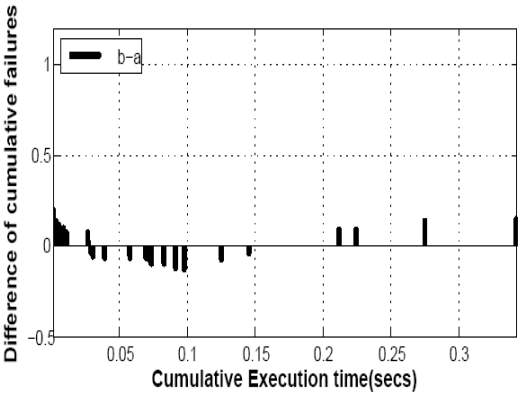
The *short term predictions* result shows better accuracy than *end-point predictions* for both the data sets. The *AE*, we found in this experiment show better accuracy and consistent result than some well-known methods that are presented in Table 6 Table 7.

4.2 Comparison with Related Work and Observations

The prediction accuracy on the whole yields by using a large number of data sets. Since it is not practically feasible to represent graphically for large number of data set within these limited pages, we summarize the comparison using *DBS-1* and *DBS-2*. The comparison between some analytical models and the *feed forward neural network* models are summarized for *DBS-1* and *DBS-2* in Table 6 Table 7 respectively. It is observed that *FFBPN* is the better predictor than that of the described models in Table 6 Table 7.



(a) *STP* of *FFBPN* network using data set DBS-2.

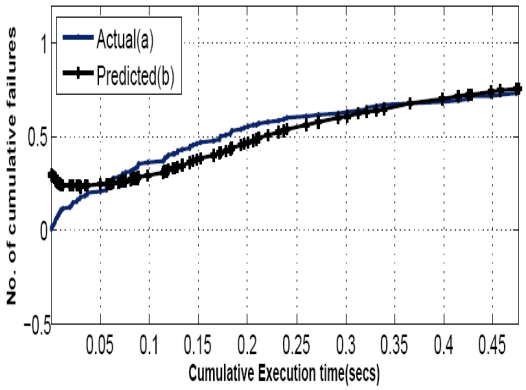


(b) Deviation between the *STP* value and the real value using data set DBS-2.

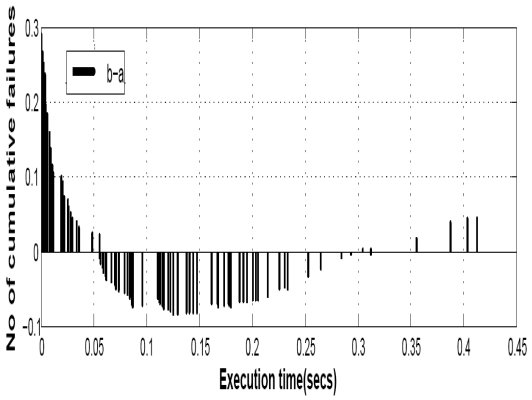
Fig. 4. *STP* and relative error of *FFBPN* network using data set DBS-2.

The accuracy and consistency of software are measured by the value of *NRMSE* and *AE* on data set which is also used as software measurement criteria. From the above result, it is found that the *next-step* prediction shows better accuracy than *end-point* predictions results for both data sets *DBS-1* & *DBS-2*. From Table 3 and 4, we observed that *DBS-1* produce better result than predictive results of *DBS-2*.

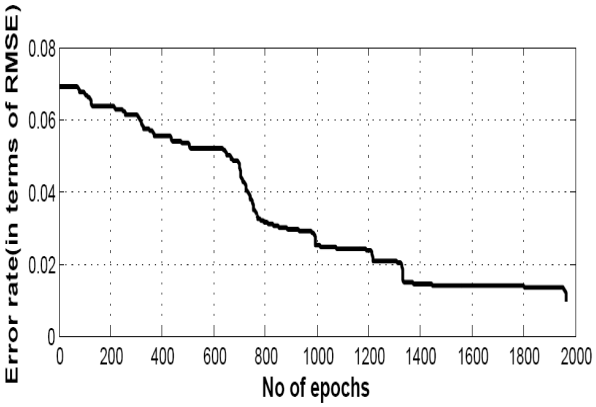
The comparison between some analytical models and the proposed feed-forward neural network model for software reliability prediction for DBS-1 is summarized in



(a) *LTP* of *FFBPN* network using data set DBS-2.



