

# Incorporating Evolutionary Approach with Neural Network for Automatic Creation and Optimization of Test Cases

Monika Chaudhary<sup>1</sup>, Komal Arora<sup>2</sup>

<sup>1</sup>M.Tech Student CSE Department, SPGOI, Rohtak  
*monika.chaudhary21@gmail.com*

<sup>2</sup>Asstt. Prof. in CSE Department, SPGOI, Rohtak  
*grewalkomal14@gmail.com*

## Abstract

Software testing is an important activity of the software development process, because most of the development cost is spent in the testing phase. In order to test the software, it is necessary to write test cases. Manually creating test data is very time consuming, especially for complex problems. It consumes resource and time. In our study we use evolutionary approach. The evolutionary approach that is used here is Genetic Algorithm. It automatically creates test cases for software application. For this, it first selects important and best test cases among all the test cases, on the basis of fitness function who will decide which test cases are good or best for execution. Then these selected test cases are used to create new test cases. But test cases are more likely to have faults. For this, multi-layer neural network is first trained by using randomly generated test data that confirm to the specification. Once training is over, a comparison tool, fault predictor is used for detecting the presence of fault in the software application. It compares the output of the application under test with the output of trained neural network and makes the decision about the presence of fault in the software application.

**Keywords:** *Evolutionary Approach, Genetic Algorithm, Automatic Creation of test cases, Heuristic approach, Neural Network in Software Testing.*

## I. Introduction

Software testing can be stated as the process of executing a program in order to find fault/defect in the program. It is a critical aspect of software quality assurance, because most of the development cost is spent in testing phase. In order to test the software, it is necessary to write test cases and generate test data for it. Manually creating test data is very time consuming, especially for complex problems. It consumes resources and adds nothing to the product in terms of functionality. Therefore, much effort has been spent in the development of

automatic software testing tools in order to significantly reduce the cost of developing software. Also test data that are good for one program are not necessarily appropriate for another program even if they have the same functionality. Therefore, an adaptive testing methodology for the software under test is necessary. Adaptive means that it monitors the effectiveness of the test data to the environment in order to produce new solutions with the attempt to maximise the test effectiveness.

The problem domain of this research is targeted at making evolutionary approach effective for test data generation. This study gives ways to improve the effectiveness of test data generators. These improvements, as well as other test data methods not based on evolutionary approach, are covered under the related work section. The main focus of this study is to automatically create test cases for software application. 100% code coverage is not our primary purpose. The test cases created by GA aim for a code coverage percentage of only 70-80%, as opposed to complete code coverage. This study is not restricted to one programming language, but applicable to all programming languages. Genetic Algorithm belongs to the larger class of Evolutionary Approach that mimics the process of natural evolution. It is an adaptive search methodology that is used in this research for automatic creation of test cases. But when the number of tests to be run can become very large, test case reduction becomes desirable. In such case we try to remove such test cases that are more likely to have faults, so that we are left only with those test cases that are not likely to yield failures. In this way, testing can be fast and efficient. This can be achieved by using neural networks with genetic algorithms. We used a neural network as a classifier to learn about the system under test and to

predict the fault exposure capability of newly generated test cases. We describe attributes of test cases to the neural network as inputs and relate them to resulting faults (neural network outputs). Then, we trained the network to recognize relationships between test case descriptors and faults. Once trained, the network acts as a fault predictor for new test cases.

## **II. Genetic Algorithm In Software Testing**

Genetic Algorithm is an adaptive search methodology that is based on natural process of evolution. In other words, it evolves new population from existing one. It is generally used in situations where the search space is relatively large and cannot be traversed efficiently by classical search methods. This is mostly the case with problems whose solution requires evaluation and equilibration of many apparently unrelated variables. As such they represent an intelligent exploitation of a random search space within a defined search space to solve a problem.

Genetic algorithms are very effective when searching or optimizing spaces that is not smooth or continuous. These are very difficult or impossible to search using calculus based methods, e.g. hill climbing. This population-by-population approach is very different from the more typical search methods of engineering optimisation. In many search methods, we move gingerly from a single point in the decision space to the next, using some decision rule to tell us how to get to the next point (hill climbing). This point-by-point method is dangerous because it often locates local peaks in the search space. GAs works from a population of points (individuals) simultaneously climbing many peaks in one generation (parallel), thus reducing the probability of finding a local optimum.

