Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 36 (2009) 2116–2122
Expert Systems with Applications
www.elsevier.com/locate/eswa

Predicting software reliability with neural network ensembles

Jun Zheng

Department of Computer Science, New Mexico Institute of Mining and Technology, Socorro, NM 87801

Abstract

Software reliability is an important factor for quantitatively characterizing software quality and estimating the duration of software testing period. Traditional parametric software reliability growth models (SRGMs) such as nonhomogeneous Poisson process (NHPP) models have been successfully utilized in practical software reliability engineering. However, no single such parametric model can obtain accurate prediction for all cases. In addition to the parametric models, non-parametric models like neural network have shown to be effective alternative techniques for software reliability prediction. In this paper, we propose a non-parametric software reliability prediction system based on neural network ensembles. The effects of system architecture on the performance are investigated. The comparative studies between the proposed system with the single neural network based system and three parametric NHPP models are carried out. The experimental results demonstrate that the system predictability can be significantly improved by combing multiple neural networks.

Keywords: Software reliability; Neural network ensembles; Neural networks; Software reliability growth model (SRGM); Nonhomogeneous Poisson process (NHPP) model

1. Introduction

Software reliability, as defined by ANSI, indicates *the probability of failure-free software operation for a specified period of time in a specified environment* (Lyu, 1996; Musa, 2004). It is an important factor for quantitatively characterizing software quality and estimating the duration of software testing period. As today's software products grow rapidly in size and complexity, the prediction of software reliability plays a critical role in the software development process. Many software reliability grow models (SRGMs) have been proposed in the literature to estimate the relationship between software reliability and time and other factors. They are mainly divided into two main categories: parametric models and non-parametric models. The parametric models estimate the model parameters based on the assumptions about the nature of software faults, the stochastic behavior of the software failure process, and the development environments. The most popular parametric models are the nonhomogeneous Poisson process (NHPP) models, which have been used successfully in practical software reliability engineering (Malaiya, Li, Bieman, & Karcich, 2002; Pham, Nordmann, & Zhang, 1999). However, it has been shown that no single such model can obtain accurate predications for all cases (Li, Yin, Guo, & Lyu, 2007). On the other hand, non-parametric models like neural network and support vector machine (SVM) are more flexible which can predict reliability metrics only based on fault history without the assumptions of parametric models. Also non-parametric methods can produce models with better predictive quality than parametric models (Karunanithi, Whitley, & Malaiya, 1992a, 1992b; Su & Huang, 2007).

In this study, we use the ensemble of neural networks to build non-parametric models for software reliability prediction. It has been shown that an ensemble of multiple predictors can achieve better performance compared with a single predictor in the average (Krogh & Sollich, 1997; Wichard & Ogorzalek, 2007). Based on the study, we demonstrate that a system constructed with neural network ensemble has better prediction capability of software reliability than single neural network based system.

E-mail address: junzheng@ieee.org

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2007.12.029

The rest of this paper has been organized as follows: In Section 2, we briefly introduce some related woks in software reliability prediction. Section 3 presents the proposed method to predict software reliability with neural network ensembles. Section 4 shows the experimental results and the conclusions are drawn in Section 5.

2. Related works

In this section, we briefly introduce some related works on using neural networks in software reliability modeling and prediction.

Karunanithi et al. (1992a, 1992b) first used neural networks to predict software reliability. The feed-forward neural networks (FFNN) and recurrent neural networks including Elman neural networks and Jordan neural networks were applied for predicting the cumulative number of detected faults by using the execution time as input. The effects of various training regimes and data representation methods were also investigated in their study. Their results showed that neural networks can construct models with varying complexity and adaptability for different datasets.

In Sitte (1999), two software reliability prediction methods: neural networks and recalibration for parametric models, were compared by using common predictability measure and common datasets. The study revealed that neural networks are simpler and equal or better trend predictors.

Cai, Cai, Wang, Yu, and Zhang (2001) examined the effectiveness of the neural back-propagation network method (BPNN) for software reliability prediction. They used the multiple recent inter-failure times as input to predict the next failure time. The performance of neural network architectures with various numbers of input nodes and hidden nodes were evaluated. They concluded that the effectiveness of a neural network method highly depends on the nature of the handled datasets.

