A Software Reliability Prediction Model

Using Improved Long Short Term Memory Network

Fu Yangzhen, Zhang Hong, Zeng Chenchen and Feng Chao
Science and Technology on Reliability and Environmental Engineering Laboratory
School of Reliability and Systems Engineering, Beihang University
Beijing, P.R. China
fuyangzhen@buaa.edu.cn, zh@buaa.edu.cn, zengcengceng@foxmail.com, fengchao9286@126.com

***Abstract*—With the development of software reliability research and machine learning, many machine learning models have been used in software reliability prediction. A long short term memory network (LSTM) modeling approach for software reliability prediction is proposed. Profit from its particular data flow control structure, the model overcomes the vanishing and exploding sensitivity of simple recursive neural network for software reliability prediction. Proposed approach also combines with layer normalization and truncate back propagation. To some extent, these two methods promote the effect of the proposed model. Compared with the simple recursive neural network, numerical results show that our proposed approach has a better performance and robustness with respect to software reliability prediction.**

Keywords—software reliability prediction; long short term memory network; vanishing and exploding sensitivity

I. INTRODUCTION

Software failure data is the basis of software reliability estimation. For evaluate software reliability, the researchers usually use failure-count data and time-between failures to collect failure data. Failure data is the basis for software reliability assessment and prediction.

This paper presents an improved long short term memory network model, belonging to a kind of recurrent neural network (RNN). After the RNN model was first used by Karunanidhi and Darrell in the field of reliability prediction [1], they were also the first research team to introduce neural network into the field of software reliability prediction, scholars have invented many variants to use. But the bottleneck of the current performance improvement is still the problem of gradient disappearance.

II. IMPROVED LONG SHORT TERM MEMORY NETWORK

Training deep network always based on backward propagation random gradient descent method. With the increase in the number of layers, the lower level of the gradient exponent can not accept the effective training signal. The neural network is limited by the problem of unstable gradients. If the network uses the sigmoid activation function, then the gradient of the previous layer will vanish exponentially.

Long short term memory network, referred to as LSTM, is a kind of special RNN, has the ability to learn long-term dependencies. LSTM was proposed by Hochreiter & Schmidhuber [2], and many researchers carried out a series of work to improve and make it flourish. LSTM can avoid the problem of gradient instability because of its special design of the architecture. LSTM has the ability to add or remove information about the cell state, which is controlled by a structure called gate. The gate is a way of selectively passing information. The forget gate inputs h_{t-1} and x_t, and outputs a number between 0 and 1 for each element in cell state C_{t-1}. Value 1 means completely retain the information, value 0 means completely discard the information:

$$\begin{cases} f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \\ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t = o_t * \tanh(C_t) \end{cases} \tag{1}$$

Where f_t is the output of forget gate, σ means sigmoid function, W and b are the weights and biases need to be calculated. The next step is to decide which new information we will store in the cell state. i_t ,the information we want to update. $\tilde{C}_t$, which may be added to the cell state. $i_t * \tilde{C}_t$ is made up of the new candidate value $\tilde{C}_t$ multiplied by the candidate value of update degree i_t we decided in each state. o_t is the information we output each state.

Back propagation (BPTT) through time is too sensitive to recent distractions. In a feedforward neural network, exponential vanish means that changes in weight in the early neural layers will be much less than those in late neurons. Williams et al. proposed the truncated back-propagation [3]. This operation can overcome a series of questions complete BPTT brings when training model.

Inspired by batch normalization, Lei Ba et al. proposed an RNN performance optimization method - layer normalization [4] which can reduce RNN network training time and get better overall performance.

978-1-5386-2072-4/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS-C.2017.115

III. PERFORMANCE VALIDATION

The sample input sequence and the corresponding desired output sequence are defined as follows, x_t is the time-between failures in the training data sequence, and t is the order index:

Input Sequence: $x_0, x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots$

Output Sequence: $x_1, x_2, \ldots, x_t, x_{t+1}, x_{t+2}, \ldots$

We use the normalization method L. H. Tsoukalas et al. recommended to minimize the impact of the absolute size [5], the specific formula is as follows and the second formula can scale back the output. y is the scaled value we feed into our network, x is the actual value of scale:

$$\begin{cases} y = \dfrac{0.8x}{x_{\max} - x_{\min}} + (0.9 - \dfrac{0.8x_{\max}}{x_{\max} - x_{\min}}) \\ x = \dfrac{y - 0.9}{0.8}(x_{\max} - x_{\min}) + x_{\max} \end{cases} \quad (2)$$

Four datasets were used to test the performance of our proposed model. They are real-time control application and flight dynamic application data sets, SYS1: From Musa et al [6]. DATA-11, DATA-12, DATA-13: From Park et al [7].

