Improved Neural Network based on Dynamic Predication Model of Software Reliability

Changjie Ma, Guochang Gu, Jing Zhao
College of Computer Science and Technology, Harbin Engineering University, Harbin, 150001, China
christina_pc@126.com
doi: 10.4156/jcit.vol6.issue7.43

Abstract

During the last three decades, many software reliability models have been proposed and analyzed for measuring software reliability. Those models are mathematical models that represent software failures as a random process and can be used to evaluate development status during testing. Also they proposed artificial neural network based approach for software reliability estimation and modeling. They already have shown how to apply neural network to predict a dynamic model. In this paper, we propose an improved neural network based dynamic predication model of software reliability, we improve the original model by choosing well-selected data models from all data models. Due to well-selected data models, proposed model improve the predication capability.

Keywords: Software Reliability, Artificial Neural Network, Dynamic Predication Model

1. Introduction

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The high complexity of software is the major contributing factor of Software Reliability problems. Software Reliability is not a function of time - although researchers have come up with models relating the two. The modeling technique for Software Reliability is reaching its prosperity, but before using the technique, we must carefully select the appropriate model that can best suit our case. Measurement in software is still in its infancy. No good quantitative methods have been developed to represent Software Reliability without excessive limitations. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability.

The basic principle of software reliability model is to apply the software failure time data or the testing results to determine if the reliability of a software system is growing sufficiently to meet the desired requirements. With software reliability model, software engineers can easily measure or forecast the software reliability or quality and plot the software reliability growth charts. The charts of software reliability growth depict the trends that are used to forecast software failures as a function of calendar time. Besides, the charts can also us to determine the additional time needed to meet the reliability requirements and the associated costs. SRGMs (Software Reliability Growth Models) are powerful tools to predict and assess the software reliability. Generally, SRGMs rely on certain assumptions in the nature of the software faults or the stochastic behaviors of the software failure process.

Various methods were proposed to select the best model for the specific failure data [1-4]. However, sometimes this task can be very difficult. For example, when software projects do not totally comply with the assumptions of any model; or when the project is still in the early stage and information is incomplete. Some researchers made the first step in solving this problem by combination. Lyu and Nikora proposed a linear combination method [5]. The experiment result showed that combination models tend to have more accurate short-term and long-term predictions.
than single model. Yamada et al. proposed a software error detection model which is a
nonhomogeneous Poisson process where the mean-value function has an S-shaped growth curve [6].

Neural network (NN) approaches have proven to be a universal adaptor for any non-linear continues
function with arbitrary accuracy [7, 8, 9, 10]. Many papers published in the literature address that
neural networks could offer promising approaches to software reliability estimation and modeling [6, 7,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 31, 32]. For example, Sherer [23] applied neural
networks to predict software faults in several NASA projects. She found that if software faults tend to
cluster, then identification of fault-prone modules through initial testing could guide subsequent testing
efforts that focus on these modules. Besides, Sitte [24] compared the predictive performance of two
different methods of software reliability prediction: ‘neural networks’ and ‘recalibration for parametric
models’. She found that neural networks are not only much simpler to use than the
recalibration method but that they are equal or better trend predictors. Later, Su et al. [25] show how to
apply neural network to predict software reliability by designing different elements of neural networks.
Furthermore, they use the neural network approach to build a dynamic weighted combinational model.
The applicability of proposed model is demonstrated through real software failure data sets.

In this paper, we propose an improved neural network based dynamic predication model of software
reliability, we improve the original model by choosing well-selected data models from all data models.
The proposed model have selective model framework, and use selected models to build up combination
model. To evaluate the performance of the proposed model, we take experiments on real failure data.
Due to well-selected data models, experiments result showed proposed model improve the predication
capability.

2. Preliminaries

2.1. Neural networks concepts

Let us consider a multilayer feedforward neural network. The network has one input layer, one
output layer and several hidden layers. The first layer with \( n \) neurons is the input layer including last
neuron being a bias neuron of constant output. The last layer with \( m \) neurons is the output layer. The
number of hidden layers depends on the desired learning accuracy and the training data set.