Ho, Xie, and Goh (2003) proposed a modified Elman recurrent neural network to model and predict software failures. They studied the effects of different feedback wrights in the proposed model. They also carried out a comparative study between the proposed architecture with FFNN, the Jordan recurrent model, and some parametric models.

Tian and Noore (2005a, 2005b) proposed an evolutionary neural network modeling method for software reliability prediction based on multiple-delayed-input single-output architecture. The genetic algorithm was used to on-line optimize the numbers of input nodes and hidden nodes of the neural network architecture. The neural network architecture is dynamically reconfigured when new failure time data arrives.

Su and Huang (2007) used a dynamic weighted combinational model (DWCM) for software reliability prediction based on neural network approach. Different activation functions designed from traditional SRGMs were used in the hidden layer. Their experimental results from two real software failure data sets demonstrated that the proposed DWCM method provides better predictions than traditional SRGMs.

All of the aforementioned studies only consider single neural network for software reliability prediction. However, it is known that the performance of a neural network system can be significantly improved by combining a number of neural networks (Granitto, Verdes, & Ceccatto, 2005; Hansen & Salamon, 1990). The neural network ensembles have been applied successfully in applications such as medical diagnosis (Zhou, Jiang, Yang, & Chen, 2002), financial decision (West, Dellana, & Qian, 2005), electric load forecast (Abdel-Aal, 2005), image classification (Giacinto & Roli, 2001) etc. To our best knowledge, this is the first time that neural network ensembles are used for software reliability prediction.

3. Methods

In this section, we present the proposed system for software reliability prediction based on neural network ensembles.

3.1. Software reliability data

Software may fail during the execution process. The software failure process is illustrated in Fig. 1, where t_i is the execution time for ith software failure and $\Delta t_i = t_i - t_{i-1}$ is the time interval between the $(i-1)$th and ith software failures. The software reliability data are normally arranged in pairs as $\{t_i, N_i\}$ where N_i is the accumulative number of failures in the software execution time t_i. Fig. 2 shows an example of software reliability data. The aim of software reliability prediction is to accurately estimate the number of failures in future execution time based on the software fault history.

3.2. System architecture

The prediction system based on neural network ensembles (PNNE) is shown in Fig. 3. The input of the system is the software execution time t_i. The output of the system is the predicted number of failures N'_i. The neural network ensemble consists of K component neural networks where each component neural network is a three-layer single-input single-output FFNN with n_h nodes in the hidden layer. Each component neural network is trained with different initial weights connecting three-layers. The outputs of the component neural networks are combined together

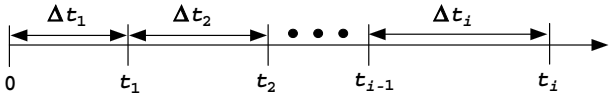

Fig. 1. Software failure process.

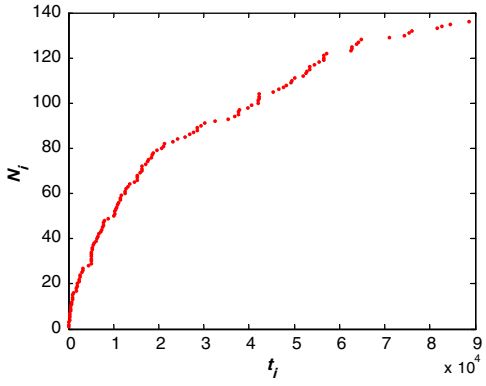

Fig. 2. An example of software reliability data.

