

# **EXPLORATORY TEST ORACLE PREDICTION USING MULTILAYER PERCEPTRON NEURAL NETWORK**

A Thesis submitted in the partial fulfillment of the requirements  
for the award of the degree of  
**MASTER OF TECHNOLOGY**  
(INFORMATION SYSTEMS)

Submitted By:  
**WELLINGTON MAKONDO**  
(2K12/ISY/26)

Under the esteemed guidance of  
**Dr. N. S. RAGHAVA**  
Associate Professor



**DEPARTMENT OF INFORMATION TECHNOLOGY  
DELHI TECHNOLOGICAL UNIVERSITY  
BAWANA ROAD, DELHI-110042  
SESSION: 2012-2014**

## **CERTIFICATE**

This is to certify that work entitled “**Exploratory Test Oracle Prediction using Multilayer Perceptron Neural Network**” submitted by **WELLINGTON MAKONDO** (2K12/ISY/26), to Delhi Technological University, Delhi for the award of the degree of **Master of Technology** is a bonafide record of research work carried out by him under my supervision.

The content of this thesis, in full or parts, have not been submitted to any other institute or university for the award of any degree or diploma.

**Dr. N. S. Raghava**

Project Guide

Associate Professor

Department of Information Technology

Delhi Technological University

Shahbad Daultpur, Bawana Road, Delhi-110042

Date:-----

## **ACKNOWLEDGEMENT**

I would like to thank my project guide, Dr. N. S. Raghava for his valuable guidance and wisdom in coming up with this project. I humbly extend my words of gratitude to Dr. O. P. Verma, Head of Department, and other faculty members of IT department for providing their valuable help and time whenever it was required. I thank all my friends at DTU who were constantly supporting me throughout the execution of this thesis.

Special thanks to the Almighty Lord for giving me life and the strength to persevere through this work. Last but not least, I thank my family for believing in me and urging me. May you all be blessed.

Wellington Makondo  
Roll No: 2K12/ISY/26  
M.Tech (Information Systems)  
Department of Information Technology  
Delhi Technological University

## ABSTRACT

Software testing is aimed at evaluating a feature, or capability of a program and determine whether it meets its required results. Different software testing approaches have been practiced over the years to improve quality of software. Studies have shown that exploratory testing has become of great importance. The approach gained momentum as it is focused towards error detection in a timely manner and high possibility of testing complex combinations that cannot be designed in test case based techniques.

Traditionally, software testing is carried out by focusing on prior test design at the early stages of the software development life cycle, whereas exploratory testing is an approach where the tester does not require to follow a specific pre-designed tests. However, exploratory testing should enable the tester to test the complete system comprehensively. The exploratory testing approach relies on testers' skills and experience. The main claimed benefits of exploratory testing are the ability of the tester to apply personal knowledge and creativity while testing as well as agility in terms of adapting to changes and effective in working with imperfect documentation.

However, no emphasis was put on exploratory test oracle to help the testers. It remained the judgment and responsibility of the tester to determine whether a test executed passed or failed. In general, the exploratory testers designs and executes the tests on the fly and compares the actual output produced by the application under test with the expected output in their mind. Due to the fact that humans are fallible in nature, failures are not always detected by the exploratory testers even when revealed by the test case. This is one of the main drawbacks of relying on human knowledge.

Because of the lack of guidelines, there is a high possibility of judging a passed test as failed and vice versa. Humans are more prone to errors when evaluating complex behaviors or detailed specifications and the correctness of the human oracle drops precipitously with an increase in the number of tests to be assessed. Even if it were more reliable, the human oracle is prohibitively very expensive for enormous volumes of test cases.

Therefore, in this thesis efforts were made to explore the feasibility of using a multilayer perceptron neural network (MLP-NN) as an exploratory test oracle. The MLP-NN was improved by adding another weight on each connection to perfectly generate reliable exploratory test oracles for transformed different data formats.