In software testing, working of genetic algorithm starts with an initial population of test suits. These test suits are generated randomly. The selection of initial population (starting generation) has a significant role in the creation of new population (next generation). Firstly, the fitness function of each test suit is calculated. The factors that form the basis for fitness function calculation are Likelihood, Close to boundary value and Branch coverage. These factors contribute to the selection of the test suit. These factors here are used as the evaluation of the test suit for its goodness. The search is guided by a survival of the fittest principle and it proceeds for a number of generations, for each generation the two test suits are selected

among the population, according to fitness function, to form a new population. This process of formation of new test suits from the old test suits is known as Reproduction. During Reproduction, two operators namely crossover, mutation are applied. During each generation, the test suits are passed through the cycle of Evaluation, Selection, Reproduction, Re-Evaluation. The cycle will repeat for a number of generations until certain termination criteria are met.

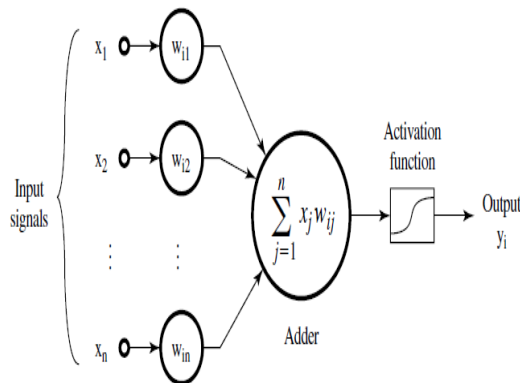
It could terminate after a fixed number of generations, or after a test suit with a certain high fitness value is created.

The power of genetic algorithms is the technique of applying a recombination operator (crossover and mutation) to a population of individuals. Despite their randomised nature, GAs are not a simple random search. GAs takes advantage of the old knowledge held in a parent population to generate new solutions with improved performance. Thereby, the population undergoes simulated evolution at each generation. Relatively good solutions reproduce; relatively bad ones die out and are replaced by fitter offspring. In most GAs, individuals are represented by a fixed-length string over a finite alphabet. The binary alphabet is one of many possible ways of representing the individuals. GAs work directly with this representation and they are difficult to fool because they are not dependent upon continuity of the parameter space and existence of a derivative. Test cases created by Genetic Algorithm are likely to have faults. Genetic Algorithm is not effective in terms of fault prediction. For this, neural network proves to be effective, which is to be discussed in following sections.

## **III. Background Of Artificial Neural Networks**

Artificial Neural Network (ANN) is a mathematical model inspired by biological neural networks. They were developed to simulate the behaviour, architecture and computation of the human brain. ANN consists of interconnected group of artificial neurons. There may be of one or more layers of neurons, the interconnections of which have associated synaptic weights. Each neuron has computational ability that contributes to the learning process or training of the network. The information computed during training is stored in the synaptic weights. Thus, ANNs are massive parallel information processing systems that learn and store knowledge about its environment.

The reasons of superior computational ability of ANN are its parallel design and its capability to extrapolate the stored information (gained during learning), to yield outputs for inputs that not specified during training. This makes ANN to solve complex problems very easily. ANNs emphasize automatic learning. Neural networks have been used as memories, pattern recall devices, pattern classifiers, and general function mapping engines.



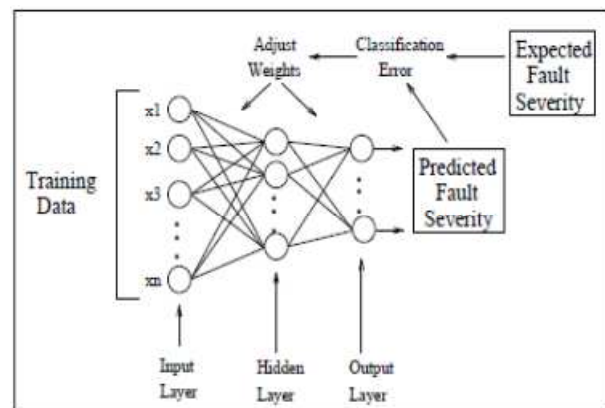
**Figure 1 Structure of neuron**

In our work we are using multilayer neural network which consist of three layers: input layer, hidden layer and output layer. The input layer consists of number of nodes, known as input units. Similarly the nodes in hidden layer and output layer are known as hidden units and output units respectively (see Figure 2). The number of input units and output units are problem dependent. The number of hidden units is usually not known. Hidden units are responsible for learning as it forces the network to develop its own internal representation of the input space. The network that produces the best classification with the fewest units is regarded as best topology. But a neural network with too few hidden units cannot perform the mapping accurately. On the other hand, a neural network with too many hidden units can “memorize” the training data but it cannot generalize well to new data.

