Predicting Time between Software Failures Using ISGNN

Aiguo Li[†,††] , *Dashan Qiu*[††] *and ZhanHuai Li*[†],

[†] School of Computer Science and Engineering,
Northwestern Polytechnical University, 710072, Xi'an, China
[††]Department of Computer Science and Technology, Xi'an University of Science and Technology, 710054, Xi'an, China

Summary
Neural networks methods have been used in prediction of time between software failures. However, most of the existing methods have some shortages, including requiring much complex computing and requiring advanced users to set their network structures and many parameters. Iteration learning self-generating neural networks (ISGNN) is an improved self-generating neural networks (SGNN). It has inherited the advantages of SGNN, for example, users do not need to set network structures and parameters and it has better classification precision. We propose a scheme based on ISGNN to predict time between software failures in this paper. Two real failure datasets are used in experiments in this paper. Experimental results show that ISGNN is easy to use and its training time consume is about one sixth of BP (Back-Propagation) networks', and the MAE (mean absolute error) and RMSE (root mean square error) of ISGNN are both reduced about 3%~15% than that of BP neural networks.

Key words:
software reliability; neural networks; prediction of time between failures; self-generating neural network

Introduction

Software reliability modeling is a subject of growing importance. Its aim is to improve software reliability and helps to develop reliable software and check software reliability. There are many factors that may affect software reliability, such as software development environment, software development organization and software complexity, and so on [1-3]. Generally speaking, these factors are highly interactive and demonstrate non-linear patterns. Traditional statistical modeling is imposed severe limitations, such as depending heavily upon the assumptions of independence and linearity [4]. The neural networks model requires only failure history as input and predicts future failures more accurately than some analytic models. Therefore, it has been used in software reliability modeling, evaluation and prediction. But like general application of neural networks, using the method based on neural networks to predict time between software failures requires users to set the network structures and parameters and so on. And these works are the most important and the most difficult issue. Consequently, a universal model that can provide highly accurate prediction under all circumstances without any assumptions is most desirable.

ISGNN (Iteration learning SGNN) [5] is an improved SGNN (self-generating neural networks) [6]. It has inherited the advantages of SGNN, for example, users do not need to set network structures and parameters and it has better precision. We propose a scheme based on ISGNN to predict time between software failures in this paper. The scheme has some advantages, for example, it can reduce much of the learning time consumption and has more precise predicting results.

The principle of using ISGNN to predict time between software failures has three steps: (1) preprocessing the dataset of time between software failures by polynomial fitting; (2) generating an SGNT (self-generating neural tree) through learning in the training sample dataset; (3) using SGNT to predict the testing sample dataset. Experimental results show that ISGNN method proposed in this paper is a potential method for predicting time between software failures.

2. Iteration learning Self-Generating Neural Networks

Conventional neural networks are needed advanced users to design the network structures. It updates the connect weights among consequent layers by learning from sample dataset. Consequently, obtaining a suitable structure of networks is difficult. However, SGNN algorithm generates an SGNT (self-generating neural tree) by learning from the sample dataset. Entire structure contains neurons, the relation and weight value of neurons, and all are generated automatically from training dataset directly. Hence it has advantages of adaptability and flexibility. Wen et al [6] gave some definitions as follows:

Definition 1: The input sample data e_i is composed of its attributes $e_i = (e_{i1}, e_{i2}, \cdots, e_{iM}, e_{i(M+1)})$. Where $(e_{i1}, e_{i2}, \cdots, e_{iM})$ is the input vector, $e_{i(M+1)}$ is the class label.

Definition 2: n_j is the j-th neuron that is an ordered pair (w_j, c_j) . w_j is the weight vector of the neuron:

Manuscript received June 5, 2006.
Manuscript revised June 25,2006.

$w_j = (w_{j1}, w_{j2}, \cdots, w_{jM}, w_{j(M+1)})$. c_j is the aggregation of leaves n_j .

The pseudo-codes described in c language about the SGNT generating are given as follows [6]:

Input: The training sample set $E = \{e_i\}, i = 1, 2, \cdots, n$;

Calculation of distance d (e_i , n_j);

Output: An SGNT generated from E.