# Table of Contents

<b>Certificate .....</b>	<b>i</b>
<b>Acknowledgement.....</b>	<b>ii</b>
<b>Abstract.....</b>	<b>iii</b>
<b>Chapter 1 Introduction .....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 Exploratory Testing .....	1
1.2.1 Procedures.....	2
1.2.1.1 Learning .....	3
1.2.1.2 Test Design .....	3
1.2.1.3 Test Execution .....	3
1.2.2 Characteristics .....	4
1.2.3 Management of Exploratory Testing .....	4
1.2.3.1 Free Style Exploratory Testing .....	4
1.2.3.2 Session Based Test Management.....	5
1.2.3.3 Session Based Testing Light.....	6
1.2.3.4 Sogeti Session Based Exploratory Testing .....	7
1.2.4 Combination with other testing approach .....	8
1.2.5 Adaptability .....	10
1.2.6 Suitability and non-suitability.....	11
1.2.7 Advantages.....	12
1.2.8 Disadvantages .....	13
1.2.9 Misconceptions .....	14

1.2.10 Challenges.....	14
1.2.11 Test Oracle.....	15
<b>Chapter 2 Literature Review .....</b>	<b>17</b>
2.1 Literature Review .....	17
2.2 Motivation of the Work .....	21
2.3 Aim of the Work .....	22
<b>Chapter 3 Proposed Methodology .....</b>	<b>23</b>
3.1 Introduction to Test Oracle Prediction.....	23
3.2 Architecture of Proposed Test Oracle .....	23
3.3 Description of Tool .....	25
3.3.1 Multilayer Perceptron Neural Network Parameters.....	25
3.3.2 Input Parameters of Application Under Test.....	26
3.3.3 Data Loading and Transformation.....	27
3.3.4 Feedforward MLP-NN .....	28
3.3.4.1 The Artificial Neuron.....	29
3.3.4.2 Activation functions.....	30
3.3.4.2.1 Sigmoid Activation Function.....	30
3.3.4.2.2 Hyperbolic Tangent Activation Function .....	31
3.3.4.2.3 Pure Linear Activation Function.....	31
3.3.5 Training Phase .....	32
3.3.5.1 Backpropagation .....	32
3.3.6 Testing Phase .....	36
3.3.6.1 Batch Testing.....	37

3.3.6.2	Manual Testing .....	37
3.3.7	Output Display .....	38
3.4	Summary of Exploratory Test Oracle .....	39
3.5	Performance Evaluation.....	39
3.6	Test Cases .....	40
3.6.1	Test Cases for Training.....	40
3.6.2	Test Cases for Testing.....	42
<b>Chapter 4:</b>	<b>Experimental Results .....</b>	<b>45</b>
4.1	Introduction.....	45
4.2	Multilayer Perceptron Neural Network .....	45
4.3	Modified Multilayer Perceptron Neural Network .....	46
4.3.1	Changing the Activation Functions .....	46
4.3.2	Changing the Number of Neurons in the Hidden Layer .....	50
4.3.3	Varying the Number of Iterations .....	54
4.3.4	Error Threshold Variations .....	56
4.3.5	Learning Rate Variations .....	61
4.4	Summary of Experimental Results .....	63
4.5	Exploratory Test oracle.....	64
<b>Chapter 5:</b>	<b>Conclusion and Future Work.....</b>	<b>66</b>
<b>References</b>	<b>.....</b>	<b>67</b>



## List of Figures

<b>Figure 1.1:</b> Exploratory testing paradigm .....	16
<b>Figure 3.1:</b> Proposed Exploratory Test Oracle Architecture .....	24
<b>Figure 3.2:</b> The Interface for Exploratory Test Oracle Prediction.....	25
<b>Figure 3.3:</b> MLP Parameters .....	26
<b>Figure 3.4:</b> Input Parameters of AUT .....	26
<b>Figure 3.5:</b> Load Dataset and Train the MLP-NN .....	27
<b>Figure 3.6:</b> Multi-layer Perceptron Neural Network .....	28
<b>Figure 3.7:</b> An Artificial Neuron .....	29
<b>Figure 3.8:</b> The Sigmoid Function.....	30
<b>Figure 3.9:</b> The Hyperbolic Tangent Function .....	31
<b>Figure 3.10:</b> The Pure Linear Activation Function.....	32
<b>Figure 3.11:</b> The Backpropagation Algorithm in MLP-NN .....	33
<b>Figure 3.12:</b> Two Weighted MLP-NN.....	36
<b>Figure 3.13:</b> Batch Testing .....	37
<b>Figure 3.14:</b> Exploratory Testing.....	37
<b>Figure 3.15:</b> Output Display .....	38
<b>Figure 4.1:</b> Output of a Common MLP-NN .....	45
<b>Figure 4.2:</b> Output of a Modified MLP-NN .....	46
<b>Figure 4.3:</b> Sigmoid Activation function in both Hidden and Output Layer.....	46
<b>Figure 4.4:</b> Sigmoid in the Hidden and Hyperbolic Tangent in the Output Layer .....	47
<b>Figure 4.5:</b> Hyperbolic Tangent in both layers - Hidden and Output .....	47
<b>Figure 4.6:</b> Hyperbolic Tangent in the Hidden Layer and Sigmoid in the Output Layer .....	48
<b>Figure 4.7:</b> Hyperbolic Tangent in the Hidden Layer and Pure Linear in the Output Layer.....	48

