http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (3): 469-474, 2007
ISSN 1812-5638

Oracle Model Based on RBF Neural Networks for Automated Software Testing

[1]Ying Lu and [2]Mao Ye
[1]Computer Science and Engineering College, Xi'an Technological University, Xi'an 710032, China
[1]School of Electronic and Information Engineering, Xi'an Jiaotong University,
Xi'an 710049, China

Abstract: Oracle is one of the most difficult and expensive parts in automated software testing. It is explored in this paper to use Radial Basis Function Neural Networks (RBF NN) to construct an automated oracle model. The automated oracle generate the approximate outputs that are close to expected outputs after training. Actual outputs are then compared with the approximate outputs to determine if there is a failure when software is running. Oracle can therefore be implemented automatically and the precision be adjusted by parameters. It will save a lot of time and cost in software testing.

Key words: Oracle, neural networks, radial basis function (RBF), software testing

INTRODUCTION

Oracle includes two processes which are to generate expected outputs and to compare actual outputs with them (Memon *et al*., 2003). Oracle is generally implemented manually. It is important to automate the oracle in automated software testing in which these two processes are automatically implemented. A perfect automated oracle would be behaviorally equivalent to the Application Under Test (AUT) and completely trusted, i.e., for each possible input specified for the AUT it would always compute a correct output. As a result it is very difficult to develop a perfect oracle. There are very few techniques that have been proposed to automate oracle. Testers are in general assumed to provide the expected behavior of the software in the form of a table of pairs (Peters and Parnas, 1994), logical expressions which must be satisfied by the software (Bousquet *et al*., 1999), or temporal constraints that specify conditions which must not be violated during software execution (Dillon and Ramakrishna, 1996). In capture/replay tool which is used to test Graphical User Interfaces (GUI) software expected outputs are saved while recording test scripts or inserted in the test scripts manually (Andersson and Bache, 2004; Chen *et al*., 2005; Ostrand *et al*., 1998). It however need a lot of labor because there are different expected output for different input of the software. Chen proposes a specification based testing approach for GUI software. The expected behavior is manually contained in the model of the software (Chen and Subramaniam, 2002). Memon presents a planning method to generate expected outputs (Memon and Xie, 2005). It needs to construct GUI model and set conditions for each operator manually. Schroeder can automatically generate a large combinatorial test set that include inputs and expected outputs by manually generating the expected relationship between input and output from requirement specification documents (Schroeder *et al*., 2002). To implement more automated oracle, Aggarwal explores using neural networks based approach to generate expected outputs (Aggarwal *et al*., 2004) for the triangle classification problem (Aggarwal and Singh, 2001). By experiment they made the conclusion that neural networks can be used as oracle with reasonable degree of accuracy for classification problem in software testing.

GUI software can be logically divided into two parts, one is GUI and the other is underlying logic implementation. In the underlying logic implementation the relationship from inputs to outputs is in nature a function. When the function is continuous an automated oracle is designed in this paper to generate expected outputs and to compare actual outputs with them. AUT in this paper is used to represent the underlying logic implementation of the GUI software. Radial Basis Function Neural Networks (RBF NN) are then used to implement the oracle. RBF NN has gained much popularity for that it can approximate complex nonlinear mappings directly from the input and output data with a simple topology structure (Alippi *et al*., 2001; Duda *et al*., 2001; Karayiannis and Mi, 1997; Li *et al*., 2004; Roy *et al*., 1997). RBF NN can approximate arbitrary continuous function effectively without the need to have knowledge of that function. Few experiments have been conducted to validate the results of our method.

