Journal of Computer Science 1 (3): 341-345, 2005
ISSN 1549-3636

Empirical Studies on Applications of Neural Networks as Approximation Oracle

K.K.Aggarwal, Yogesh Singh and Arvinder Kaur
G.G.S.Indraprastha University, Delhi, India - 110006

Abstract: Expected results are essential to any test design. In testing, an Oracle is a trusted source of expected results. This research examines the performance of Artificial Neural Network (ANN) as Approximation Oracle. The supervised learning paradigm has been used and found to be suitable for the purpose with the help of case studies.

Key words: Test Oracle, Neural Networks, Software Testing

INTRODUCTION

Software testing is the process of attempting to detect discrepancies between a program's specification and its actual behaviour. This requires test suite development. In this phase, a set of test cases has to be constructed that (in principal) has the capability of exposing any/every fault in the software. A single test case may be interpreted as a tuple in which the first element is the member of set of possible inputs to the software and the second element is the corresponding (desired) output for the implementation under test (IUT) [1-4]. So a test case may be represented as:

Test Case ={<inputs>, <outputs>} (1)

The set of all test cases of this type constitute the test suite.

The test suite cannot be used without the desired output value being a part of the test case. Therefore in many approaches in software testing the presence of a Software Oracle is assumed [5]. An oracle is a trusted source of expected results for a given input. The Oracle (for a specific system) can be a program specification, a table of examples, or simply the programmer's knowledge of how a program should operate [6]. If an Oracle can be built for the IUT, the development of test suite can easily be implemented but developing a perfect oracle however is something like a philosopher's stone for software. We cannot guarantee the ability of any algorithm to decide that an algorithm is correct in all possible cases [7]. So in simple situations the expected results are generally obtained manually or some partial or approximation oracle is used [6]. An Approximation Oracle develops approximate expected results. In many complex situations such as simulation in mechanical design, weather and economic forecasting and geological exploration there is no simple way to check the output generated by IUT and errors in these problems are as high as 500%. So an Approximation Oracle is one of the solutions that will give confidence in the results obtained from IUT. In this research, we present an Artificial Neural Network (ANN) based approach to the development of an Approximation oracle for a given IUT.

Neural Network Methodology

Neural Network: Neural Networks (NNs) comprise a set of interconnected neurons, each having a transformation function that it performs on the weighted sum of its inputs to produce an output. In NN modeling, we determine a pattern of connections (architecture), a method of determining interconnection weights w (training algorithm) and a transformation function. To select the architecture of NNs, the number of nodes in the input and output units, the number of hidden units and the number of nodes in each hidden unit needs to be decided along with the types of connection between all the nodes. The number of input units and the number of output units are problem dependent. The number of hidden units is usually not known. Hidden units allow the neural network to learn by forcing the network to develop its own internal representation of the input space. The network that produces the best classification with the fewest units is selected as the best topology. There is no clearly defined theory that allows for the calculation of the ideal parameter settings and as a rule even slight parameter changes can cause major variations in the behavior of almost all networks [8]. There are broadly two paradigms of neural learning algorithms namely Supervised and Unsupervised. Supervised training involves the process of adjusting the weights in a neural network using a training algorithm where the desired output for each of the training input vectors is presented to the network. As a guideline approximately 10 training data/observations are required for each weight in the neural network [8] In Unsupervised training, the weights in a neural network are modified without specifying the desired output for any input patterns. Supervised training has been used in this work. *Error*

Backpropagation is the most popular training algorithm for multilayer neural networks [9-12].

Many software engineering tasks can be formulated as basic learning tasks that ANN can perform. So several areas in software development have already witnessed the use of Neural Networks. Finnie and Witting used back propagation neural network models for effort estimation and reported accuracies of within 10% for a model of this type. They compared analogy based method using CBR with different versions of FP-based Regression models and ANN. The data used consisted of 299 projects from 17 different organizations and concluded that ANN performed better than Analogy followed by regression models [8]. According to Gray and McDonnell, neural networks is the most common software estimation model-building technique used as an alternative to mean least squares regression. These are estimation models that can be "trained" using historical data to produce ever better results by automatically adjusting their algorithmic parameter values to reduce the delta between known actual and model predictions [13]. ANN has also been used for metrics generation [14, 15], reliability prediction [16] and reverse software engineering [17, 18]. A. Mayrhauser et al. has shown neural network to be a useful approach for test case effectiveness prediction [10, 11, 19].