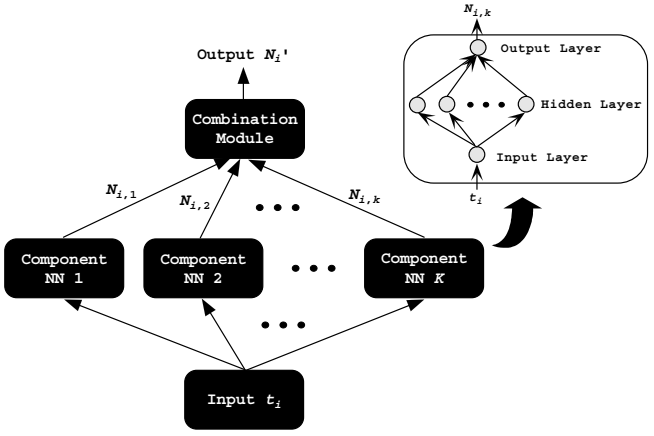

Fig. 3. The architecture of the prediction system based on neural network ensemble. Each component neural network of the ensemble is a three-layer single-input single-output FFNN.

by the combination module to produce the final output of the system.

In this study, we consider three different combination rules for the combination module:

(1) *Mean rule*: The system output is the average of the outputs of all component neural networks, which is defined as

$$N_i' = \frac{1}{K}\sum_{k=1}^{K} N_{i,k}, \tag{1}$$

where $N_{i,k}$ is the output of kth component neural network, $k = 1, 2, \ldots, K$.

(2) *Median rule*: The system output is the median of the outputs of all component neural networks, i.e. $N_i' = \text{Median}(N_{i,k})$, $k = 1, 2, \ldots, K$.

(3) *Weighted mean rule*: The system output is the weighted average of the outputs of all component neural networks, which is defined as

$$N_i' = \frac{1}{K}\sum_{k=1}^{K} w_k N_{i,k}, \tag{2}$$

where w_k is the weight assigned to kth component neural network and $\sum_{k=1}^{K} w_k = 1$, $k = 1, 2, \ldots, K$. The weight of each component neural network is set to be proportional to the inverse of the average relative error (AE) of its prediction on the training data.

3.3. Component neural network

The component neural network used in our study is a three-layer single-input single-output FFNN with n_h hidden nodes. Each component neural network has the same architecture.

A node in the FFNN is modeled as an artificial neuron as shown in Fig. 4, which computes the weighted sum of the inputs at the presence of the bias, and passes this sum through the activation function. The whole process is defined as follows:

$$\begin{aligned} v_j &= \sum_{i=1}^{n} w_{ji} x_i + \theta_j, \\ y_j &= f_j(v_j), \end{aligned} \tag{3}$$

where v_j is the linear combination of inputs $x_1, x_2, \ldots, x_n$, θ_j is the bias, w_{ji} is the connection weight between the input x_i and the neuron j, and $f_j(\cdot)$ is the activation function of the jth neuron, and y_j is the output.

We choose the most commonly used sigmoid function as the activation function, which is defined in Eq. (4)

$$f(a) = \frac{1}{1 + e^{-a}}. \tag{4}$$

Once the architecture of the FFNN is determined, the connection weights of the network have to be adjusted through a learning algorithm based on the training data. We use the Levenberg–Marquardt (LM) algorithm which is a blend of gradient descend and Gaussian–Newton algorithms to achieve fast convergence (Haykin, 1998). During the training process, the weights of the network are updated as follows:

$$\begin{aligned} W_{t+1} = W_t - (&\mathbf{J}^{\mathrm{T}}(W_t)\mathbf{J}(W_t) \\ &+ \mu_t \mathbf{I}(W_t))^{-1}\mathbf{J}^{\mathrm{T}}(W_t)\mathbf{e}(W_t), \end{aligned} \tag{5}$$

where W_{t+1} and W_t are the weight matrixes in tth and $(t+1)$th iterations, $\mathbf{J}$ is the Jacobian matrix that contains

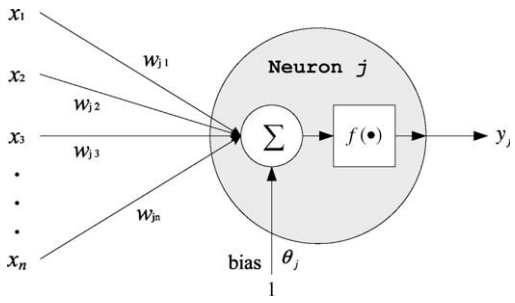

Fig. 4. A node in FFNN modeled as a neuron.

first derivatives of the network errors with respect to the weights and biases, and **e** is a vector of network errors. μ_t is the parameter that allows the LM algorithm to switch smoothly between gradient descent and Gaussian–Newton algorithms (Wilamowski & Iplikci, 2001).

