Volume: 09 Issue: 03 Pages: 3455-3460 (2017) ISSN: 0975-0290

Robust Fault-Tolerant Training Strategy Using Neural Network to Perform Functional Testing of Software

Manas Kumar Yogi
Department of Computer Science, Pragati Engineering College, Kakinada City, India
Email: manas.yogi@gmail.com
L. Yamuna
Department of Computer Science, Pragati Engineering College, Kakinada City, India
Email: yamuna.lakkamsani@gmail.com

----------**ABSTRACT**----------

This paper is intended to introduce an efficient as well as robust training mechanism for a neural network which can be used for testing the functionality of software. The traditional setup of neural network architecture is used constituting the two phases -training phase and evaluation phase. The input test cases are to be trained in first phase and consequently they behave like normal test cases to predict the output as untrained test cases. The test oracle measures the deviation between the outputs of untrained test cases with trained test cases and authorizes a final decision. Our framework can be applied to systems where number of test cases outnumbers the functionalities or the system under test is too complex. It can also be applied to the test case development when the modules of a system become tedious after modification.

Keywords - **ATNN, Fault, Neural, Test Case, Test Oracle**

Date of Submission: Oct 23, 2017 Date of Acceptance: Dec 01, 2017

I. INTRODUCTION

In software testing what matters most is how much application conforms to specifications. In practice, agreement documents are indicators of the level to be accepted up to which the required functionality can be achieved. Software testing consumes substantial quantity of time as well as effort, so strategies have to be developed to carry out the functionality testing in a manner which is efficient in terms to deliver quality software with minimum effort & time. In past, Artificial Neural Networks (ANNs) were used to handle aspects of testing. ANNs are developed to mimic the structure and information processing powers of the human brain. The architectural components of a neural network are units same as the neurons of the brain. A neural network is formed from one or more layers of these neurons, the interconnections of which have associated synaptic weights. Each neuron in the network is able to perform calculations that contribute to the overall learning process, or training of the network. The neuron interconnections are associated with synaptic weights that store the information computed during the training of the network. It is rightly said that the neural network is a massive parallel information processing system which uses the distributed control to learn and store knowledge about its environment. Clearly, the two crucial factors that affect the superior computational capability of the neural network are its distributed design working in parallel layers and its ability to extrapolate the learned information to yield outputs for inputs not presented during training phase. These properties of the neural network allow multiple complex problems to be solved.

Data mining, pattern recognition, and function approximation are some of the tasks that can be handled by neural networks. In this paper, a design of Artificial Testing Neural Network (ATNN) is proposed to train on a suite of test cases developed manually. Sometimes manual test cases are found to have a greater degree of fault finding ability and this efficient element of manual test cases are used to train the test cases on a ATNN. The result is a set of superior or trained test cases which have the ability to find a fault in the functionality of the application in minimum time. If this approach is repeated over time, these trained test cases can show better fault finding ability over other programs under test.

II. EXISTING WORK

Domain based testing models already exist which predicts faults taking into account fault exposing metrics which are traditional in nature. Tools like SLEUTH use this model for purpose of effective test suite generation with the help of test case metrics, a synthetic test oracle judge's individual test case for error classification. The neural network is imparted training on test metric input sequence and maps them to the test oracle's error classification system. Once trained, the network acts as a test case effectiveness predictor. The metrics used for the experiment were loosely based on coverage metrics for Domain Based Testing. In real testing environment, the set of metrics needed for an arbitrary testing criterion is not known well in advance. This rises a huge challenge of selecting a dynamic approach for finalizing test case metrics.