Codes of program:

```
copy (n1,e1);     //n1 is the root node
for (i=2, j=2; i<=n; i++){
  nwin=choose(ei, ni);
  if (leaf(nwin)){
    copy (nj, nwin);
    connect (nj, nwin);
    j++;
  }
  copy (nj, ei);
  connect (nj, nwin);
  j++;
  update (nwin, ei);
}
```

In order to get the Well Organized (WO) while each new data is inserted, Li et al [5] proposed iteration learning SGNN. Iteration learning SGNN is an improved SGNN. The WO of SGNT would be changed while each node is inserted into the SGNT. At the same time, the SGNT would be VWP, HWP and merged. The SGNT has been WO as the last node is inserted.

Program codes are given as below:

```
copy (n1,e1);     //n1 is the root node
for (i=2, j=2; i<= number_of_nodes; i++){
  insert (tree,ei);
  vwp (tree);
  hwp (tree);
  merge (tree);
  j++;
}
```

Some details of function definitions that are used in above program codes can be found in literature [5-6].

3. Proposed Method

3.1 Preprocessing

We can display the dataset of time between software failures in a X-Y scatter plot, where X axis represents software failures number, Y axis represents time between software failures. Knowing from X-Y scatter plot, the total trend is that the bigger software failures number is, the longer time between software failures is. We can obtain the optimal polynomial fitting by least-square method in order to eliminate the affect of the trend. Namely for the given dataset of time between software failures $(x_i, y_i), (i = 1, 2, \cdots, n)$, we find the function relation between independent variable and dependent variable y = F(x). Supposed that software failures number x_i has dispersion $\delta_i = F(x_i) - y_i, (i = 1, 2, \cdots, n)$, and then it is required to find a polynomial $y = s^*(x)$ to make the sum of squares of dispersion minimal, namely

$$\sum_{i=1}^{n} \delta_i^2 = \sum_{i=1}^{n} [s^*(x_i) - y_i]^2 = \min \sum_{i=1}^{n} [s(x_i) - y_i]^2 \quad (1)$$

Here we choose $y = a + bx$ or $y = ax^2 + bx + c$ as regression models, then obtain the new dataset by preprocessing $y_i^* = y_i - s^*(x_i), (i = 1, 2, \cdots, n)$. The dataset y^* fluctuate around the fitting curve.

3.2 Generating an SGNT

At first, we supposed that input vector's dimension for algorithm of generating SGNT is M, prediction step length is T, then convert y^* to the matrix as follows:

$$E = \begin{bmatrix} y_{n-T-M+1}^* & \cdots & y_{n-T}^* & y_n^* \\ \vdots & \ddots & \vdots & \vdots \\ y_2^* & \cdots & y_{M+1}^* & y_{M+T+1}^* \\ y_1^* & \cdots & y_M^* & y_{M+T}^* \end{bmatrix} \quad (2)$$

Using part of E as the training sample set. Let's define $e_i = (y^*_{i1}, y^*_{i2}, \cdots, y^*_{iM}, y^*_{i(M+1)})$ as an input sample data, where M is input vector dimension, i equal to the number of input sample. Dataset of time between software failures are inserted to generating algorithm of SGNT one by one.

3.3 Prediction

As soon as generating an SGNT from the training dataset, we can use the SGNT to predict future failures. When predicting a sample, we compare the distances between M dimension vector of the sample and weight value vector of each leaf node of the SGNT, and find the minimal distance d_j , where d_j indicates that the current sample is nearest to the SGNT's the j-th leaf node. Finally output class value $\tilde{y}_j$ of the leaf could be reverted to the sample's predicting value $\hat{y}$ using the formula as follows:

$$\hat{y} = \tilde{y}_j + s^*(x) \quad (3)$$

4. Experimental Results

We use two datasets to compare proposed method and BP neural networks method [2]. The first was obtained from http://www.cse.cuhk.edu.hk/~lyu/book/reliability/DATA/CH4/SYS1.DAT; The second was obtained from http://swag.uwaterloo.ca/~rekram/reports/ece750--software-reliability-growth-modeling-and-prediction.PDF.