Back propagation is the most popular training algorithm for multilayer neural networks. The network is initialized with a random set of weights and training is done by using input-output pairs. The training is actually done in two stages: forward pass and backward pass. In forward pass (the first stage of network training), the input vector starts

from input layer, and then propagating through the hidden layer reaches the output layer (see Figure 2). Each hidden units in the hidden layer calculates the weighted sum of the input vector and its interconnection weights and uses this weighted sum to calculate its activation. Now hidden layer is activated and propagates input vector to the output layer. Now each node in the output layer again calculates its weighted sum and activation and thus produces output. This output of the network is compared to the expected output of the input-output pair. If actual output and expected output do not match to each other then it indicates the presence of classification error. In the backward pass (the second stage of network training), the output error propagates backward from output layer to hidden layer to update weights along the interconnections between hidden layer and output layer. Now each hidden unit calculates an error, based on error from each output unit. This error then propagates backward from hidden layer to input layer to update weights along the interconnections between input layer and hidden layer. One training epoch passes when the network sees all input-output pairs in the training set.

Training stops when the sum squared error is acceptable or when a predefined number of epochs have been passed.



**Figure 2**

After training is over, weights of the network gets fixed and then network can act as a pattern classifier (see Figure 3). As a classifier, the network examines new input vectors for output vectors and predicts the fault in the network.

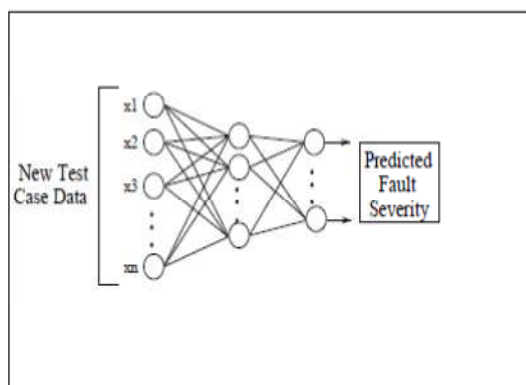


Figure 3

#### IV. Research Methodology: Prediction of Fault in Test Cases Using Neural Network

Once test cases are generated by genetic algorithm, we would like to evaluate whether it is exposed to fault or not. This decision is made by using neural network fault predictor. The neural network fault predictor compares the output of the application under test with the output of trained neural network and makes the decision whether the output is correct or incorrect, based on the network activation function. If both outputs do not match or there is a deviation between the actual output and expected output then it indicates the new version has errors (faults) and it do not confirm to the specification. If the output is correct then it indicates the new version is error or fault free. Figure 4 presents the overview of the proposed testing methodology.

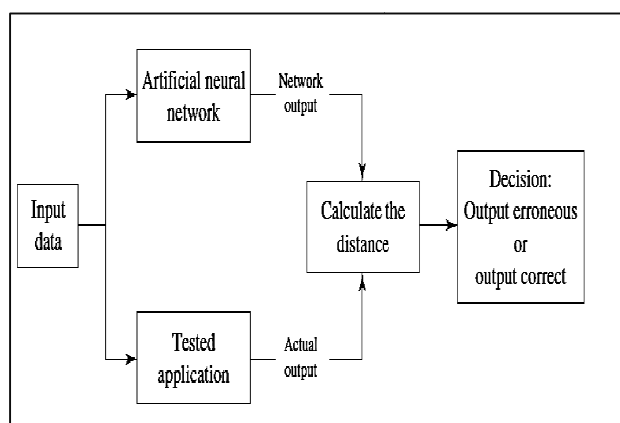


Figure 4

The Fault predictor is first needs to be trained for particular test cases and then it can be used to evaluate faults or errors in the software using oracles. Oracle reveals true errors whereas trained fault predictor reveals those errors which are predicted. Figure 5 shows how neural network fault predictor is actually trained.

*Metric calc.* is the test case metric that measure length of test case, command frequencies, and parameter use frequencies. An *oracle* is used to classify errors, exposed by test cases.

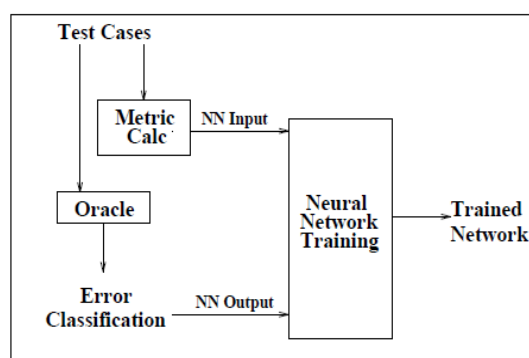
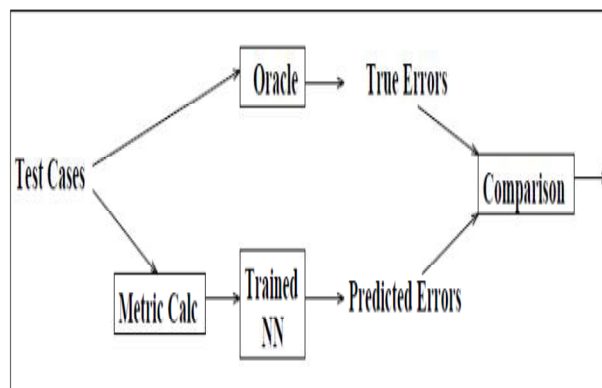