To avoid the overfitting problem and improve the generalization of the neural network, the regularization method can be used which modify the error function E to be a linear sum of the mean squared error and the mean squared network weights and biases as shown in the following:

$$E = \gamma \mathrm{MSE} + (1-\gamma)\mathrm{MSW}, \tag{6}$$

where γ is the performance ratio, MSE is the mean squared error, MSW is the mean squared wrights and biases. The purpose to use regularization method is to force the network to have smaller weights and biases. Thus the network response tends to be smoother and less likely to overfit. In our study, we use the Bayesian frame proposed by Foresee and Hagan (1997) to automatically optimize the performance ratio γ during the training. The details of the Bayesian regularization can be found in Foresee and Hagan (1997) which is used in combination with LM training algorithm.

3.4. Performance measures

To compare the prediction powers of different models, some meaningful performance measures are needed to quantify the prediction accuracies. The approach adopted in our study is variable-term prediction, which is commonly used in the software reliability research community (Karunanithi et al., 1992b; Su & Huang, 2007). In this approach, only part of the failure data is used to train the model for non-parametric method or estimate the model parameters for parametric method. The trained models are then applied to the rest of the failure data. For a given execution time point t_i, the predicated failure number N'_i is compared with the actual value N_i. The performance measures for variable-term-predictability approach are the relative error (RE) and the AE which are defined as:

$$\mathrm{RE}_i = \left|\frac{N'_i - N_i}{N_i}\right| \times 100, \tag{7}$$

$$\mathrm{AE} = \frac{1}{M}\sum_{i=1}^{M}\mathrm{RE}_i, \tag{8}$$

where M is the number of data points for evaluation.

4. Experiments

Two software reliability datasets DS1 and DS2 from Lyu (1996) are used in our experiments. The dataset DS1 was collected from a real-time command and control application with 21,700 assembly instructions and 136 failures. The dataset DS2 was collected from a single-user workstation with 397 failures. The execution time and the number of failures of each dataset are normalized to the range of $[0,1]$.

We compare the performance of the PNNE system with that of prediction system based on single FFNN (PSNN) and three classical parametric NHPP models: (1) Geol and Okumoto model (G–O model): $N_t = a(1-\mathrm{e}^{-bt})$; (2) Duane model: $N_t = at^b$; (3) S-shaped model: $N_t = a(1-(1+bt)\mathrm{e}^{-bt})$. The parameters of the NHPP models are estimated by using gradient descent algorithm.

In the experiment, the systems are trained by using the fist 114 samples which cover the first 60% of the time axis for the dataset DS1. For the dataset DS2, the training data are the first 300 samples covering 33% of the time axis. For each dataset, the remaining software failure data are used for testing.

4.1. Effects of system architecture

We first investigate how the performance of the PNNE system changes with the system architecture. For each setup of the architecture, we carry out 20 runs of the experiment. Figs. 5 and 6 show the error plots of the effects of the number of hidden nodes, n_h, on the performance of the PNNE system under the three combination rules for DS1 and DS2, respectively. The number of component neural networks, K, is set to 15. From Figs. 5 and 6, we can find that the number of hidden nodes has significant effect on the performance of the PNNE system. A too small or too large n_h results in large prediction error of the

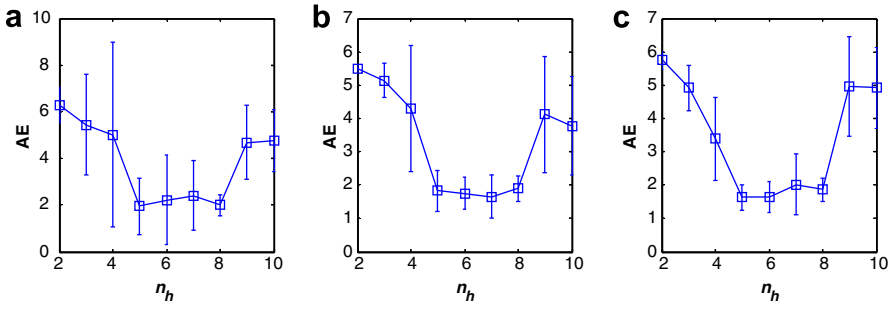

Fig. 5. Effects of the number of hidden nodes, n_h, on the performance of PNNE system for DS1: (a) mean rule, (b) median rule and (c) weighted mean rule.