Corresponding Author: Ying Lu, Computer Science and Engineering College, Xi'an Technological University, Xi'an 710032, China

STATEMENT OF THE PROBLEM

AUT accepts inputs from a user or system, computes results and output them. AUT discussed in the paper is a determined program. That is, for the same inputs, program will always produce a determined output. Let $x = (x_1,...,x_n)$ be the input vector to AUT and $y = (y_1,..., y_m)$ be the output vector from the AUT. Then the relationship between x and y is $y = f(x)$. Each component y_j in y is also a function of x, i.e., $y_j = f_j(x)$, $j = 1,...,m$.. The function investigated in this paper is continuous. Let V_{x_i} be the set of all possible values of x_i and V_x be the set of all possible values of x. Therefore V_x includes every possible combination of the value from V_{x_i} and:

$$\|V_x\| = \prod_{i=1}^{n} \|V_{x_i}\| \quad (1)$$

where $\|.\|$ is the size of a set. From the Eq. 1 it can be seen that the size of V_x is determined by the size of V_{x_i} and n. The size of V_x is usually very large. Let x^i be a data item in V_x and $y^i = f(x^i)$, then x^i is a test case and y^i is the expected output of the AUT under the test case. The number of test cases is huge due to the size of V_x. To manually generate all expected outputs y^i for these test cases need a lot of time and labor. A new technique is explored to overcome it by an automated method. Automated test oracle can generate y^i from x^i and compare actual output with it automatically.

RBF NEURAL NETWORKS

A RBF NN can be regarded as a three-layered networks with input, hidden and output layers (Alippi *et al*., 2001; Duda *et al*., 2001; Karayiannis and Mi, 2003; Li *et al*., 2004; Roy *et al*., 1997). Each hidden neuron in RBF NN represents a RBF. An output neuron computes the weighted sum of the hidden neurons' outputs. The output of an output neuron is as follows:

$$F(x) = \sum_{j=1}^{m} w_j \phi(\|x - c_j\|) \quad (2)$$

where $x \in R^n$ is an input vector, ϕ is a radial basis function in hidden layer, $w_j \in R$ is the weight between jth hidden neuron and the output neuron, $c_j \in R^n$ is the center of the RBF in the jth hidden neuron and m is the number of hidden neurons. $\|x-c_j\|$ stands for the distance between the input vector x and the RBF center c_j. From the Eq. 2 it can be seen that the output of the RBF NN is a weight sum of the hidden layer's RBF. A RBF is a local function. One of the special RBF commonly used is a Gaussian function which has the form as follows:

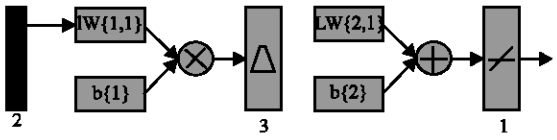

Fig. 1: The architecture of a RBF NN with two inputs, three neurons in hidden layer and one output

$$\exp\left(-\frac{\|x - c_j\|^2}{2\sigma^2}\right) \quad (3)$$

where $\sigma>0$ is a parameter. Generally Gaussian functions are superior to sigmoid function in estimating a broad class of functions. By using the Gaussian function RBF NN can approximate arbitrary continuous functions in theory. In MATLAB7 the parameter σ in the Eq. 3 is replaced by the parameter β:

$$2\sigma^2 = \left(\frac{\beta}{0.8326}\right)^2 \quad (4)$$

where β is the width of the neuron in hidden layer.

Figure 1 is the architecture of a RBF NN with two inputs, three neurons in hidden layer and 1 output.

A RBF NN is usually trained in two important steps. The first step is to find centers of the networks, which are parameters of hidden neurons. The second step is to determine the weights between the hidden layer and output layer. More information about training algorithm can be found in references (Chen *et al*., 1991; Roy *et al*., 1997; Schilling *et al*., 2001).

AUTOMATED ORACLE BASED ON RBF NEURAL NETWORKS

General model of oracle is as Fig. 2. The expected outputs of the AUT are generated according to inputs and then compared with the actual outputs from AUT. If they are not same, it implies a failure. The process of generating expected outputs is traditionally implemented manually or by formal specification such as Z specification which is manually generated from specification documents of the AUT. Expected outputs are then compared with actual outputs by determining if they are exactly same. If they are same, it means no failure. Otherwise a failure is reported. To automate oracle, a new oracle is presented in Fig. 3.