Neural Network Architecture: The network used in this work belongs to Multilayer Feed Forward networks and is referred to as M-H-Q network with M source nodes, H nodes in hidden layer and Q neurons in the output layer as shown in Fig. 1. The input nodes are connected to every node of the hidden layer but are not directly connected to the output node. Thus the network does not have any lateral or shortcut connection. In our work, log sigmoidal activation and purelin activation function is used.

The log-sigmoid activation function is given as:

$$\sigma(x) = \frac{1}{1+e^{-x}};\ x \in \Re \tag{2}$$

The purelin activation function is given as:

$$\sigma(x) = x;\quad x \in \Re \tag{3}$$

Where, $\Re$ is the set of real numbers.

The weights between the jth hidden node and input nodes are represented by W_{ji} while the weights between the jth hidden node and output node are represented by $_j$. The threshold of the jth hidden node is represented by $_j$ while the threshold of the output layer is represented by . If x represents the input vector to the network, the net input to the hidden node j is given by:

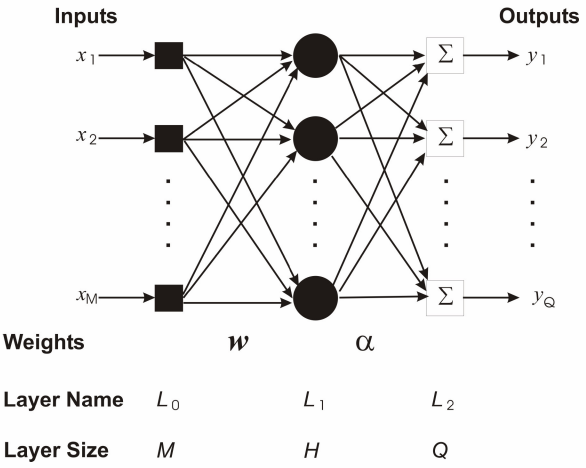

Fig. 1: Architectural Graph of ANN

$$net_j = \sum_{i=1}^{M} W_{ji}x_i + \beta_j;\quad j = 1,2.....N` \tag{4}$$

The output from the jth hidden node is:

$$\sigma_j = \sigma(net_j) \tag{5}$$

The output from the network is given by:

$$y = \sigma\left[\sum_{j=1}^{N} \alpha_j \sigma_j + \beta\right] \tag{6}$$

Thus the output obtained from the network, corresponding to input vector x is given by y, while the actual target value for the input x is represented by t. NN repetitively adjusts different weights so that the difference between desired output t from the network and actual output from NN is minimized. Figure1 represents the network used in this work.

It is known that ANN can be used to model any arbitrary input-output mapping and are capable of approximating any measurable function. So ANN should be able to model the functionality of the software also. In a broad sense, the neural network itself is a model because the topology and transfer functions of the nodes are usually formulated to match the current problem [9, 20].

Software Testing Problems/Case Studies: We used two different case studies in our experiments. So in our work we refer these case studies as case study 1(C1) and Case study 2(C2).

Case Study 1(C1): The first case study taken is the famous Triangle Classification Problem TRIYP [1, 3, 5, 7]. The program accepts three integers that represent the relative lengths of the sides of a triangle.

The output of the program is Equilateral, Isosceles, Scalene or Not a triangle depending on length of sides of triangle. The input variables can have any integer value between 1-200. Since all the three input variables can have any value between 1-200, the number of input combinations is 200^3 = 80 Lacs.

Case Study 2(C2): The second case study taken is the output generation for Unadjusted Function Points (UFP). Function points are the most popular sizing metric given by Albrecht and Gaffney [21]. The principal of Albrecht's Function Point is that a system is decomposed into five functional components (inputs, outputs, inquiries, internal logical files and external interfaces).
The number of each component type (EIF, ILF, EI, EO, EQ) present is placed into its assigned cell next to its weight in the matrix shown in Table 1.
Their complexity is classified as being relatively low, average or high according to a set of standards that define complexity in terms of objective guidelines [21]. Then UFP count of all five components is added to give total UFP.

Table 1: Complexity Weight Matrix

Type	Complexity of component			
	Low	Average	High	Total
External inputs	__*3=__	__*4=_ _	__*6=_ _	
External output	__*4=__	__*5=__	__*7=_ _	
External inquiries	__*3=__	__*4=__	__*6=_ _	
Internal logical files	__*7=__	__*10=__	__*15=__	
External interface file	__*5=__	__*7=__	__*10=__	
Total UAF				

Experimental Design: The experiment conducted is to explore the possibility of usage of ANN as Approximation Test Oracle. We used both case studies for this purpose.