The results from training four networks showed how well each network predicted individual fault severities. The no

Volume: 09 Issue: 03 Pages: 3455-3460 (2017) ISSN: 0975-0290

error net predicted the best with 94.4%, and the second-best predictor was the most severe fault net, with 91.7%. placed The incorrectly classified tests were placed into one of three categories: False Positive, False Negative, and Other Incorrect. A False Positive response was recorded for severity 1- 3 errors when the network predicted a fault that doesn't truly exist. For no error severity, a false positive meant that the neural network predicted that there was not an error, when indeed the test case would have uncovered one. For other severity classes, a False Negative response was recorded when the neural net predicted severity of no fault exposed when the test case indicates a fault. Other Incorrect represented to tests that were classified by the neural net as exposing a fault, but of the incorrect type. We have used this information to analyze three test data generation objectives.

Objective 1: Minimize Number of Test Cases
Objective 2: Widen scope of Severe severity classes
Objective 3: Reduce error rate during training

III. PROPOSED MECHANISM

2.1 Phases of Proposed Mechanism

The proposed mechanism consists of two phases - a training phase and evaluation phase.

3.1.1 Phase 1

Hybrid Training Mechanism - Training Phase (Construction of Trained Test Cases)

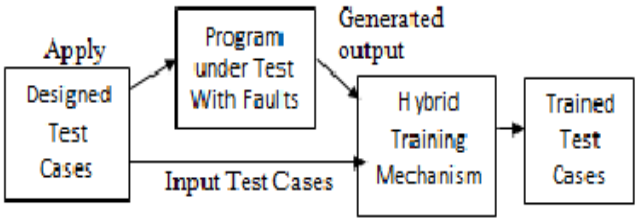

Figure.1. Mechanism of training phase

Hybrid training mechanism is resident on an ATNN (Artificial Testing Neural Network). The ATNN is modelled after a Feed-Forward Neural Network which has two layers namely, Hidden Layer and Output Layer/Visible Layer. In this work, only two layers are considered, i.e., input layer and output layer. If we denote $y_i^{(l)}$ as the output of the i^{th} test case of input layer l, the function of the network is represented as:

$$y_i^l = f\left[\sum w_{ij}^{(l,l-1)} y_j^{(l-1)} + \emptyset_i^{(l)}\right], \text{ where } l=1,2,\ldots..n \quad (1)$$

Where $w_{ij}^{(l,l-1)}$ represent the weight from test case 'j' of layer 'l-1' to the test case 'i' of layer 'l'. $\emptyset_i^{(l)}$ is the threshold of test case 'i' of layer 'l'.

The decision factor is considering the number of test cases in layer 'l'. The weight of test case is indicated by the fault detection ability of a test case. We limit the weight value of a test case between 0 and 1.

With help of above principles, we present a hybrid training mechanism for a test suite T_s. After application of a learning algorithm on a test suite T_s, we get what we term it as Trained Test Cases.

Let x(1),x(2),…x(m) be given input vectors containing test case input values and y(1),y(2)…y(m) be the corresponding desired output vectors. We apply the back propagation algorithm as our basis, so as to adjust the weights and threshold of the test cases.
We calculate the sum of square error:

$$E = \sum_{m=1}^{M} E(m) \quad (2)$$

$$E(m) = \|y(m) - y^{(L)}\|^2,$$

Where $y^{(L)}$ indicates vector of outputs of the network, when input is x(m). We repeat the adjustments of weights, so that the network maps each x(m) to y(m) as close as possible. When this situation appears, we say the test cases x(1), x(2) …..x(m) are now trained. The threshold is decided based on overall testing time available.

3.1.2 Phase 2

Evaluation Phase (Decision Making based on nature of output)

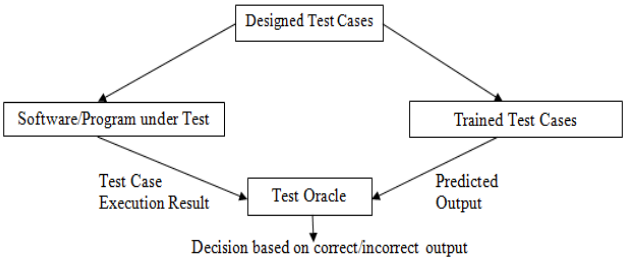

Figure.2. Mechanism of evaluation phase

In this stage, the designed test cases of a test suite T_s are applied under the Program under Test (PUT) and results are given as input to the Test Oracle. In an alternate procedure conducted parallel to the above mentioned procedure, the trained test cases generate the predicted outputs which are fed to the Test Oracle. Finally, the Oracle decides the functionality of the program based on the outcomes; the Test Oracle has comparison ability of trained test case output with outputs of designed test cases.