<b>Figure 4.8:</b> 1 Neurons in the Hidden Layer.....	50
<b>Figure 4.9:</b> 2 Neurons in the Hidden Layer .....	50
<b>Figure 4.10:</b> 3 Neurons in the Hidden Layer .....	51
<b>Figure 4.11:</b> 4 Neurons in the Hidden Layer .....	51
<b>Figure 4.12:</b> 5 Neurons in the Hidden Layer .....	52
<b>Figure 4.13:</b> 6 Neurons in the Hidden Layer .....	52
<b>Figure 4.14:</b> 7 Neurons in the Hidden Layer .....	53
<b>Figure 4.15:</b> 8 Neurons in the Hidden Layer .....	53
<b>Figure 4.16:</b> Testing done on 100 test cases .....	54
<b>Figure 4.17:</b> Testing done on 500 test cases .....	55
<b>Figure 4.18:</b> Testing done on 1000 test cases .....	55
<b>Figure 4.19:</b> Error Reduction with 1 Sample when Error Threshold is 0.1 .....	56
<b>Figure 4.20:</b> Error reduction with 60 samples when Error Threshold is 0.1 .....	56
<b>Figure 4.21:</b> Error Reduction with 1 Sample when Error Threshold is 0.05 .....	57
<b>Figure 4.22:</b> Error Reduction with 60 Samples when Error Threshold is 0.05 .....	57
<b>Figure 4.23:</b> Error Reduction with 1 Sample when Error Threshold is 0.01 .....	58
<b>Figure 4.24:</b> Error Reduction with 60 Samples when Error Threshold is 0.01 .....	58
<b>Figure 4.25:</b> Error Reduction with 1 Sample when Error Threshold is 0.001 .....	59
<b>Figure 4.26:</b> Error Reduction with 60 Samples when Error Threshold is 0.001 .....	59
<b>Figure 4.27:</b> Error reduction with 1 Sample when Learning Rate is 0.1 .....	61
<b>Figure 4.28:</b> Error reduction with 1 Sample when Learning Rate is 0.2 .....	61
<b>Figure 4.29:</b> Error reduction with 1 Sample when Learning Rate is 0.3 .....	62
<b>Figure 4.30:</b> Error reduction with 1 Sample when Learning Rate is 0.4 .....	62
<b>Figure 4.31:</b> Expected Passed Test .....	64
<b>Figure 4.32:</b> Expected Failed Test.....	64

## List of Tables

<b>Table 1.1:</b> Preference of using ET approach in industry .....	9
<b>Table 3.1:</b> Combination of Activation Functions .....	39
<b>Table 3.2:</b> Specifications of the AUT .....	40
<b>Table 3.3:</b> Test Cases for Training.....	41
<b>Table 3.4:</b> Test Cases for Testing .....	42
<b>Table 3.5:</b> Summary of testing and training datasets.....	45
<b>Table 4.1:</b> Evaluation of Activation Functions .....	49
<b>Table 4.2:</b> Summary of MLP Outputs for Various Neurons in Hidden Layer .....	54
<b>Table 4.3:</b> Summary of Effects of Error Threshold of 1 training sample. ....	60
<b>Table 4.4:</b> Summary of Effects of Error Threshold of 60 training samples. ....	60
<b>Table 4.5:</b> Learning Rate Effects on Epochs. ....	63

# **CHAPTER 1**

## **INTRODUCTION**

## 1.1 INTRODUCTION

In software development life cycle, software testing has always been a crucial phase. The previous studies have witnessed many software project failures that result from lack of proper testing. The reputation of the company is easily broken by a poorly tested software. Better tests and software are achieved when there is a proper relationship between testing and requirements [1], nonetheless what if the developed system requirements are not explicit and frequently changes. Consequently, testing approaches and techniques were developed to mitigate these shortcomings [2].