Case Study1 (C1): In first case study, producing output manually for 80Lac combinations of input is a tough job. So we explored the possibility of usage of ANN to produce output for each combination of input.
Here number of inputs(M)=3, corresponding to three sides of triangle. The number of neurons in the output layer (Q) were set to 2 to represent four possible outputs as described below as target values.
Test cases were divided into four categories based on four types of expected outputs and their target values were set as mentioned below:

* T1 (Test cases corresponding to Equilateral output) Target Value: 1 1
* T2 (Test cases corresponding to Isosceles output) Target Value: 0 1
* T3 (Test cases corresponding to Scalene output) Target Value: 1 0
* T4 (Test cases corresponding to Not a triangle output) Target Value: 0 0

To find the number of neurons (H) in the first (hidden) layer experiment was done with an ensemble of networks, starting from network with 10 hidden neurons and going up to 30 neurons to find out which of these networks gives the best performance, keeping the output layer neurons constant in each of the networks i.e 2 representing the four target values. Transfer functions were '*log-sigmoid*' in both the layers. In this experimentation the performance varied from 0.0846096 to 0.0298441. Each of the networks was trained with 10000 epochs, keeping the goal as 0.03. The network with 16 and 25 hidden neurons are found to be most appropriate for further study, as performance in both these cases is found to be 0.0299.
After deciding the network architecture, the network is trained with the initial training set of 150 exemplars. After the training, simulation is done on the network. Training and Simulation was done with 16 and 25 neurons in the hidden layer to see the difference in performance. It was found that the net with 16 neurons performed better. Earlier we have shown the simulation results with 16 neurons in the hidden layer.

Case Study 2(C2): We trained an ANN to compute UFP for every possible combination of input. Then results from algorithmic equation and Neural Network have been compared to see if ANN gives accurate results and can act as test oracle to produce output for every combination of input. In this case study number of inputs (m)=15, corresponding to 5 input components (inputs, outputs, enquiries, internal logical files and external interfaces) with each component having 3 complexity levels. So for each of the five components, there are three inputs corresponding to Low, Medium and High complexity. The number of neurons in the output layer (Q) were set to 1 to represent Unadjusted function points. In the first (hidden) layer neurons (H) were varied from 2 to 8.The performance is found to be best with 6 neurons. The linear transfer function '*purelin*' is used in both the layers. All the input values are normalized in the range of 0 to 1. A program is written in C language. It chose the value of 15 input factors randomly between 0 to 100 and correspondingly computed the Unadjusted Function Points.
Initially the network is trained with 20 training data. It was found that output was perfect. So three new networks were trained with 15,10,5 exemplars only to see the difference in the performance. It was found that the performance change was significant. Table 2 lists the architecture topology for each case study

Table 2: Architecture Topology of Case Studies

Experiment	Case Study	Input nodes	Hidden nodes	Output nodes	Transfer function
E1	C1	4	16	2	Log sigmoid
E1	C2	15	6	1	Purelin

Measures Used for Analysis of Results: The following measures are used for analysis of the results.

$$\text{Percentage Quality of Predication (\%QP)} = \frac{N_{RC}}{N} * 100 \quad (7)$$

$$\text{Percentage Misclassification error (\%MCE)} = \frac{N_{MC}}{N} * 100 \quad (8)$$

$$N = N_{MC} + N_{RC} \quad (9)$$

Where N_{MC}, N_{RC} are the numbers of test cases misclassified, rightly classified. N is the number of Total Test cases. The criterion for misclassification was that if the desired value is say (0 0), then any value different than (0 0) is a misclassification.

EXPERIMENTAL RESULTS AND DISCUSSION

Figure 2 gives the simulation plot for case study1. Simulation was performed on 30 different data sets with 140 test cases (T1, T2, T3and T4 combined) in each set. Test cases were chosen randomly, by a Random Number Generator. From Fig. 2, we can see that the Mean Misclassification error is 7.813% with standard deviation of 2.273. Quality of Prediction = 92.187%