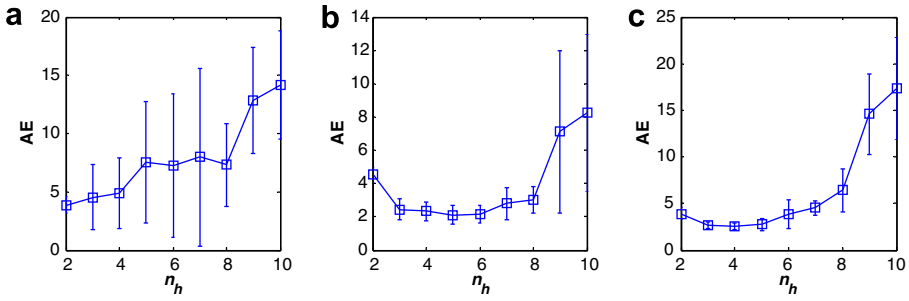

Fig. 6. Effects of the number of hidden nodes, n_h, on the performance of PNNE system for DS2: (a) mean rule, (b) median rule and (c) weighted mean rule.

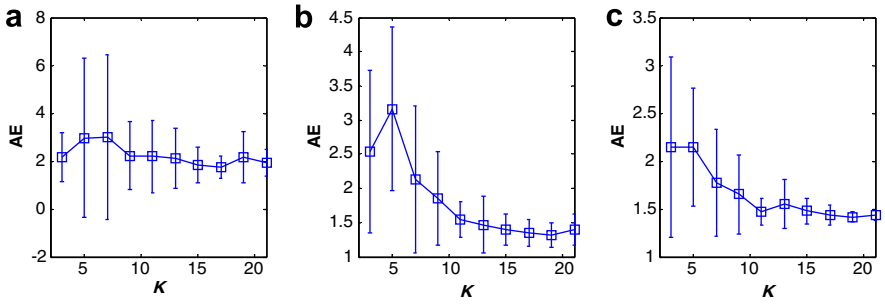

Fig. 7. Effects of the number of component neural networks, K, on the performance of PNNE system for DS1: (a) mean rule, (b) median rule and (c) weighted mean rule.

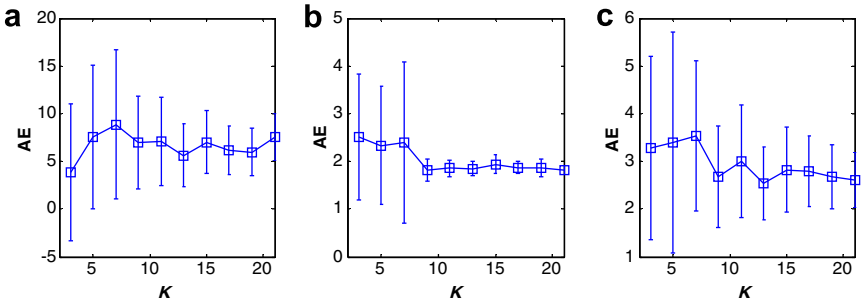

Fig. 8. Effects of the number of component neural networks, K, on the performance of PNNE system for DS2: (a) mean rule, (b) median rule and (c) weighted mean rule.

PNNE system. For the combination rules, mean rule has the worst performance while the median rule has the best performance for the two datasets.

Figs. 7 and 8 show the error plots of the effects of the number of component neural networks, K, on the performance of the PNNE system under the three combination rules, where n_h for each component neural network is set to 5. It can be observed that small K results in large variation of the system performance. As more components neural networks are involved in prediction, the system achieves better performance with smaller variation. From Figs. 7 and 8, we can also find that the median rule obtains the best performance among the three combination rules while the mean rule is the worst one.