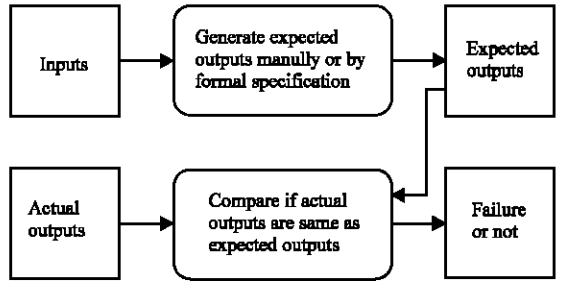

Fig. 2: General oracle model

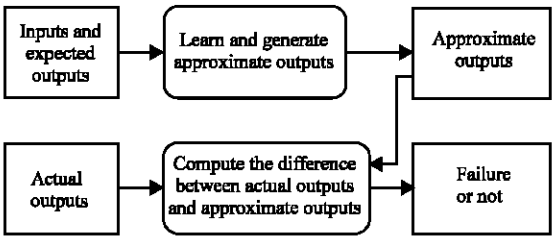

Fig. 3: Automated oracle model

In our automated oracle model the relationship between inputs and outputs are learned from a small set of training data which consists of inputs and expected outputs and the relationship learned is then used to generate approximate outputs for AUT. Let X and Y be the domain and co-domain of the AUT. The training set is:

$$D = \{(x,S(x)) \mid x \in X', X' \subset X, S(x) \in Y\} \quad (5)$$

where S(x) is the expected output which is computed from the specification of the AUT when the input is x. Train RBF NN from D to generate the relationship O:X→Y for AUT. Trained RBF NN is then used to generate the approximate outputs for any input of the AUT. The approximate outputs are not as same as the expected outputs. But they can approach the expected outputs in any precision. Comparison process is now implemented by computing the difference between the actual outputs and the approximated outputs as follows:

$$|t-\alpha|_\varepsilon = \begin{cases} 0 & \text{if } |t-\alpha| \le \varepsilon \\ |t-\alpha| - \varepsilon & \text{otherwise} \end{cases} \quad (6)$$

where t and α are actual outputs and approximate outputs respectively and ε controls the criterion if an actual output t is right or not. We can adjust ε between test precision and test cost. In experiments we will describe t he effect of the ε. If $|t\text{-}\alpha|_\varepsilon = 0$, it means that the actual outputs are right within the precision ε. Otherwise, if $|t\text{-}\alpha|_\varepsilon > 0$, it means that a failure occurs because the actual outputs are not right within the precision ε. The difference between the actual and approximate outputs is described by $|t\text{-}\alpha|\text{-}\varepsilon$. The larger it is, the more difference there is. RBF NN has been established to be an effective tool for approximations of the function. It can approach any continuous function in theory. This feature can be used to generate approximate outputs in an automated oracle. To implement the automated oracle, two processes must be automated. One is to generate approximate outputs by RBF NN and the other is to compare the actual outputs from AUT with the approximate outputs. Automated oracle can be summarized as follows.

Step 1: Generate training set and set the parameter β.
Step 2: Train RBF NN by the training set.
Step 3: When the training process finishes, keep the weight and go to step 4.
Step 4: Set test precision ε.
Step 5: Get an input from the specification of the AUT and then input it to RBF NN and AUT respectively. Get the approximate output α from the RBF NN and the actual output t from the AUT.
Step 6: Compare α and t according to the Eq. 6. Determine if there is failure or not by the result.
Step 7: Repeat step 5 and 6 until other inputs of the AUT are tested.
Step 8: If it is needed to test in different precision, go to step 1. Otherwise, this process finishes.

EXPERIMENTS

The goal of the experiments is whether the method proposed is effective, i.e., if the oracle can generate the approximate output which is close to the expected output and if the test precision can be adjusted by parameters. In our experiment we used single input variable, two input variables and multi input variables function respectively. The number of the output variables is one. It can be generalized to the situations of multi output variables. We use the parameter β in the Eq. 4 which is used in MATLAB 7 instead of the parameter σ in the Eq. 3.