In experiments, parameters of BP neural networks were selected as follows: weight factor is 0.99; and the coefficient for update weight matrix between input layer and hidden layer and the coefficient for update weight matrix between hidden layer and output layer are both 0.03; and maximum iteration number is 100. The stop iteration condition of BP is that iteration number bigger than maximum iteration number. The node number of hidden layer is 3; Dimension of input vector is 2; Prediction step length is 1. Parameters of ISGNN were selected as follows: prediction step length is 1; Dimension of input vector is 2. When BP neural networks and ISGNN are set as corresponding parameter value described above, they both get the optimal results.

Table 1 and Table 2 show the experimental results for two different datasets. Prediction results of ISGNN are stable, whereas prediction results of BP are unstable. So when using BP neural networks to predict, we repeat predict for 50 times. Where training time denotes average time consumption of training; MRMSE $\pm$ std denotes mean of root mean square error $\pm$ standard deviation of root mean square error, MMAE $\pm$ std denotes mean of mean absolute error $\pm$ standard deviation of mean absolute error. MRMSE and MMAE are given respectively as follows:

$$MRMSE = \frac{\sum_{i=1}^{n_1} \sqrt{\frac{\sum_{j=1}^{n_2} (y_j - \hat{y}_{ij})^2}{n_2}}}{n_1} \quad (4)$$

$$MMAE = \frac{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} | y_j - \hat{y}_{ij} |}{n_1 \times n_2} \quad (5)$$

Where n_1 is the number of repeat prediction; n_2 is the number of testing samples; y_j is the j-th sample's real value; $\hat{y}_{ij}$ is predicted value of the j-th sample of the i-th prediction.

Table 1. Performance Evaluation of two methods with dataset 1 (fetch the first 100 data as the training dataset, the next 35 data as the testing dataset)

Criterion	BP	ISGNN
Training time (sec)	8.161	1.625
MRMSE $\pm$ std	1655.501 $\pm$ 169.916	1600.724
MMAE $\pm$ std	1303.468 $\pm$ 147.515	1191.888

Table 2. Performance Evaluation of two methods with dataset 2 (fetch the first 60 data as the training dataset, the rest 24 data as the testing dataset)

Criterion	BP	ISGNN
Training time (sec)	4.740	0.734
MRMSE $\pm$ std	24.479 $\pm$ 2.810	21.949
MMAE $\pm$ std	20.119 $\pm$ 2.477	17.157

From Table 1 and Table 2, we can clearly see that ISGNN method's training time consumptions are as about one sixth to one fifth as that of BP method, and MRSE and MAE of ISGNN method are both decreased about 3%~15% than that of BP method.

5. Conclusions

We propose a new method based on ISGNN to predict time between software failures in this paper. The method has some advantages in contrast to BP neural networks, such as requiring less parameter needed to be selected by users, and reducing much of the training time consumption. Experimental results show that ISGNN is easy to use and its training time consumption is about one sixth to one fifth of that of BP, the MAE and RMSE are both reduced about 3%~15% than that of BP method.

References

1. Karunanithi, N., Whitley, D., Malaiya, Y.K.: Using neural networks in reliability prediction. In: IEEE Software, Vol. 9(4) (1992) 53-59
2. Cai, K,Y., Cai, L., Wang, W.D., Yu, Z.Y., Zhang, D.: On the neural network approach in software reliability modeling. In: Journal of Systems and Software, Vol. 58(1) (2001) 47-62
3. Musa, J.D.: Software Reliability Engineering. In: McGraw-Hill, New York, (1998)
4. Inoue, H., Narihisa, H.: Efficient Pruning Method for Ensemble Self-Generating Neural Networks. In: Journal of Systemic, Cybernetics and Informatics, Vol. 1(6) (2003) 72-77
5. Li, A.G., Yong, H., Li, Z.H.: Iteration Learning SGNN. In: Proc of IEEE ICNN&B[C], Beijing, China, (2005)
6. Wen, W.X., Jennings, A., Liu, H.: Learning a neural tree. In: Proc of International Joint Conference on Neural Networks[C]. Beijing, China, (1992)

Aiguo Li, born in 1966, Ph.D., associate professor. His research interests include machine learning and data mining, information fusion, and software reliability engineering.