3.2 Advantages of proposed training mechanism

The following are the advantages of proposed work-

i. The trained test cases can act as a Test Oracle itself for next phase of test process like regression testing.
ii. For a complex program to be tested, the test cases can be designed in such a manner that during regression testing only the test cases which are to be executed on the modified version of the program under test (PUT) are compared against the corresponding trained test cases for decisive outcome. In this way, large amount of test cases

Volume: 09 Issue: 03 Pages: 3455-3460 (2017) ISSN: 0975-0290

won't be re-executed, thus saving testing time and effort to appreciate level.

iii. The Test Oracle we employ is unbiased unlike human testers. Human testers may be biased due to prior knowledge of the program. The ATNN contains layers of test cases with specific weights.

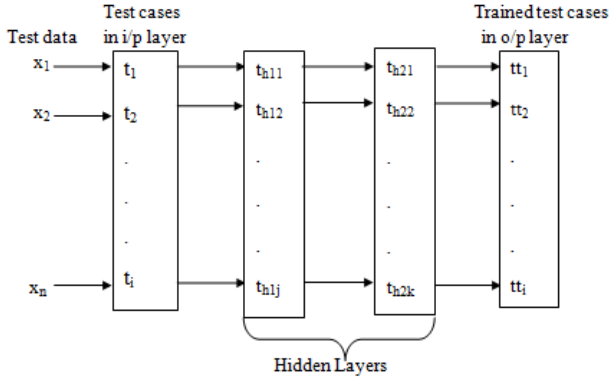

Figure.3. Architectural design for ATNN

Here, t_{h11} …t_{h21} indicates the test cases in hidden layers. These test cases have enhanced weight values and by application of different test data, i.e., x_1, x_2….x_n, their weights change. For example, consider a test case for checking password field. The validation rule is at least 6 characters and 2 of them should be special symbols. The input test case, say 't' will have all 6 characters as special symbols, so this test case will fail due to high deviation of validation rule, thus giving it a very high weight say 'w_h', where h=higher.

In second layer, i.e., the first hidden layer, the test case t_{h1} should be designed with even higher weight. So, we give test data input as all 6 characters blank. This will give it weight say $w_{vh,}$ where vh=very high.

In third layer, i.e., second hidden layer, we design the test case with 1 character as special and rest is general characters. This test case does not deviate much from the test validation rule, so we assign this test case with weight say $w_{m,}$ where m=medium.

4. EXPERIMENTAL RESULTS

We used a credit card approval system to verify the effectiveness of our proposed mechanism. The process that the experiment follows begins with the generation of test cases. The input attributes are created using the specification of the program that is being tested, while the outputs are generated by executing the tested program. The data undergo a preprocessing procedure in which all continuous input attributes are normalized (the range is determined by finding the maximum and minimum values for each attribute), and the binary inputs and outputs are either assigned a value of 0 or 1. The continuous output is treated in a different manner, and the output of each example is placed into the correct interval specified by the range of possible values and the number of intervals used. The processed data are used as the data set for training the neural network. The network parameters are determined before the training algorithm begins. The training of the network includes presenting the entire data set for one epoch, and the number of epochs for training is also specified. The back propagation training algorithm concludes when the maximum number of epochs has been reached or the minimum error rate has been achieved. The network is then used as an "oracle" to predict the correct outputs for the subsequent regression tests.