Test case based approach was the most commonly used, which is based on the requirements, test cases are designed and later executed. When changes are made in the requirements, the test cases need to be updated too, but the updates may not be effected in time or even forgotten. Therefore, this approach may not prove to be efficient. In a typical software development project more time and cost are typically consumed [3]. Since there is a trend to produce complex software in a limited time frame while maintaining quality, many software practitioners believe that a shift toward agile development methodologies will help in dealing with the constant requirements inflow. When the flow of requirements is parallel to the development phase, the systematic test case based approach is considered difficult to use owing to the constant requirement change while designing test cases [4]. Hence, an approach such as exploratory testing (ET) can be worthwhile for such environments, where the requirements collection has not been finished early in the software development process.

## 1.2 EXPLORATORY TESTING

ET is also referred to as ad-hoc testing [5], which does not dependent on the test cases that were designed earlier in the software development life cycle. In modern days, this approach has gained attention and reputation, particularly among professional software testing practitioners [3]. The ET term was initially coined by Bach and Kaner in 1983 and it was discussed as one of the approaches adopted by many skilled professional testers in the industry [6]. Although the approach gained familiarity lately, it was claimed to be first recognized by Myers [7]. Myers's book revealed that, "people are

subconsciously practicing it, more often than not, it is a test-case-design technique that takes the form of error guessing.”

Today, most researchers defined ET as an activity where a tester learn, design and execute the tests simultaneously [5]. Generally, the testers explores the application software, learns its functions and executes it based on their intuition. There is no detailed systematic approach followed, as in a scripted test case document that guides the tester to execute the tests. The tester himself controls the test designs while executing and learning the software. This helps to build effective tests while exploring unrealized parts of the software. The tester has to identify the defects using any possible combination of methods that forces the software to misbehave. This approach is called destructive testing; which is a testing approach that seeks errors which want to destroy the system and then check whether the system can resist these destructive errors or not.

ET is a proactive approach for detecting the software defects, but the practice still needs to be codified. Also, it put more emphasis on the expertise of the testers, meaning that there is a need to find out what skills are required for testers to practice ET approach. One major trait of an exploratory tester, is knowledge about the behaviour of the software and this knowledge differs from one tester to another, depending on their prior understanding of the application [5].

### **1.2.1 PROCEDURES**

Exploratory testers actively control the test designs and uses information obtained during testing to design better and new tests [8]. Exploring means focused navigating, wandering with a general assignment through a space without a pre-defined route. Workshops on exploratory testing were conducted and it was proposed that exploratory testing should be done in pairs [9]. Thus, everything being tested is peer reviewed and hence increases productivity. This technique also facilitates training for naive testers ensuring that beginners learn by testing with experienced testers.

Jonathan Bach focused on defect finding [5], while, less emphasis was put on test documentation and optimized approach for detection of defects. The main purpose of ET is to find defects, at the same time exploring the software functionalities. Documentation of test results is more important in ET than scripting and planning the

test execution pathways. Chartered exploratory testing was suggested by Copeland. The charter defines what should be tested, what types of defect to look for and the risks involved. It works as a guide and defines a mission for testing, but without the proper test steps. In ET, a tester can expand test execution into productive areas, as emphasized by Craig and Jaskiel [9].

The differences between exploratory and scripted testing are stated in [10]. Scripted testing is guided from software requirements which are determined in advance. ET is a process explained by performing learning, test design and test execution simultaneously.

### **1.2.1.1 LEARNING**

Learning encompasses anything that can lead to what to test, how to test and how to detect defects. ET involves continuous learning through experimenting with the program under test. In order to build new effective tests, a tester should continuously learn about a product, a product's previous defects, its market and risks associated with the product. Based on the increasing knowledge of the tester, the new tests will be more effective than the previously executed ones.

### **1.2.1.2 TEST DESIGN**

Test design involves construction, creation, or fashioning according to plan, as conceived in the mind. The design of a test is perceived in the tester's mind and guided by the learning aspect drawn from various sources. It is the representation of a plan in the mind and is not scripted.