(b) Deviation between the *LTP* value and the actual value using data set DBS-2.



(c) Performance result against *RMSE* using data set DBS-2.

Fig. 5. *LTP* and relative error of *FFBPN* network using data set DBS-2.

Table 6. The *AEs* found in this experiment in *next-step* prediction shows better accuracy and consistent result than some well known methods.

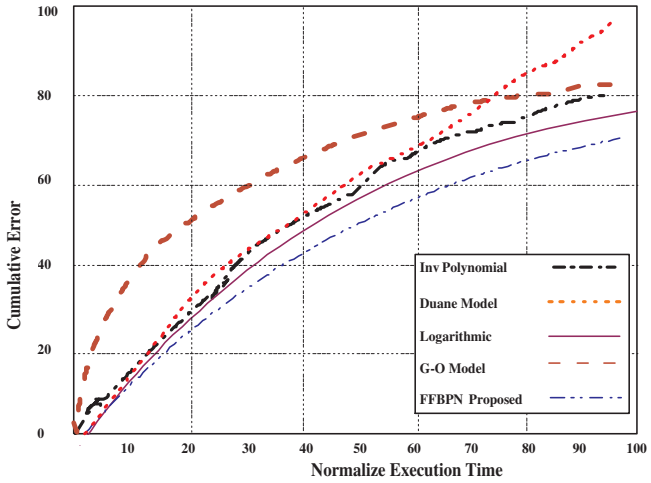


Fig. 6. Comparison with other models for data set DBS-1.

In Figure 6, we experimented on well-known parametric software reliability growth models such as: *Inv Polynomial model proposed by Littlewood and Verrall (2009), Duane Model (2009), Logarithmic model proposed by Musa and Okumoto (1987), G-O Model (1979*) with our propose model *FFBPN* in *STP*. We observed that our propose model produce satisfactory results than the above model mentioned in Table 6 in *STP*. It is also observed that the propose model *FFBPN*, predictive performance in *STP* is better than in *LTP* for both data sets.

Moreover, in Table 7, the *NRMSE* value is showing satisfactory result than that of Mohanty et al. (2013). Mohanty et al. (2013) also used the same data set to test the efficacy of their method. However, since they used the lagged data in their experimentation. Here, our results compared with one of their lagged result.

It is observed that *next-step* prediction produces better accuracy compared to *end-point predictions* for data sets *DBS-1 & DBS-2*.

The *STP* result in Table 6 shows that our proposed model has less *AE* values than the other models in (Karunanithi et al., 1992b). Again by comparing this *NRMSE* with the *NRMSE* of many related work (Mohanty et al., 2013), it is observed that *FFBPN* is giving satisfactory result.

Model quality is considered to be better if its prediction points are close to the ideal line passing through the zero error (Karunanithi et al., 1992). In these experiments, Figure 2(b), Figure 3(b), Figure 4(b), and Figure 5(b) show the prediction closeness between the actual vale and prediction value. It is found that the difference between

Table 6. Comparison of *AE* values with various approaches.

Approach	Values
FFN Generalization (Karunanithi et al., 1992b) (with encoding)	9.48
(Tian and Noore, 2004)	4.51
(Cost et al., 2005)	7.15
Inv Polynomial	10.64231
Logarithmic	11.97389
Duane Model	10.5328
G-O Model	4.4835
FFBPN(Proposed)(DBS-1)	3.0018
FFBPN(Proposed)(DBS-2)	3.3540

Table 7. Comparison of *NRMSE* values with various approaches.