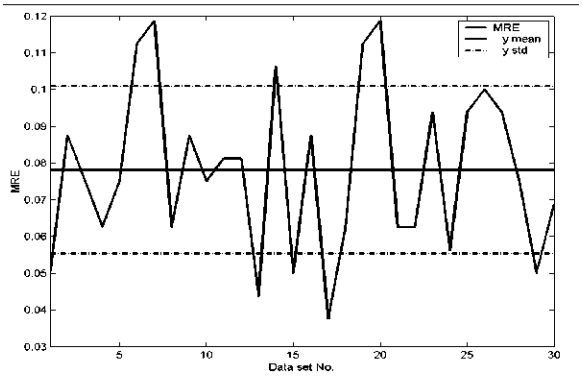

Fig. 2: Simulation Plot for Case Study 1

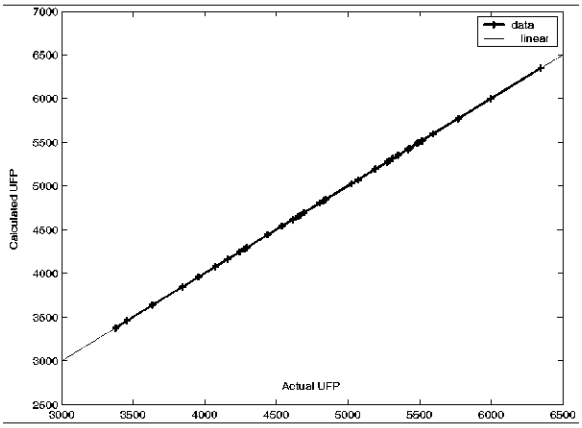

Fig. 3: UFP Results 20 Training Inputs, 6 Neurons in Hidden Layer

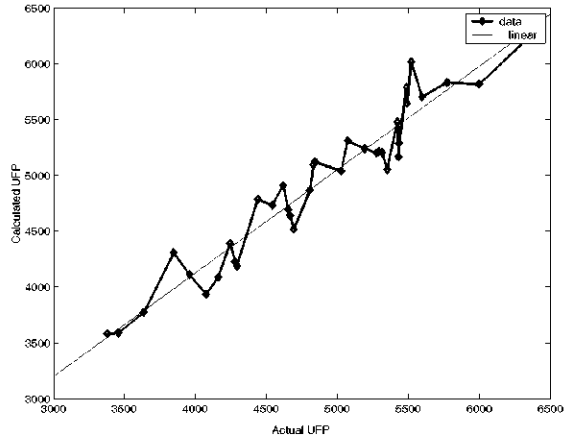

Fig. 4: UFP Results 15 Training Inputs, 6 Neurons in Hidden Layer

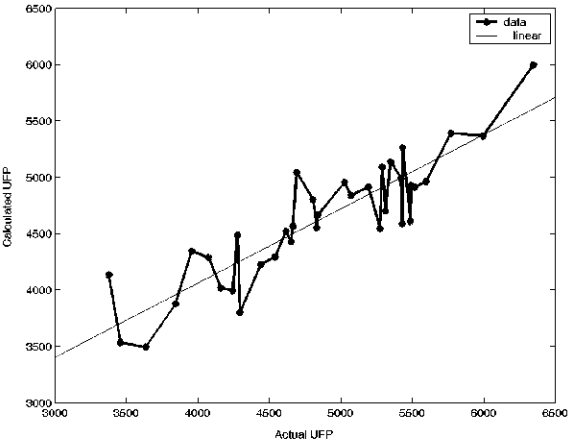

Fig. 5: UFP Results 10 Training Inputs, 6 Neurons in Hidden Layer

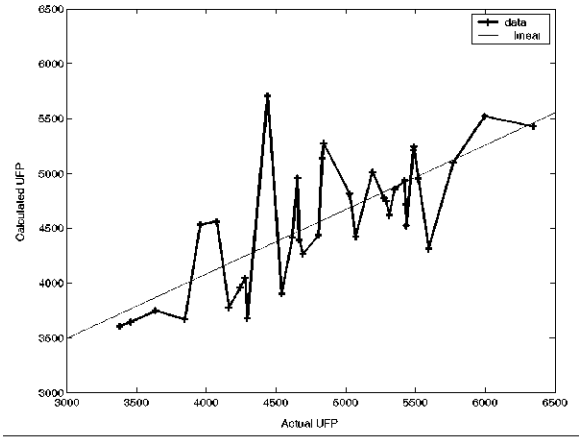

Fig. 6: UFP Results 5 Training Inputs, 6 Neurons in Hidden Layer

Case Study 2: Figures 3-6 show the results of test data with 6 neurons in the hidden layer and 20, 15, 10, 5 training data, respectively. These figures are plot between UFP computed using algorithmic equation and calculated UFP obtained from ANN.