### **1.2.1.3 TEST EXECUTION**

Is performing the actual test and collecting the results. Execution of test cases can be manual or automated. When execution is manual, the tester plays the end user's role to find the defects. In the case of automated execution, the test datasets are provided first before execution begins. There is little human intervention, when execution begins, it stops when all tests are executed without human interruption.

## **1.2.2 CHARACTERISTICS**

The following exploratory testing properties have been derived by Itkonen [11]

- Tester's knowledge, experience and skills is directly related to testing effectiveness.
- Focuses on defect finding by exploration, as opposed to prior defined test cases.
- It includes any document, i.e. user manual, marketing brochure of target system under test used by ET testers and previously performed test execution results. Knowledge obtained guides the approach.
- No detailed test scripts are defined.

## **1.2.3 MANAGEMENT OF EXPLORATORY TESTING**

The nature of ET approach makes it difficult to determine what has been tested or not and to track what modules have been tested by an individual tester. Therefore, to deal with these challenges, exploratory testing management approaches were proposed, namely;

- Free Style Exploratory Testing
- Session based test management
- Session Based Testing Light
- Sogeti Session Based Exploratory Testing

### **1.2.3.1 FREE STYLE EXPLORATORY TESTING**

The purpose of free style ET is to detect the defects and report them at the end of the testing phase. Free style ET is focused on walking through the application under test without any predefined test plan. The test lead allocates the charters to the testing team for execution and reporting the results of the charters.

Management is done by delegation [5]. Each tester becomes responsible for managing the tests. By implementing this way the testing process becomes traceable. The regular meetings to discuss progress and problems encountered, can be done to make the process more controllable [8]. Management of ET by delegation aids to assess



performance of individual tester. Understanding the capabilities of testers involved in the testing process, helps when assigning the tasks.

Another way of managing free style ET [5] is by participation. The test lead also participates in the testing process like any other team members. The test strategies are continuously directed by the test lead and conveys his expectations to the testing team members, hence boosts up the efficacy of the whole team. Also prompt decisions are done during testing by active participants to help the lead.

### **1.2.3.2 SESSION BASED TEST MANAGEMENT**

ET can be managed, planned, structured and disciplined provided that the test plan does not provide details of the test cases and execution steps [10]. Session based test management (SBTM) [5] is a clear structure to manage, control and plan ET in short fixed length sessions. The test sessions are planned on a charter sheet. Each session has a charter or mission to accomplish. The test cases executed are not pre-defined, but after the session, a clear report is produced.

In SBTM approach, 90 minutes was the suggested time frame, but it is not compulsory since the strict time frame restriction can affect quality of testing [5]. The setup time and bug reporting time are not included in the time frame, although they are related to the testing process. What is being done and accomplished can be traced by keeping a record of these sessions and missions achieved. It becomes easier to predict the number of sessions needed to test the entire system. Therefore, the complete testing cycle progress can be estimated which is useful to approximate the entire testing duration.

The report of the test session is presented, discussed and concluded in the debrief meeting. Testers fill the session sheet which encloses the details of each session which serves as a report for the test session. It involves reporting:

- Bugs,
- Notes, and
- Issues.

*Bug:* The software defects and quality threats discovered.

*Notes:* Findings, Suggestions, potential risks that are noted.

*Issues:* Difficulties related to the test session.

The session sheet is used for creating data for management and for the analysis of the progress of the testing process.

Another metric of charter vs. opportunity [5] was suggested by Jonathan Bach. It reveals the time spent on bugs described in the charter as well as time spent on issues. Such issues are not part of the charter, but requires attention. The report structure of the final session is as follows; Date, Mission of the session, Tester name, Task break down Metrics.

### **1.2.3.3 SESSION BASED TESTING LIGHT**

Session Based Testing Light approach [13] was proposed by Kalman to avoid the shortcomings of the original Session Based Test Management approach. The following points were considered as shortcomings;

- Extra training,
- Additional reporting, and
- Report Analysis.

*Extra training:* Testers require training when testing complex or diverse applications.

*Additional reporting:* Session reports results as more time is spent testing, hence, affects the performance of the tester.

*Report Analysis:* More time is also spent analyzing the reports, which results in inadequate time for actual testing.

Due to these disadvantages, Kalman proposed to have two different components, namely, reporting and project components. Each component summarizes its related benefits and costs which maximizes the benefits and minimizes the cost of the project.