Table I. Input attributes of the data

Attribute name	**type**	**Attribute type**	**details**
Aadhar id	integer	Input	unique
Citizenship	integer	Input	0-indian 1-others
State	integer	Input	0-29
Age	integer	Input	1-100
Sex	integer	Input	0: Female 1: Male
Region	integer	Input	0–6 for different regions in India
Income class	integer	Input	0 if income p.a. < Rs.10k 1. if income p.a. ≥ Rs 10k 2 if income p.a. ≥ Rs 25k 3 if income p.a. ≥ $50k
Number of dependents	integer	Input	1–4
Marital status	integer	Input	0: Single 1: Married
Credit approved	integer	Output	0: No 1: Yes
Credit amount	integer	Output	≥ 0

Volume: 09 Issue: 03 Pages: 3455-3460 (2017) ISSN: 0975-0290

Table II. Sample data used during training (before preprocessing)

Aadhar id	Citizenship	State	Age	Sex	Region	Income class	Number of dependents	Marital status	Credit approved	Credit amount
1	1	1	23	1	1	1	1	1	0	0
2	1	12	45	1	3	1	1	0	0	0
3	1	22	65	0	4	1	0	0	1	10000
4	1	11	34	0	6	1	0	1	0	0
5	1	5	26	0	2	2	2	1	1	20000
6	1	7	28	1	2	0	1	0	0	0
7	0	7	41	1	5	2	2	0	1	10000
8	0	8	55	1	6	2	2	0	0	0
9	1	19	58	1	4	2	3	1	1	20000

In this experimental setup in MATLAB 2016 , we used eight input units for the eight relevant input attributes (the first is not used, as it is a descriptor for the example), and twelve output computational units for the output attributes. The first two output units are used for the binary output. For training purposes, the unit with the higher output value is said to be the “winner”. The remaining ten units are used for the continuous output. The initial synaptic weights of the neural network are obtained randomly and covered a range between –0.5 and 0.5. Experimenting with the neural network and the training data, we concluded that one hidden layer with twenty-four units was sufficient for the neural network to approximate the original application to within a reasonable accuracy. A learning rate of 0.50 was used, and the network required 1,200 epochs to produce a 0.2 percent misclassification rate on the binary output and 5.38 percent for the continuous output. The minimum error rate for the continuous output (low threshold = 0.10, high threshold = 0.90).

Volume: 09 Issue: 03 Pages: 3455-3460 (2017) ISSN: 0975-0290

Table III. The minimum error rate for the continuous output (low threshold = 0.10, high threshold = 0.9

Injected fault number	**Number of correct outputs**	**Number of incorrect outputs**	**Percentage of correct outputs classified as being incorrect (%)**	**Percentage of incorrect outputs classified as being correct (%)**
2	140	860	28.14	1.43
3	307	693	6.78	49.51
4	587	413	5.33	21.12
5	822	178	8.99	3.89
6	69	931	23.63	13.04
7	559	441	11.11	5.72
8	355	645	21.86	7.89
9	217	783	8.17	73.27
10	303	697	7.60	52.15
11	238	762	7.74	66.39
12	276	724	24.17	10.51
13	371	629	23.05	6.47
14	99	901	22.86	23.23
15	65	935	23.32	33.85
16	407	593	23.27	4.91
17	273	727	22.56	13.55
18	20	980	24.49	50.00
19	71	929	24.54	1.41
20	1000	0	0.00	4.20
21	125	875	20.91	50.40
Percentage average			**16.93**	**24.65**
Total average			**20.79**	

Volume: 09 Issue: 03 Pages: 3455-3460 (2017) ISSN: 0975-0290

The last table summarizes the results for the minimum error rate of the continuous output (credit amount.) The tables include the injected fault number, the number of correct outputs and incorrect outputs as determined by the "oracle," and the percentages for the correct outputs classified as being incorrect and incorrect outputs classified as being correct. The percentages were obtained by comparing the classification of the "oracle" with that of the original version of the application. The original version is assumed to be fault-free, and is used as a control to evaluate the results of the comparison tool. The best thresholds were selected to minimize the overall average of two error rates. Due to the increased complexity involved in evaluating the continuous output, there is a significant change in the capability of the neural network to distinguish between the correct and the faulty test cases: the minimum average error of 8.31 achieved for the binary output versus the minimum average error of 20.79 for the continuous output. An attempt to vary the threshold values also did not result in an evident change to the overall average percentage of error for the continuous output.