AUT of single input variable function: Let the relationship between input and output of the AUT is as follows:

$$y = x^3 \sin(3\pi x)e^{1-x^2} + x\cos(5x) + 2\cos(7x) + 2x\tanh(10x)\cos(2\pi x) \quad (7)$$

Where $0 \le x \le 2\pi$. It is a single input variable function. Generate training set of 100 samples manually. The variable x is evenly dropped from the interval $[0,2\pi]$ and the variable y is computed manually. We set the parameter $\beta = 0.1$. When training process finishes, test the output on 10000 samples to see if the approximate output is close to the expected output. The expected outputs, approximate outputs obtained from RBF NN and the difference between them is as Fig. 4. The difference is in the interval [-0.0287,0.0834]. It shows the difference is small enough when β is set to 0.1.

Different test precision can be achieved when changing the value of β (Table 1). The column Min and Max in Table 1 are the min and max difference, respectively between the expected and approximate outputs. Column ε is the test precision that can be achieved under the different value of β. It shows that we can change the value of the parameter β to test the AUT in different demand of test precision in software testing. If a precision ε below 0.003 is needed, we can set $\beta = 0.25$. If the actual outputs are in the interval $[\alpha-0.0023, \alpha+0.0023]$, where α are the approximate outputs obtained from the RBF NN, it means there is no failure. Otherwise, it means a failure occurs.

AUT of two input variables function: Let the relationship between input and output of the AUT is as follows:

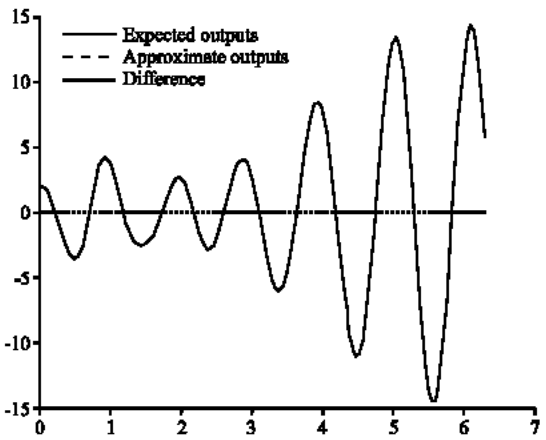

Fig. 4: Plots of the expected, approximate outputs and their difference when $\beta = 0.1$ for the function in Eq. 7

Table 1: Precision achieved under the different value of β for the function in Eq. 7

β	Min.	Max.	ε	β	Min.	Max.	ε
0.0500	-0.4007	0.1290	0.4007	0.5000	-0.0006	0.0037	0.0037
0.0750	-0.0606	0.0643	0.0643	0.7500	-0.0427	0.0196	0.0427
0.1000	-0.0287	0.0834	0.0834	1.0000	-0.3182	0.3047	0.3182
0.2500	-0.0003	0.0023	0.0023	1.2500	-0.5721	0.5788	0.5788

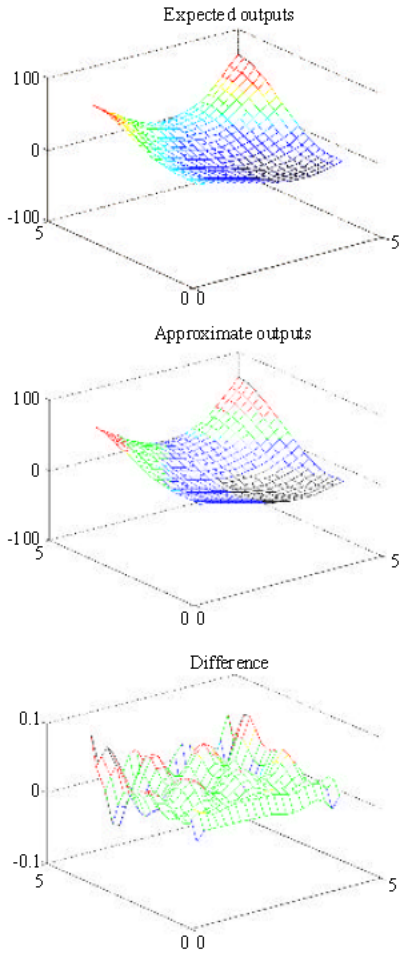

Fig. 5: Figures of the expected, approximate outputs and their difference when $\beta = 5$ for the function in Eq. 8