The weight matrix \( W^l \) connects layer \( l \) and layer \( l+1 \) with elements \( w_{ij}^l \). Element \( w_{ij}^l \) connects
neurons \( i \) of layer \( l \) with neurons \( j \) of layer \( l+1 \). Note that the \( W^0 \) matrix connects the input   layer and
the first hidden layer, whereas the \( W^L \) matrix connects the last hidden layer and the  output layer. We
assume only the input layer has a bias neuron, while the hidden layers and the output layer have no bias
neuron. The nonlinear activation function is denoted as \( \sigma(\cdot) \). For example, we can use the so-called
sigmoidal function:

\[
\sigma(x) = \frac{1}{1+e^{-x}}
\]

whose output is in the range of \((0,1)\), or a hyperbolic function

\[
\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]

whose output is in the range of \((-1,1)\) as an activation function.

Given a training data set \( D = \{x^i, o^i\}_{i=1}^N \), let \((x^i, o^i)\) be the \( i \)th input–output training pair, where \( x^i = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \) is the input signal vector and \( o^i = (o_1, o_2, \ldots, o_m) \in \mathbb{R}^m \) is the corresponding target
output vector. For given \( N \) sets of input–output vector pairs as examples to be learned, we can
summarize all given input vectors into a matrix \( X^o \) with \( N \) rows and \( n + 1 \) columns. Here the last
column of \( X^o \) is a bias neuron of constant value 0. Each row \( X^o \) contains the signals of one input vector.
Note $X^0 = [X \ | \ \theta]$, where matrix X consists of all signal $x_i$ as row vectors. All desired target output vectors are summarized into a matrix $O$ with $N$ rows and $m$ columns. Each row of the matrix $O$ contains the signals of one output vector $o_j$.

The described networks are of multilayer perception type: They first compute an inner product of the incoming signals matrix with their respective weight matrix. Afterwards, an activation function is applied, producing the output of the neuron which is sent to all neurons of the following layer. In this designed network structure, the activation function is not applied to the output layer, so the last layer is linear. Basically, the task of training the network means trying to find the weight matrix which minimizes the sum-square-error function:

$$E = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{m} \| g_j(x', \Theta) - o_j' \|^2$$

(3)

Where $g(x, \Theta)$ is a network mapping function and $\Theta$ is the network parameter set. $\Theta$ includes connection weight $W$ and a bias parameter. In a three-layer structure case

$$g_j(x, \Theta) = \sum_{i=1}^{N} w^l_{i,j} \sigma_i \left( \sum_{i=1}^{n} w^0_{i,j} x_i + \theta_j \right)$$

(4)

where $\theta_0$ is a bias value for the network input.

For simplification, we can write the system error function in the matrix form:

$$E = \frac{1}{2N} \text{Trace} \left[ (G - O)^T (G - O) \right]$$

(5)

Propagating the given examples through the network, multiplying the output of layer $l$ with the weights between layers $l$ and $l+1$, and applying the nonlinear activation function to all matrix elements, we get:

$$Y^{l+1} = \sigma(Y^l W^l)$$

(6)

and the network output should be

$$G = Y^L W^L$$

(7)

where we use superscript $L$ to denote the last layer. Fig.1 shows the ANN network structure.
2.2. Multi-Criteria Model Selection

In this paper, the unascertained set is applied to the process of software reliability model selection, and the software reliability model selection method based on unascertained set is presented. The using process of this method as following:

1. Suppose a set of failure data is given, some suitable models are selected according to the model’s supposeion conditions and expert’s experience [26].
2. Each model’s parameter is calculated based on failure data, and is applied to the model, and then the specific model is got. According to the specific model, the evaluation indexes of the models are calculated, including model goodness of fit, prequential likelihood, model bias, model bias trend, and model noise [27,28].
3. The single-index measure function is constructed based on unascertained set, and then the single-index measure evaluation matrix is computed. During constructing the measure function, the grade classification standard is non-fixing because of the variety of the index value. Using the following algorithm to calculate classification points [29].

For the given failure data, the index value of all models are divided into corresponding grades. Firstly, sort the single index value of all the models from small to big. Secondly, if a certain value is the maximum or the minimum, it is one classification points. Otherwise, divide the part between the maximum and the minimum value into N-2 sub-parts of equal, and N is the number of the models. Thirdly, calculate the index weight using entropy theory. Fourthly, calculate multi-index comprehensive measure matrix according to the single-index measure matrix and the index weight. Finally, compute the adaptability ranking of the models for the failure data using the confidence identification criteria, and the confidence $\lambda$ is equal to 0.6 generally.