4.2. Performance comparison

Based on the above analysis in Section 4.1, the architecture of the PNNE system used for performance comparison consists of 19 component neural networks with median combination rule and each network is a three-layer single-input single-output FFNN with five hidden nodes. For PSNN system, the number of hidden nodes is also set to 5.

Fig. 9 shows the prediction curves of PNNE system for the two datasets, respectively. The results demonstrate that the PNNE system has good prediction ability. The REs and AEs of different prediction models are shown in Fig. 10 and Table 1, respectively. The results of PNNE

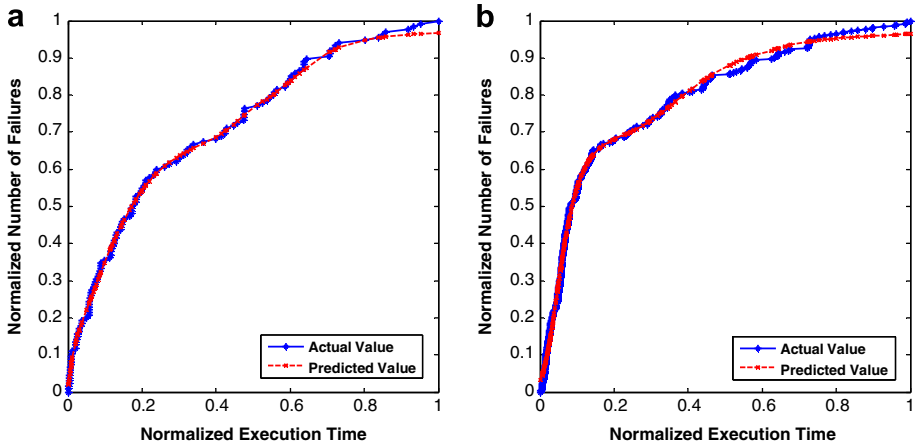

Fig. 9. The prediction results of the PNNE system for the two software reliability datasets: (a) DS1 and (b) DS2.

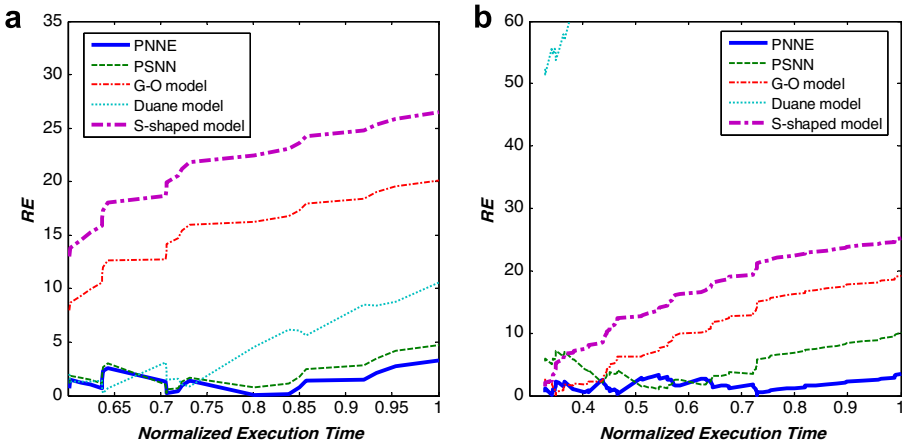

Fig. 10. The REs for different models: (a) DS1 and (b) DS2.

Table 1
AEs for different models

Dataset	PNNE	PSNN	G–O model	Duane model	S-shaped model
DS1	1.2878	1.8974	14.2851	3.5328	20.0817
DS2	1.6339	4.6528	8.7570	105.1197	14.3751

and PSNN are the averages of 20 runs. From Fig. 10 and Table 1, we can find that the two non-parametric systems have much lower prediction errors than the parametric models, which proves the capability of neural network in software reliability prediction. It is shown that no single parametric model can fit well to both two datasets. Among the three parametric models, Duane model has the best performance for DS1 but the worst performance for DS2. The best parametric model for DS2 is G–O model. It also can be observed that the PNNE system has lower prediction error than the PSNN system, which confirms our initial intention that the neural network ensemble can be a better software reliability predictor than a single neural network.