Table 2: Precision achieved under the different value of β for the function in Eq. 8

β	Min.	Max.	ε	β	Min.	Max.	ε
1.0000	-0.2500	0.7228	0.7228	5.0000	-0.0673	0.0687	0.0687
2.0000	-0.0421	0.0076	0.0421	6.0000	-0.1009	0.1572	0.1572
3.0000	-0.0511	0.0112	0.0511	7.0000	-0.1515	0.3089	0.3089
4.0000	-0.0361	0.0105	0.0361	8.0000	-0.1929	0.3567	0.3567

$$y = (x_1^2 + x_2^2)\cdot(\cos(x_1) - \sin(x_2)) + 3x_1x_2e^{1-x_1^2} \qquad (8)$$

where $1 \le x_1, x_2 \le 5$. It is a two input variables function. Generate training set of 100 samples manually. The variables x_1 and x_2 are evenly dropped from the interval [1,5]. The variable y is computed manually. We set the parameter $\beta = 5$. When training process finishes, test the output on 400 samples to see if the approximate output is close to the expected output. The expected outputs,

Table 3: Precision achieved under the different value of β for the function in Eq. 9

β	Min.	Max.	ε	β	Min.	Max.	ε
3.0000	-1.0339	0.4886	1.0339	7.0000	-0.0380	0.0182	0.0380
4.0000	-0.3375	0.1665	0.3375	8.0000	-0.0229	0.0103	0.0229
5.0000	-0.1475	0.0657	0.1475	9.0000	-0.0147	0.0062	0.0147
6.0000	-0.0746	0.0303	0.0746	10.0000	-0.0091	0.0045	0.0091

approximate outputs obtained from RBF NN and the difference between them is as Fig. 5. The difference is in the interval [-0.0673,0.0687]. It shows the difference is small enough when β is set to 5.

Different test precision can be achieved when changing the value of β (Table 2). The column in Table 2 has the same meanings as Table 1. It shows that we can change the value of the parameter β to test the AUT in different demand of test precision in software testing. If a precision ε below 0.04 is needed, we can set β = 4. If the actual outputs are in the interval [α-0.0361,α+0.0361], where α are the approximate outputs obtained from the RBF NN, it means there is no failure. Otherwise, it means a failure occurs.

AUT of multi input variables function: Let the relationship between input and output of the AUT is as follows:

$$y = x_1^2 + x_2^2 + x_3^2 + 2x_1x_2 + 3x_2x_3 + 5x_1x_3 \quad (9)$$

where $1 \le x_1, x_2 x_3 \le 5$. It is a three input variables function. Generate training set of 125 samples manually. The variable x_1, x_2 and x_3 is evenly dropped from the interval [1,5] and the variable y is computed manually. We set the parameter β = 5. When training process finishes, test the output on 1000 samples to see if the approximate output is close to the expected output. The difference is in the interval [-0.1475,0.0657]. It shows the difference is small enough when β is set to 5.

Different test precision can be achieved when changing the value of β (Table 3). The column in table 3 has the same meanings as Table 1. It shows that we can change the value of the parameter β to test the AUT in different demand of test precision in software testing. If a precision ε below 0.01 is needed, we can set β = 10. If the actual outputs are in the interval [α-0.0091, α+0.0091], where α are the approximate outputs obtained from the RBF NN, it means there is no failure. Otherwise, it means a failure occurs.