Another similar approach was presented, which included the ways to control the testing scope, risk assessment and tests prioritization. They applied the concept of ‘test points’ [6]. The notion of test points is referred to as a single unit of work and each session as a unit of time. Test points are utilized for coverage features of testing and scope management. Test points are further used to monitor the test progress in a unit of time.

In each test session, the data is gathered as follows [14]:

- Risk
- Description
- Time spent
- Estimated time of completion
- Estimated essential testing percentage concluded

*Risk:* The risk associated with the test session.

*Description:* Test session mission description.

*Time spent:* Time consumed in the session.

*Estimated time of completion:* Approximate time to complete the session.

*Estimated essential testing percentage concluded:* Approximate percentage of the important aspects to be tested.

Software testing practitioners are also applying this approach to increase control of the ET process.

#### **1.2.3.4 SOGETI SESSION BASED EXPLORATORY TESTING**

Sogeti Session Based Exploratory Testing (SBET) framework has three main phases which summarizes all session related activities. The three phases are:

- Test Session Planning,
- Test Session execution and
- Test Session Controlling.

*Test Session Planning:* Defines the project objectives, the number of testers, their roles and responsibilities involved in the session, time required, mission of the session and risks associated with the product.

*Test Session execution:* Components to be executed.

*Test Session Controlling:* How the session should be controlled.

Some overlapping activities are in phase two and phase three [6]. The goals are also formulated and categorized into classes during risk assessment of the product. The class determines the priority of the test that is, Risk class C is the lowest and Risk class A is the highest. Likewise, the test strategy is aimed at covering high risk classes timeously in the test project [6]. A successful ET sessions were conducted using a SBET

framework. The framework can be frequently used for each test session [6] with a unique mission. A testing project can have numerous test sessions.

There are differences between SBET [6] and SBT [5]. SBT process lacks traceability, whereas in SBET framework the traceability is visible. SBET framework has also a customized tool for control, support and track the ET sessions. The system log continuously logs all testers' actions in the database. It makes reproduction of bugs ease, if required, whereas, in SBTM session sheets there is no extensive logging of entire ET session. ET tool is used by SBET and LogCat for tracking, controlling and reporting test sessions. The tool used by SBTM was developed in Perl. It produces heterogeneous tables and metrics which result from scanning the information on the session sheet [5].

## 1.2.4 COMBINATION WITH OTHER TESTING APPROACH

ET is considered as a situational practice [5] which depends on certain aspects e.g. Complexity, customer requirements, and nature of software. Based on these aspects mentioned, other testing techniques can be combined with ET [6].

**Table 1.1:** Preference of using the ET approach in industry [6]

Preference	Percentage
In combination with other testing techniques	76%
As complimentary	48%
Using other techniques in an exploratory manner	30%
Solely exploratory	22%

ET is used in combination with other testing techniques in industry [6] which includes;

- Risk Based Testing (RBT),
- Cross functional testing,
- Requirements based testing,
- Pairwise testing,
- Acceptance testing,
- Checklist based testing,

- Scripted Testing,
- Usability Testing,
- Security Testing and
- Boundary Value Analysis (BVA).

*Risk-based testing* is a type of software testing that functions as an organizational principle used to prioritize the tests of features and functions in the software product, based on the risk of failure, the function of their importance and likelihood or impact of failure.

*Cross functional testing*: Performing Functional Testing with multiple approach for example, based on the task, use-case, different scenario, etc. It helps to detect the bugs earlier; same time in this approach you can find lot of accidental, unexpected bugs.

*Requirements-Based Testing* (RBT) delivers a proven, rigorous approach for designing a consistent and repeatable set of highly optimized test cases. Companies employing RBT practices have achieved twice the requirements coverage with only half the tests they previously maintained.

*Pairwise testing* is an effective test case generation technique that is based on the observation that most faults are caused by interactions of at most two factors. Pairwise-generated test suites cover all combinations of two factors and yet still be very effective in finding defects.

*Acceptance testing* is a test conducted to determine if the requirements of a specification or contract are met. In systems engineering, it may involve black-box testing performed on a system prior to its delivery. Software developers often distinguish acceptance testing by the system provider from acceptance testing by the customer prior to accepting transfer of ownership. In the case of software, acceptance testing performed by the customer is known as user acceptance testing (UAT).