Figure 5: Fault Predictor: Training Phase

The “test case metric” is used by neural network fault predictor during training phase as Input pattern and “error classification” as Output pattern. Once network is trained, it can predict the fault exposure of new test cases. Figure 6 shows how we evaluate the effectiveness of the neural net fault predictor.

**Table Comparison of Accuracy for Random Testing Versus NN-Based Testing**

|                            | NN-based Testing | Random Testing |
|----------------------------|------------------|----------------|
| Overall Accuracy           | 81%              | 96%            |
| Error Accuracy             | 96%              | 76%            |
| Average per Error Accuracy | 83%              | 29%            |



**Figure 6: Fault Predictor: Evaluation Phase**

Given a test case, the “test oracle” identifies the actual faults exposed by the test case and the “trained neural network” predicts the fault exposure. Comparing the two, we measure how effective is the neural network as a test case fault predictor.

## V. Result

It has been observed that with the help of NN-based fault predictor, there is a drastic improvement in the software testing in terms of accuracy, cost and time. This can be shown with the help of following table:

## VI. Conclusion and Future Work

In this paper, we have used a neural network as an automated “fault predictor” for detecting the faults in the software application. We then used a comparison tool to evaluate the correctness of the obtained results based on the absolute difference between the two outputs. The neural network is shown to be a promising method of testing a software application provided that the training data have a good coverage of the input range. The backpropagation method of training the neural network is a relatively rigorous method capable of generalization, and one of its properties ensures that the network can be updated by learning new data. As the software that the network is trained to simulate is updated, so too can the trained neural network learn to classify the new data. Thus, the neural network is capable of learning new versions of evolving software.

The benefits and limitations of the approach presented in this paper need to be fully studied on additional software systems involving a larger number of inputs and outputs. However, as most of the methodology introduced in this paper has been developed from other known techniques in artificial intelligence, it can be used as a solid basis for future experimentation. One possible application can include generation of test cases that are more likely to cause faults. The heuristic used by the comparison tool may be modified by using more than two thresholds, or an overlap of thresholds by fuzzification. The method can be further evaluated by introducing more types of faults into a tested application.

## References

- [1] Goldberg D. “*Genetic Algorithms*”. Addison Wesley, 1988.
- [2] D. Ince. “*The automatic generation of test data*”. Comput. J., vol. 30, no. 1, pp. 63-69, 1987.
- [3] Roy P. Pargas, Mary Jean Harrold, and Robert Peck. “*Test-data generation using genetic*

- algorithms. Software Testing*". Verification & Reliability, 9(4):263–282, 1999.
- [4] DeJong K. A.: 'Learning with Genetic Algorithms: An overview', Machine Learning 3, Vol. 3, pp. 121-138, 1988.
  - [5] Morell L. J.: 'A theory of fault-based testing', IEEE Transactions on Software Engineering, Vol. 16, No. 8, pp. 844-857, August 1990.
  - [6] Voas JM, McGraw G. Software Fault Injection; 1998.
  - [7] Korel B.: 'Dynamic method for software test data generation', Software Testing, Verification and Reliability, Vol. 2, pp. 203-213, 1992.
  - [8] Watkins, A. L.: 'The automatic Generation of Test Data using Genetic Algorithms', Conference proceedings Dundee, 1995.
  - [9] Laurene Fausett. Fundamentals of Neural Networks. Prentice Hall: Englewood Cliffs, New Jersey, 1994.
  - [10] von Mayrhauser, A.; Anderson, Ch.; Mraz, R.; "Using A Neural Network to Predict Test Case Effectiveness", Procs. IEEE Aerospace Applications Conference, Snowmass, CO, Feb. 1995.
  - [11] Jacek M. Zurada. Introduction to Artificial Neural Systems. West Publishing: St. Paul, 1992.
  - [12] Haykin S. Neural Networks. A Comprehensive Foundation; 1999.
  - [13] Kirkland LV, Wright RG. Using neural networks to solve testing problems. IEEE Aerospace and Electronics Systems Magazine 1997.
  - [14] Alander J.T., Mantere T. and Turunen P. (1997) Genetic Algorithm Based Software Testing.
  - [15] Sherer SA. Software fault prediction. Journal of Systems and Software 1995.