In first case study out of 80 Lac possibilities of inputs, the net has been trained with only 155 inputs. If we substantially increase the number of training data, the accuracy of neural network can further be improved. In second case study, it can be observed from the graphs that calculated results completely mapped the actual results with 20 training data. So a perfect Oracle is achieved in this case. With 15,10,5 training data the results are oscillating around the linear line. As we kept on decreasing the training data the oscillations are increased. The training data required in this case was very less against the popular belief that neural networks require large amount of data for training.

CONCLUSION

In first case study, the Quality of prediction achieved is 92.187%. In second case study the results are 100% perfect. The results are very encouraging and suggest that ANN can be used as Approximation oracle and in some cases as perfect oracle also. In complex situations where there is no simple technique for checking the output produced by the IUT, the usage of ANN based Approximation test Oracle can give confidence in predictions. We intend to further explore ANN with more case studies and with different architectures. It is also possible to explore alternative neural network models like cascade correlation model [9] that dynamically build the neural network architecture model. We can further explore if we can achieve a perfect Oracle in all cases using ANN .

REFERENCES

1. Aggarwal, K.K., Yogesh Singh, Arvinder Kaur and O.P. Sangwan, 2004. A neural net-based approach to test oracle. ACM SIGSOFT, 29: 1-6.
2. Aggarwal, K.K., Yogesh Singh, Arvinder Kaur, 2004. Code coverage based technique for prioritizing test cases for regression testing. ACM SIGSOFT, 29: 5.
3. Beizer, B., 1990. Software Testing Techniques. Van Nostrand Reinhold, New York.
4. Collofello, J.S. Introduction to Software Verification and Validation. SEI-CM-13-1.1, Software Engineering Institute, Pittsburgh, P.A., USA.
5. Aggarwal, K.K. and Yogesh Singh, 2001. Software Engineering: Programs, Documentation, Operating Procedures. New Age International Publishers.
6. Binder, Robert V., 1999. Testing Object-Oriented Systems-Models, Patterns and Tools. Addison Wesley.
7. DeMillo, R.A., R.J. Lipton and F.G. Sayward, 1978. Hints on test data selection: Help for practicing Programmers. IEEE Computer, C-11: 34-41.
8. Finnie, G.R. and G.E. Wittig, 1996. AI tools for software development effort estimation. Software Engineering and Education and Practice Conference. IEEE Computer Society Press, pp: 346-353.
9. Laurene Fausett, 1994. Fundamentals of Neural Networks. Prentice Hall: Englewood Cliffs, New Jersey.
10. Mayrhauser, A. Von, C. Anderson and R. Mraz, 1995. Using a neural network to predict test case effectiveness. Proc. IEEE Aerospace Applications Conference, Snowmass, CO.
11. Mayrhauser, A. Von, J. Walls and R. Mraz, 1994. testing applications using domain based testing and sleuth, Proc. Fifth Intl. Software Reliability Engineering Symposium, Monterey, pp: 206-215.
12. Zurada Jacek M., 1992. Introduction to Artificial Neural Systems. West Publishing: St. Paul.
13. Gray, A.R. and S.G. MacDonnell, 1997. A comparison of techniques for developing predictive models of software metrics. Information and Software Technology, pp: 425-437.
14. Boetticher, G. and D. Eichmann, 1993. A neural net paradigm for characterizing reusable software. Proc. First Australian Conference on Software Metrics, pp: 41-49.
15. Boetticher, G., K. Srinivas and D. Eichmann, 1993. A neural net-based approach to software metrics. Proc. Fifth Intl. Conference of Software Engineering and Knowledge Engineering, San Francisco, CA, pp: 271-274.
16. Karunanithi, N., D. Whiyley and Y.K. Malaiya, 1992. Using neural networks in reliability prediction. IEEE Software, 9: 53-9.
17. Whittington, G., C.T. Spracklen and J. MacRae, 1991. Applications of artificial neural networks to reverse software engineering. Proc. Second IEE Conference on Artificial Neural Networks, Bournemouth, UK.
18. Whittington, G. and C.T. Spracklen, 1992. The potential for Artificial Neural Networks within Software Engineering. Presented at the IEE Colloquium on Software Engineering and AI.
19. Anderson, C., A. Mayrhauser and R. Mraz, 1995. On the use of neural networks to guide software testing activities. Proc. Intl. Test Conference, Oct. 1995.
20. Haykin, S., 2003. Neural Networks. A Comprehensive Foundation. Prentice Hall, India.
21. Albrecht, A. and J.E. Gaffney, 1983. Software function source lines of code and development effort prediction: A software science validation. IEE Trans. Software Engineering, SE-9: 639-648.