*Checklist based testing* is used by experienced testers who are using checklists to guide their testing. The checklist is basically a high-level list, or a reminder list, which lists areas to be tested. This may include items to be checked, lists of rules, or particular criteria or data conditions to be verified. Checklists are usually developed over time and draw on the experience of the tester as well as on standards, previous trouble-areas, and known usage scenarios. Coverage is determined by the completion of the checklist.

*Scripted Testing* follows a path that is written by the tester themselves or someone else. The script includes test cases and test steps that are documented. There can be no deviation from the path laid out in the script. The tester's job in a scripted environment is to follow each instruction to the nth degree and report on the findings.

*Usability Testing* is a technique used in user-centered interaction design to evaluate a product by testing it on users. This can be seen as an irreplaceable usability practice, since it gives direct input on how real users use the system. This is in contrast with usability inspection methods where experts use different methods to evaluate a user interface without involving users. Usability testing focuses on measuring a human-made product's capacity to meet its intended purpose.

*Security Testing* is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders. Typical security requirements may include specific elements of confidentiality, integrity, authentication, availability, authorization and non-repudiation. Actual security requirements tested depend on the security requirements implemented by the system.

*Boundary Value Analysis* recognized that input values at the extreme ends of input domain cause more errors in system. More application errors occur at the boundaries of input domain. 'Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

RBT and ET provides the most effective combination because of its easiness when deciding the test mission in ET. It is more focused on the removal of all entitled risks and it can be referred to as focused testing [6].

### **1.2.5 ADAPTABILITY**

ET is a situational exercise that can be adapted relative to the required results [6]. ET is appropriate in a situation where testing requirements are;

- Learning of system
- Perform complementary testing
- Testing is under time constraints
- Targeted testing
- Test driven development

It is difficult to design test cases for all possible combinations of the inputs, but with ET one can explore, design and run tests simultaneously. In this situation ET would be more effective and efficient to find bugs. When tester has little knowledge about the product area, ET can be adopted for learning the product area. When software solution is unknown from the start, ET also can be adapted for testing of system [6]. Exploratory testing is a good approach to learning legacy systems [6].

The following are test requirements or situations where exploratory testing can be adapted when;

- Testers have more experience and knowledge of the product and testing
- Other testing methods no longer yield any important bugs
- There is limited time available for testing
- The product is too complex
- There is limited knowledge about the product under test
- Vague or insufficient documentation or requirements
- Testing is performed for non-functional and functional features
- The system is constantly changing and is unstable.

### **1.2.6 SUITABILITY AND NON-SUITABILITY**

Exploratory testing is most valuable where results of earlier executed tests are required for the subsequent tests to be performed since it is not possible to determine them early. Ad hoc testing can be useful to explore the scope, size and disparities of defects found so as to give better feedback to developers.

The following different contexts describe where ET would fit very well:

- Rapid feedback
- Learning of product is needed
- Inadequate time available for systematic testing approaches
- Examine the status of particular product risks
- Provide more information in addition to scripted tests
- Testing from the perspective of an end user view point

There are certain circumstances in which exploratory testing is not appropriate for some types of systems. It is not suitable in the following systems [6].

- Financial systems, e.g. Banking systems

- Critical systems, e.g. Medical systems
- Scientific systems, e.g. Nuclear reactor systems
- High risk systems, e.g. Aviation related systems

The reason why exploratory testing is not applied in the above mentioned systems is because they are of very critical nature. These systems require proper or well-structured testing and none of their areas should be left untested [6]. Such systems require rigorous testing and complete documentation plays a very crucial role.

### **1.2.7 ADVANTAGES**

The following advantages of ET are discussed in [6], [8], [11].

- Little test preparation
- Simultaneous learning
- Rapid feedback
- Effectiveness
- Adaptable to project situation
- Tester's utilization

*Little test preparation:* No extensive pre-test planning and documentation are required. More time and effort is saved which can be utilized in other testing activity.

*Simultaneous learning:* ET facilitates the simultaneous learning of the product under test, which is learning about the product, its weaknesses and how the system fails. Test case creation requires learning the system first.

*Rapid feedback:* The progress and status of product under test can be quickly determined by testers and developers.