3. The proposed structure

The proposed structure follows a selective combination framework, which is illustrated in Fig. 2. The steps are shown as follow:

Step 1. Collect failure data. Both the model selection and the neural network training will be based on these data.
Step 2. Elect candidate models. We elect models that roughly comply with the failure data as candidates.
Step 3. Select well models. In this step, system will select well models by Multi-Criteria Model Selection method.
Step 4. Combine the selected models. We combine the selected models through neural network. The details will be discussed later.

At here, we proposed another way to improve the accuracy. We elect a queue for well models, if the queue is a fixed queue, enter the step 4 directly; if not, after step 4, check out the results of test, if it is better than the threshold which we set, then it is ok, if not, return to step 3 to select well models again, at the same time, adjust condition to change well selected model queue.
4. Experimental studies and results

This section discusses the performance of the proposed software reliability prediction approach. In the following experiment, the software failure data, presented in Table 1. It is used to demonstrate the predicting performance of the proposed approach. The data contains 136 observations of the time series \((i, X_i)\) pertaining to software failure. Here \(X_i\) represents the failure space time of the software after the \(x\)th modification has been made.

![Figure 2. The framework of proposed model](image-url)
Improved Neural Network based on Dynamic Prediction Model of Software Reliability
Changjie Ma, Guochang Gu, Jing Zhao
Journal of Convergence Information Technology, Volume 6, Number 7, July 2011

Table 1. Data of software failures

<table>
<thead>
<tr>
<th>Num</th>
<th>X</th>
<th>Num</th>
<th>X</th>
<th>Num</th>
<th>X</th>
<th>Num</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>35</td>
<td>4</td>
<td>69</td>
<td>543</td>
<td>103</td>
<td>146</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>36</td>
<td>0</td>
<td>70</td>
<td>10</td>
<td>104</td>
<td>843</td>
</tr>
<tr>
<td>3</td>
<td>113</td>
<td>37</td>
<td>8</td>
<td>71</td>
<td>16</td>
<td>105</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td>38</td>
<td>227</td>
<td>72</td>
<td>529</td>
<td>106</td>
<td>261</td>
</tr>
<tr>
<td>5</td>
<td>115</td>
<td>39</td>
<td>65</td>
<td>73</td>
<td>379</td>
<td>107</td>
<td>180</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>40</td>
<td>176</td>
<td>74</td>
<td>44</td>
<td>108</td>
<td>865</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>41</td>
<td>58</td>
<td>75</td>
<td>129</td>
<td>109</td>
<td>143</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td>42</td>
<td>457</td>
<td>76</td>
<td>810</td>
<td>110</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>112</td>
<td>43</td>
<td>300</td>
<td>77</td>
<td>290</td>
<td>111</td>
<td>143</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>44</td>
<td>97</td>
<td>78</td>
<td>300</td>
<td>112</td>
<td>108</td>
</tr>
<tr>
<td>11</td>
<td>138</td>
<td>45</td>
<td>263</td>
<td>79</td>
<td>529</td>
<td>113</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>46</td>
<td>452</td>
<td>80</td>
<td>379</td>
<td>114</td>
<td>311</td>
</tr>
<tr>
<td>13</td>
<td>77</td>
<td>47</td>
<td>255</td>
<td>81</td>
<td>44</td>
<td>115</td>
<td>124</td>
</tr>
<tr>
<td>14</td>
<td>138</td>
<td>48</td>
<td>197</td>
<td>82</td>
<td>129</td>
<td>116</td>
<td>943</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>49</td>
<td>193</td>
<td>83</td>
<td>810</td>
<td>117</td>
<td>700</td>
</tr>
<tr>
<td>16</td>
<td>77</td>
<td>50</td>
<td>6</td>
<td>84</td>
<td>290</td>
<td>118</td>
<td>875</td>
</tr>
<tr>
<td>17</td>
<td>24</td>
<td>51</td>
<td>79</td>
<td>85</td>
<td>300</td>
<td>119</td>
<td>245</td>
</tr>
<tr>
<td>18</td>
<td>108</td>
<td>52</td>
<td>816</td>
<td>86</td>
<td>529</td>
<td>120</td>
<td>729</td>
</tr>
<tr>
<td>19</td>
<td>88</td>
<td>53</td>
<td>1351</td>
<td>87</td>
<td>281</td>
<td>121</td>
<td>187</td>
</tr>
<tr>
<td>20</td>
<td>670</td>
<td>54</td>
<td>148</td>
<td>88</td>
<td>160</td>
<td>122</td>
<td>447</td>
</tr>
<tr>
<td>21</td>
<td>120</td>
<td>55</td>
<td>21</td>
<td>89</td>
<td>828</td>
<td>123</td>
<td>386</td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>56</td>
<td>233</td>
<td>90</td>
<td>101</td>
<td>124</td>
<td>446</td>
</tr>
<tr>
<td>23</td>
<td>114</td>
<td>57</td>
<td>134</td>
<td>91</td>
<td>445</td>
<td>125</td>
<td>122</td>
</tr>
<tr>
<td>24</td>
<td>325</td>
<td>58</td>
<td>357</td>
<td>92</td>
<td>296</td>
<td>126</td>
<td>990</td>
</tr>
<tr>
<td>25</td>
<td>55</td>
<td>59</td>
<td>193</td>
<td>93</td>
<td>175</td>
<td>127</td>
<td>948</td>
</tr>
<tr>
<td>26</td>
<td>242</td>
<td>60</td>
<td>236</td>
<td>94</td>
<td>106</td>
<td>128</td>
<td>108</td>
</tr>
<tr>
<td>27</td>
<td>68</td>
<td>61</td>
<td>31</td>
<td>95</td>
<td>178</td>
<td>129</td>
<td>22</td>
</tr>
<tr>
<td>28</td>
<td>422</td>
<td>62</td>
<td>369</td>
<td>96</td>
<td>860</td>
<td>130</td>
<td>75</td>
</tr>
<tr>
<td>29</td>
<td>180</td>
<td>63</td>
<td>748</td>
<td>97</td>
<td>983</td>
<td>131</td>
<td>482</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>64</td>
<td>0</td>
<td>98</td>
<td>707</td>
<td>132</td>
<td>550</td>
</tr>
<tr>
<td>31</td>
<td>114</td>
<td>65</td>
<td>232</td>
<td>99</td>
<td>33</td>
<td>133</td>
<td>100</td>
</tr>
<tr>
<td>32</td>
<td>600</td>
<td>66</td>
<td>330</td>
<td>100</td>
<td>868</td>
<td>134</td>
<td>10</td>
</tr>
<tr>
<td>33</td>
<td>15</td>
<td>67</td>
<td>365</td>
<td>101</td>
<td>724</td>
<td>135</td>
<td>107</td>
</tr>
<tr>
<td>34</td>
<td>36</td>
<td>68</td>
<td>1222</td>
<td>102</td>
<td>232</td>
<td>136</td>
<td>371</td>
</tr>
</tbody>
</table>