Approach	Values
GMDH (Mohanty et al., 2013)	0.076335
(Kiran et al., 2007)	0.119402
Inv Polynomial	1.51
Logarithmic	2.24
Duane Model	1.71
G-O Model	2.34
GP Model (Cost et al., 2005)	4.74
FFBPN(Proposed) (DBS-1)	0.0334
FFBPN(Proposed) (DBS-2)	0.0721

two predicted values is significantly close to each other, which indicates that the lesser *AE*, higher is the accuracy in prediction.

Some observations on software reliability prediction using our proposed *feed forward neural network* model are listed below:

- The network can easily be built with a simple optimizing algorithm (e.g. Steepest descent method) and less memory storage requirement that accelerates the training process as well.
- The training time cost is minimum in this model for less number of epochs required for output prediction.
- It is producing better accuracy and consistency for experimented data set like DBS-1 & DBS-2.

4.3 Threats to Validity

Below we discuss the possible internal and external threats to the validity of our work.

- Our experiment uses MATLAB as tool for reliability prediction and therefore suffers the same threats to validity as MATLAB does (i.e. Finding the exact result and number of epochs may not be exact value for every run).
- Weights are taken randomly, so every time expecting the same result on the same number of epochs is an issue.

- Our model may not produce satisfactory result for insufficient training data size.
- Our model cannot manage well with major changes that are not reflected in training phase.

Our *FFBPN* model is a powerful model because of it's nonlinear and logistic properties and having less constraints on the number of observations.

5 Conclusions

In this paper, we presented a novel technique for software reliability prediction using *feed forward neural network* with back-propagation. Our proposed technique accurately predicts the software reliability. Unlike most of the computational intelligence models, our model computes *AE*, *RMSE*, *NRMSE*, *MAE* simultaneously. In this work, most of the predictive criteria are considered. We presented experimental evidence showing that feed forward network with back propagation yields accurate result comparable to other discussed methods. Results shows with two data sets, suggest that the *FFBPN* model is better at *short-term* prediction than *end-point* predictions. Our experimental results show that this approach is computationally feasible and can significantly reduce the cost of testing the software by estimating software reliability. The proposed network can significantly influence the predictive performance. In our future work, we will try to develop a model to predict software reliability at an early stage of software development using some hybrid techniques such as *neuro-fuzzy, neuro-genetic, fuzzy-genetic* of *computational intelligence*.

References

Benala T.R., Dehuri S., Mall R. (2012), *Computational intelligence in software cost estimation: an emerging paradigm*, ACM SIGSOFT Software Engineering Notes, Vol. 37, no. 3, pp. 1-7.

Bhuyan M.K., Mohapatra D.P., Sethi S., Kar S. (2015), *An Empirical Analysis of Software Reliability Prediction Through Reliability Growth Model Using Computational Intelligence*, *Computational Intelligence in Data Mining,* Proceedings of the International Conference on CIDM, SPRINGER Smart Innovation, Systems and Technologies, Series Vol. 32, pp. 513-524.

Bhuyan M.K., Mohapatra D.P., Sethi S. (2014), *A survey of computational intelligence approaches for software reliability prediction*, ACM SIGSOFT Software Engineering Notes, Vol. 39, no. 2, pp. 1-10.

Boland P.J.(2002), *Challenges in software reliability and testing*. Technical report, Department of Statistics, National University of Ireland, Dublin Belfield - Dublin 4 Ireland.

Chiang Y.M., Chang L.C., Chang F.J. (2004), *Comparison of static-feedforward and dynamic-feedback neural networks for rainfall-runoff modeling*, Journal of Hydrology, Elsevier, Vol. 290, no. 3-4, pp. 297-311.

Costa E.O., Aurora S.R.V., Souza P.G. (2005), *Modeling software reliability growth with genetic programming* (Proceedings of the 16^{th} IEEE International Symposium on Software Reliability Engineering, Chicago, Illinois).

Goel A.L., Okumoto K. (1979), *Time-dependent fault detection rate model for software and other performance measures*, *IEEE Trans. Reliability*, Vol. R-28, no. 3, pp 206-211.

Goel A.L. (1985), *Software reliability models: assumptions, limitations, and applicability*, IEEE Transection on Software Engineering, Vol. 11, no. 12, pp. 1411-1423.