3. CONCLUSION

The main aim of this paper was to put forward a new mechanism to test the functionality of a complex system using the principles deployed in field of neural networks. An Artificial Testing Neural Network was used to train the way manual developed test cases work and to improve the fault detecting ability of the trained test cases we used a test oracle. In future we intend to apply the proposed mechanism to test a application system which already has a test case suit. The future of testing in automation is ushering into a new era with usage of neural networks in software testing which will bring about a revolution in the way automation
software testing is being done now. The researchers in this field are working diligently to evolve hybrid techniques like we proposed in this paper to save considerable amount of testing effort and time.

REFERENCES

[1] Anderson C, von Mayrhauser A, Mraz R. On the use of neural networks to guide software testing activities. In: Proceedings of ITC'95, the International Test Conference; October 21–26, 1995.

[2] Choi J, Choi B. Test agent system design. In: 1999 IEEE International Fuzzy Systems Conference Proceedings; August 22–25, 1999.

[3] Khoshgoftaar TM, Allen EB,Hudepohl JP,Aud SJ. Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks 1997;8(4):902–909.

[4] Khoshgoftaar TM, Szabo RM. Using neural networks to predict software faults during testing. IEEE Transactions on Reliability 1996;45(3):456–462.

[5] J. Yue, C. Bojan, M. Tim, and L. Jie, "Incremental development of fault prediction models," International Journal of Software Engineering and Knowledge Engineering, vol. 23, no. 10, pp. 1399–1425, 2013.

[6] Masoud Ahmadzade, Davood Khosroanjom, Toufiq Khezri, Yusef Sofi, " Test Adequacy Criteria for UML Design Models Based on a Fuzzy - AHP Approach ", American Journal of Scientific Research ISSN 1450-223X Issue 42(2012), pp. 72-84.

[7] Park, J., Baik, J.: Improving software reliability prediction through multi-criteria based dynamic model selection and combination. J. Syst. Softw. 101, 236–244 (2015).

[8] Chang, P.T., Lin, K.P., Pai, P.F.: Hybrid learning fuzzy neural models in forecasting engine system reliability. In: Proceeding of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference, pp. 2361–2366 (2004).

[9] Noekhah, S., Hozhabri, A.A., Rizi, H.S.: Software reliability prediction model based on ICA algorithm and MLP neural network. In: 7th International Conference on e-Commerce in Developing Countries: With Focus on e-Security (ECDC), pp. 1–15. IEEE, April 2013.

[10] BEWOOR, L. A. et al. Predicting Root Cause Analysis (RCA) bucket for software defects through Artificial Neural Network. Imperial Journal of Interdisciplinary Research, [S.l.], v. 3, n. 6, june 2017. ISSN 2454-1362.

[11] Ruilian zhao, shanshan lv, "Neural network based test cases generation using genetic algorithm" 13th IEEE international symposium on Pacific Rim dependable computing. IEEE, 2007, pp.97 - 100.

[12] ZhiweiXu,KehanGao,Taghi,M.Khoshgoftaar,Naeem Seliya,System regression test planning with a fuzzy expert system,Information Sciences Volume 259, 20 February 2014, Pages 532-543.

[13] Bhatnagar R, Bhattacharjee V, Ghose MK (2010) Software development effort estimation—neural network vs regression modeling approach. Int J Eng Sci Technol 2(7): 2950–2956.