In our proposed model, the main measure used for evaluating model performance is the Mean Absolute Relative Error (MARE), Mean Relative Error (MRE) and Mean average percentage error (MAPE). MARE is the preferred error measure for failure space time data of software system and is calculated as follows:

\[
MARE = \left( \frac{1}{M} \sum_{i=1}^{M} \left| \frac{R' - R}{R} \right| \right) \times 100\%
\]

MRE is calculated as follows:

\[
MRE = \left( \frac{1}{M} \sum_{i=1}^{M} \left| \frac{R' - R}{R} \right| \right) \times 100\%
\]

MAPE is calculated as follows:

\[
MAPE = \frac{1}{M} \sum_{i=1}^{M} \left( \frac{R' - R}{R} \right) \times 100\%
\]
Where $M$ represents the number of the test samples, $R_r$ is the actual failure time of the $r$th failure, $R'_r$ represents the estimated failure time of the $r$th failure.

![Figure 3. Predication comparison of the proposed model and other models](image)

**Table 2.** Predication performance comparison

<table>
<thead>
<tr>
<th>Model</th>
<th>MARE</th>
<th>MAE</th>
<th>MAPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The proposed model</td>
<td>0.0149</td>
<td>0.0134</td>
<td>0.0872</td>
</tr>
<tr>
<td>Lyu-Nikora [5]</td>
<td>0.0372</td>
<td>0.0727</td>
<td>0.1189</td>
</tr>
<tr>
<td>Yamada-Ohba-Osaki [30]</td>
<td>0.0585</td>
<td>0.1538</td>
<td>0.4142</td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper, we propose an improved neural network dynamic predication model for software reliability evaluation. We have shown the improved structure adopt well selected data model which selected from all data models to build up combination model. To evaluate the performance of the proposed model, we compare the proposed model with other models in figure 3 and table 2, experiments result showed proposed model improve the predication capability.

6. References