Ho S.L., Xie M., Goh T.N. (2003), *A Study of the connectionist models for software reliability prediction*, *An international Journal computers & mathematics with applktations*, ELSEVIER, Vol. 46, no. 7, pp. 1037-1045.

Hsu C.J., Huang C.Y. (2011), *An adaptive reliability analysis using path testing for complex component-based software systems*, *IEEE Transactions on Reliability*, Vol. 60, no. 1, pp. 158-170.

IEEE (1991), *Standard glossary of software engineering terminology.* Standards Coordinating Committee of the IEEE Computer Society.

Iyer R.K., Lee I. (1996), *Measurement-based analysis of software reliability, Handbook of Software Reliability Engineering* (McGraw-Hill), pp. 303-358.

Karunanithi N., Malaiya Y., Whitley D. (1991), *Prediction of software reliability using neural networks*, in *Proceedings of IEEE International Symposium on Software Reliability Engineering* (IEEE, Austin, TX), pp. 124–130.

Karunanithi N., Whitley D. (1992), *Prediction of software reliability using feed-forward and recurrent neural nets*, in *Neural Networks, IJCNN*,(IEEE, Baltimore, MD), Vol. 1, pp. 800–805.

Karunanithi N., Whitley D., Malaiya Y.K. (1992), *Using neural networks in reliability prediction*, IEEE Software Vol. 9, no. 4, pp. 53-59.

Karunanithi N., Whitley D., Malaiya Y.K. (1992), *Prediction of software reliability using connectionist models*, *IEEE Trans. on Software Eng*. Vol. 18, no. 7, pp. 563-574.

Khatatneh K., Mustafa T.(2009), *Software reliability modeling using soft computing technique*, *European Journal of Scientific Research*, Vol. 26, no. 1, pp. 154-160.

Khoshgoftaar T.M. , Pandya A.S., More H. (1992), *A neural network approach for predicting software development faults*, (Proceedings of Third International Symposium on Software Reliability Engineering, Research Triangle Park, NC), pp. 83–89.

Kumar P. V. V. S. R., Madhuri N., Devi M. U. (2016), *Stone Image Classification Based on Overlapped 5-bit T-Patterns occurrence on 5-by-5 Sub Images*, *International Journal of Electrical and Computer Engineering (IJECE)*, Vol. 6, no. 3.

Littlewood B., Verrall J. L. (2009), *A Bayesian reliability model with a stochastically monotone failure rate*, *IEEE Trans. Reliability*, Vol. R-23, no. 2, pp. 108-114.

Maizir H., Kassim K. A. (2013), *Neural Network Application in Prediction of Axial Bearing Capacity of Driven Piles*, IAENG Vol, 1, IMECS 2013, *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Hong Kong, March

Malaiya Y.K., Karunanithi N., Verma P. (1992), *Predictability of software reliability models*, *IEEE Transactions on Reliability*, Vol. 41, no. 4, pp. 539-546.

Mall R. (2005), *Fundamentals of Software Engineering*, 2nd edn. (Prentic-Hall India, New Delhi, India).

Mohanty R., Ravi V., Patra M.R. (2013), *Hybrid intelligent systems for predicting software reliability*, *Applied Soft Computing*, Vol. 13, no. 1, pp. 189-200.

Mark E. C. V., Linsangan N. B. (2015), *Palm Vein Recognition System using Directional Coding and Back-propagation Neural Network*, IAENG, Vol. II, WCECS 2015, *Proceedings of the World Congress on Engineering and Computer Science*, San Francisco, USA, October

Musa J.D. (1980), Software Reliability Data. Data & Analysis Center for Software.

Musa J.D., Okumoto K. (1984), *A logarithmic poisson execution time model for software reliability measurementin ICSE*. IEEE Press, Piscataway (Proceedings of the 7^{th} International Conference on software Engineering, NJ, USA), pp. 230–238.

Musa J. D., Iannino A., and Okumoto K. (1987). *Software reliability: measurement, prediction, application*, New York, USA: McGraw-Hill.

Lin C.T., Lee C.G. (1996), *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems* (Prentice Hall, Inc).