When the model is trained, its self-evaluation function is called loss function. $\hat{x}_i$ is the output prediction of input x_i, we use mean squared error (MSE) as model's loss function. Two commonly used measures in the software reliability research community are goodness-of-fit and next-step predictability [8]. We need to use the relative error (RE) to compare the difference between the actual and predicted values of the cumulative failure time. And we calculate the average relative error (AE) on test sets to do the horizontal comparison:

$$\begin{cases} MSE = \dfrac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2 \\ RE = \left|\dfrac{\hat{x}_t - x_t}{x_t}\right| \\ AE = \dfrac{1}{n}\sum_{i=1}^{n}\left|\dfrac{\hat{x}_i - x_i}{x_i}\right| \times 100 \end{cases} \quad (3)$$

The validation experiment selected 85% prior data as train sets, and last 15% as test sets. We summarize the results of the model's horizontal comparisons, where the models that participate in the comparison are from Park et al. [7], Karunanithi et al. [8] and LIANG TIAN et al. proposed a RNN model approach with bayesian regularization [9]. Park et al. applied failure sequence as input and cumulative failure time as output in the feed-forward neural network (FFN). Karunanithi et al. applied input-output learning pair of cumulative execution and a corresponding number of accumulated defects in both feed-forward neural network (FFN) and recurrent neural network (RNN). The comparison shows in table 2.

TABLE I. PERFORMANCE RESULT

Data Sets	Goodness-of-fit (RE ≤ 5%)	Next-Step-Predictability (RE ≤ 5%)
SYS1	65.23%	89.0%
DATA-11	70.3%	92.3%
DATA-12	73.6%	97.3%
DATA-13	89.32%	100%

TABLE II. AVERAGE RELATIVE PREDICTION ERROR (%)

Data Sets	Proposed improved LSTM	FFN [7]	RNN [8]	RNN+BR [9]
SYS1	2.01	2.58	2.05	1.88
DATA-11	1.87	3.32	2.97	2.10
DATA-12	1.15	2.38	3.64	1.20
DATA-13	0.85	1.51	2.28	0.99

IV. CONCLUSIONS

This paper proposed a software reliability prediction model based on long short term memory network and truncated back propagation and layer normalization were added in our model to improve the performance. Compared with other neural network models, proposed approach have better predictive performance and robustness. Our further step will mainly concentrate in how to further enhance the software reliability prediction accuracy and compare with traditional software reliability prediction methods.

REFERENCES

[1] Karunanithi N, Whitley D, Malaiya Y K. Using neural network in reliability prediction[J]. IEEE Software, 1992, 9(4): 53-59.

[2] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[3] Williams R J, Zipser D. Gradient-based learning algorithms for recurrent networks and their computational complexity[J]. Backpropagation: Theory, architectures, and applications, 1995, 1: 433-486.

[4] Ba J L, Kiros J R, Hinton G E. Layer normalization[J]. arXiv preprint arXiv:1607.06450, 2016.

[5] L. H. Tsoukalas and R. E. Uhrig, Fuzzy and Neural Approaches in Engineering (John Wiley & Sons, New York, 1996), pp. 385-405.

[6] J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill Series in Software Engineering and Technology. (McGraw-Hill, 1987).

[7] J. Y. Park, S. U. Lee and J. H. Park, Neural network modeling for software reliability prediction from failure time data, J. Electrical Engineering and Information Science 4(4) (1999) 533-538.

[8] N. Karunanithi, D. Whitley and Y. K. Malaiya, Prediction of software reliability using connectionist models, IEEE Trans. Software Engineering 18(7)(1992) 563-574.

[9] Tian L, Noore A. Software reliability prediction using recurrent neural network with Bayesian regularization[J]. International Journal of Neural Systems, 2004, 14(03): 165-174.