CONCLUSIONS

An automated oracle is proposed in this paper for automated software testing. From the results above we conclude that RBF NN can be used to implement the automated oracle. It can generate approximate outputs which are close to expected outputs for AUT. The test precision can be adjusted by parameters. As a result AUT can be tested in the precision needed. The actual outputs from AUT are compared with the approximate outputs in an interval to determine correctness. If actual outputs are in the interval specified by the approximate outputs and the test precision, it means there is no failure. Otherwise, a failure is exposed. By the method we need not to manually generate expected outputs for all possible inputs of the AUT. It can therefore save a lot time and cost in software testing.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their careful reading of this paper and for their helpful and constructive comments.

REFERENCES

Aggarwal, K.K. and Y. Singh, 2001. Software Engineering: Programs, Documentation, Operating Procedures. New Age International Publishers.

Aggarwal, K.K., Y. Singh, A. Kaur and O.P. Sangwan, 2004. A neural net based approach to test oracle. ACM SIGSOFT Software Engineering Notes, ACM Press, 29: 1-6.

Alippi, C., V. Piuri and F. Scotti, 2001. Accuracy versus complexity in RBF neural networks. IEEE Instrumentation and Measurement Magazine, 4: 32-36.

Andersson, J. and G. Bache, 2004. The video store revisited yet again: Adventures in GUI acceptance testing. LNCS3092, Springer Berlin Heidelberg, pp: 1-10.

Bousquet, L.D., F. Ouabdesselam, J.L. Richier and N. Zuanon, 1999. Lutess: A specification-driven testing environment for synchronous software. In: Proc. of the 21th International Conf. on Software Engineering, ACM Press.

Chen, S., C.F.N. Cowan and P.M. Grant, 1991. Orthogonal least squares learning algorithm for radial basis function networks. IEEE Trans. Neural Networks, 2: 302-309.

Chen, J. and S. Subramaniam, 2002. Specification-based testing for GUI-based applications. Software Quality J., 10: 205-224.

Chen, W.K., T.H. Tsai and H.H. Chao, 2005. Integration of specification-based and CR-based approaches for GUI Testing. In: Proc. of the 19th International Conf. on Advanced Information Networking and Applications.

Dillon, L.K. and Y.S. Ramakrishna, 1996. Generating oracles from your favorite temporal logic specifications. In: Proc. of the 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering.

Duda, R.O., P.E. Hart and D.G. Stork, 2001. Pattern Classification. 2nd Edn., John Wiley and Sons.

Karayiannis, N.B. and G.W. Mi, 1997. Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques. IEEE Trans. Neural Networks, 8: 1492-1506.

Li, Y.H., S. Qiang, X.Y. Zhuang and O. Kaynak, 2004. Robust and adaptive backstepping control for nonlinear systems using RBF neural networks. IEEE Trans. Neural Networks, 15: 693-701.

Memon, A., I. Banerjee and A. Nagarajan, 2003. What test oracle should I use for effective GUI testing. In: Proc. of the 18th IEEE International Conf. on Automated Software Engineering.

Memon, A. and Q. Xie, 2005. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. IEEE Trans. Software Engineering, 31: 884-896.

Ostrand, T., A. Anodide, H. Foster and T. Goradia, 1998. A visual test development environment for GUI systems. In: Proc. the ACM SIGSOFT International Symposium on Software Testing and Analysis.

Peters, D. and D.L. Parnas, 1994. Generating a test oracle from program documentation. In: Proc. of the International Symposium on Software Testing and Analysis.

Roy, A., S. Govil and R. Miranda, 1997. A neural-network learning theory and a polynomial time RBF algorithm. IEEE Trans. Neural Networks, 8: 1301-1313.

Schilling, R.J., J.J. Carroll and A.F.A. Ajlouni, 2001. Approximation of nonlinear systems with radial basis function neural networks. IEEE Trans. Neural Networks, 12: 1-15.

Schroeder, P.J., P. Faherty and B. Korel, 2002. Generating expected results for automated black-box testing. In: Proc. of the 17th IEEE International Conf. Automated Software Engineering.