Lo J.H. (2009), *The Implementation of artificial neural networks applying to software reliability modeling*, In *proceedings of Control and Decision Conference*. CCDC '09. Chinese, pp. 4349-4354.

Pai P.F., W.C. Hong (2006), *Software reliability forecasting by support vector machines with simulated vector machines with simulated annealing algorithms*, *Journal of Systems and Software*, ELSEVIER, Vol. 79, no. 6, pp. 747-755.

Pan K., Guo Y., Zhu W., Wang X., Bin Z. (2014), *Study on Reliability and Lifetime Prediction of High Power LEDs*, TELKOMNIKA *Indonesian Journal of Electrical Engineering*, Vol. 12, no. 2, pp. 1132-1142,

Qiuying L., Lei L. (2013), *Determining the Minimal Software Reliability Test Effort by Stratified Sampling*, TELKOMNIKA, *Indonesian Journal of Electrical Engineering*, Vol. 11, no. 8, pp. 4399-4406.

RajKiran N., Ravi V. (2007), *Software reliability prediction using wavelet neural networks*, in *International Conference on Computational Intelligence and Multimedia Applications*, IEEE, Sivakasi, Tamil Nadu, Vol. 1, pp. 195 – 199.

RajKiran N., Ravi V. (2008), *Software reliability prediction by soft computing techniques*, *Journal of Systems and Software*, ELSEVIER, Vol. 81, no. 4, pp. 576-583.

Shadmehr R., Argenio D.Z.D. (1990), *A comparison of a neural network based estimator and two statistical estimators in a sparse and noisy data environment*, in *IJCNN*, Vol. 1 (Washington D.C), pp. 289–292.

Singh Y., Kumar P. (2010), *Application of Feed-Forward Neural Networks for Software Reliability Prediction*, *ACM Sigsoft Software Engineering Note*, Vol. 35, no. 5, pp. 1-6.

Singh Y., Kumar P. (2010), *Prediction of software reliability using feed forward neural networks*, in *Computational Intelligence and Software Engineering (CiSE)*, 10-12 Dec. 2010, IEEE, pp. 1–5.

Sitte R. (1999), *Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration*, *Reliability, IEEE Transactions*, Vol. 48, no. 3, pp. 285-291.

Sivanandam S.N., Deepa S. (2007), *Principles of Soft Computing*, 1st edn. (Wiley India (P) Ltd., Ansari Road Daryaganj, New Delhi)).

Sun S.J., Xiao J. (2012), *A Software Reliability GEP Model Based on Usage Profile, TELKOMNIKA Indonesian Journal of Electrical Engineering*, Vol. 10, no. 7, pp. 1756-1764.

Thwin M.M.T., Quah T.S. (2002) (Ed.), *Application of neural network for predicting software development faults using object-oriented design metrics*, Vol. 5, *Proceedings of the* 9^{th} *International Conference on Neural Information Processing (ICONIP'02)*.

Tian L., Noore A. (2004), *Software Reliability Prediction Using Recurrent Neural Network With Bayesian Regularization*, International Journal of Neural Systems, Vol. 14, no. 3, pp. 165-174.

Tian L., Noore A. (2005), *On-line prediction of software reliability using an evolutionary connectionist model*, *The Journal of Systems and Software*, Vol. 77, no. 2, pp. 173-180.

Werbos P. (1988), *Generalization of backpropagation with application to recurrent gas market model*, Neural Network , Vol. 1, pp. 339-356.

Wood A. (1996). *Software reliability growth models.*, Technical Report 96.1, Tandem Computers.

Zhao L., Zhang pei J., Yang J., Chu Y. (2010), *Software reliability growth model based on fuzzy wavelet neural network* in *2nd International Conference on Future Computer and Communication (ICFCC)*, IEEE, Wuhan, Vol. 1, pp. 664–668.

Zheng J. (2009), *Predicting software reliability with neural network ensembles*, *Expert Systems with Applications*, Vol. 36, ELSEVIER, pp. 21162122.

Received March 11, 2016 , revised May 18, 2016, accepted October 5, 2016