5. Conclusion

Software reliability is an important factor for quantitatively characterizing software quality and estimating the duration of software testing period. In addition to the traditional parametric SRGMs, neural network has shown to be an effective non-parametric technique for software reliability prediction. In this paper, we develop a non-parametric software reliability prediction system based on the neural network ensembles to improve the predictability by utilizing the diversity among the combined component neural networks. The experimental results demonstrate that the proposed system achieves significantly lower prediction error compared with the single neural network and traditional SRGMs, which proves that the neural network ensembles are promising for software reliability prediction.

References

Abdel-Aal, R. E. (2005). Improving electric load forecasts using network committees. *Electric Power Systems Research, 74*(1), 83–94.

Cai, K.-Y., Cai, L., Wang, W.-D., Yu, Z.-Y., & Zhang, D. (2001). On the neural network approach in software reliability modeling. *Journal of Systems and Software, 58*, 47–62.

Foresee, F. D., & Hagan, M. T., (1997). Gauss–Newton approximation to Bayesian learning. In *Proceedings of the 1997 IEEE international conference on neural networks, Houston, TX* (pp. 1930–1935).

Giacinto, G., & Roli, F. (2001). Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing, 19*, 699–707.

Granitto, P. M., Verdes, P. F., & Ceccatto, H. A. (2005). Neural network ensembles: Evaluation of aggregation algorithms. *Artificial Intelligence, 163*, 139–162.

Hansen, L. K., & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*(10), 993–1001.

Haykin, S. (1998). *Neural Networks: A comprehensive foundation* (2nd ed.). Prentice Hall.

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

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992a). Using neural networks in reliability prediction. *IEEE Software, 9*, 53–59.

Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992b). Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering, 18*, 563–574.

Krogh, A., & Sollich, P. (1997). Statistical mechanics of ensemble learning. *Physical Review E, 55*(1), 11–825.

Li, S.-M., Yin, Q., Guo, P., & Lyu, M. R. (2007). A hierarchical mixture model for software reliability prediction. *Applied Mathematics and Computation, 185*, 1120–1130.

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

Malaiya, Y. K., Li, M. N., Bieman, J. M., & Karcich, R. (2002). Software reliability growth with test coverage. *IEEE Transactions on Reliability, 51*, 420–426.

Musa, J. D. (2004). *Software reliability engineering: More reliable software, faster development and testing*. New York: McGraw-Hill.

Pham, H., Nordmann, L., & Zhang, X. M. (1999). A general imperfect software-debugging model with S-shaped fault detection rate. *IEEE Transactions on Reliability, 48*(2), 169–175.

Sitte, R. (1999). Comparison of software-reliability-growth predictions: neural networks vs parametric recalibration. *IEEE Transactions on Reliability, 48*(3), 285–291.

Su, Y.-S., & Huang, C.-Y. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software, 80*, 606–615.

Tian, L., & Noore, A. (2005a). On-line prediction of software reliability using an evolutionary connectionist model. *Journal of Systems and Software, 77*, 173–180.

Tian, L., & Noore, A. (2005b). Evolutionary neural network modeling for software cumulative failure time prediction. *Reliability Engineering and System Safety, 87*, 45–51.

West, D., Dellana, S., & Qian, J.-X. (2005). Neural network ensemble strategies for financial decision applications. *Computers and Operation Research, 32*(10), 2543–2559.

Wichard, J. D., & Ogorzalek, M. (2007). Time series predication with ensemble models applied to the CATS benchmark. *Neurocomputing, 70*, 2371–2378.

Wilamowski, B. M., & Iplikci, S., (2001). An algorithm for fast convergence in training neural networks. In *Proceedings of international joint conference on neural networks* (pp. 1778–1782).

Zhou, Z.-H., Jiang, Y., Yang, Y.-B., & Chen, S.-F. (2002). Lung cancer cell identification based on artificial neural network ensembles. *Artificial Intelligence in Medicine, 24*(1), 25–36.