*Effectiveness:* The anecdotes provided by [3] supports ET as compared to scripted testing. ET was found to be more effective in detecting significant defects. The case study [3] shows that ET results in less false defects as compared to scripted testing. Defects that are difficult to detect were revealed by ET.

*Adaptable to project situation:* The approach suits well to rapidly changing product requirements since there is no documentation preparation required earlier to start testing.



*Tester's utilization:* Usage of testers' experience, knowledge and abilities is the greatest advantage of ET. It makes testers provide valuable involvement in the testing process [6].

Focused testing provides the biggest advantage from the customers' perspective. In the early stage of software development, when frequent changes are taking place, ET proves to be a more effective testing approach [6].

### **1.2.8 DISADVANTAGES**

There are numerous misconceptions associated with ET that includes understanding of ET itself, as an entirely ad-hoc approach to test; this is considered to be among its major drawbacks. In the next section misconceptions are stated in detail.

The research study of Itkonen [11] stressed on various findings concerning the disadvantages of ET, as being too dependent on knowledge and ability of the tester. Hence, it is reflected in literature that testing activities are more prone to human errors as compared to systematic testing approaches. However, systematic testing is also considered to be prone to human errors, because the test design is prepared by human and equal chances exist to commit some sort of errors. While exploring a system using ET approach, such defects are more likely to be detected.

It is rare to find testers with appropriate domain knowledge, skill set, and experience. Also ET results are greatly depend on the knowledge and experience of the testers [3], [5], [6], [8], [11].

It is difficult to trace the errors or reproduce the bug with the ET approach. This issue can be solved by creating more log files of every activity during the testing process. It becomes easier and helpful to report issues or defects with full details describing how they were found or generated. The bugs can be reproduced using this information [6].

It is hard to trace the progress made by each tester in ET [8], [11] and finding out how the work proceeds [3]. In [6], reveals that it is because of the fact that ET is not a technique for testing but an approach. The techniques are easy to handle in a structured manner, while approaches can be seen with different perspectives and adapted accordingly [6].

Prioritizing what should be tested is difficult when there is a time constraint [11]. Adequate testing can be done with the help of the expert testers. It is hard to judge the testing process because ET does not make available decision material and therefore, unavailability of ET expertise is considered the main disadvantage from the customer's viewpoint [6]. A question like, how much testing is completed, can be responded certainly with structured testing as compared to ET approach [3], [16].

### **1.2.9 MISCONCEPTIONS**

Generally, the prime misconception was considering ET as a technique. Lack of guidance and inadequate experts available in the market poses challenges to adapt ET, when testers are already used to traditional techniques [6].

ET is unstructured, unrepeatable and unaccountable and this makes it to be unsatisfactory to both the manager and customer. Its results are not considered reliable [6] because of these. Hence, some software firms did not adopt it [8]. A lot of experience and skills are required for producing good results. Customers start accepting the results of ET when they observe that ET identified critical bugs [6].

The test coverage problem that everything cannot be tested is also reported as weakness in [11]. Visibility of the testing process, progress, test coverage, decision material and work products are pointed out as misconceptions from the customer's viewpoint [6].

### **1.2.10 CHALLENGES**

An experiment was conducted to compare the defect detection efficiency in test-case design based with ET approach [16]. While conducting the experiments, different challenges were identified by the participants and a further validation of these is needed.

These challenges were classified in the following categories:

- Time Limitation
- Test Design
- Test Preparation
- Defect Recognition
- Test Logging
- Tester's Skills
- Feature Coverage

- Defect Reproduction

*Time Limitation:* A major constraint is the time to test the software. More time is required to test the complete functionality of the software. The time allocated to test the test object was impractical as compared to the functionality of the software to be tested. This results in insufficient exploration of different features of the software [16].

*Test Design:* The cognitive nature of ET is another mentioned challenge [16]. It was difficult for the subjects to execute tests simultaneously while designing the tests. Testers find it difficult to go through the function in detail without a pre-designed test cases. It is necessary to motivate testers to explore the system so as to detect maximum defects without using the pre-designed test cases, but it is difficult to have a good test design parallel to test execution.

*Test Preparation:* Testing the software features without test cases was tough for the subjects that are unfamiliar with the test object beforehand, more time is consumed in learning the software. It is challenging to acquire complete knowledge about an application while executing tests. Insufficient detail may lead the tester to wrongly classify the software defects and intended software functionality [16].

*Defect Recognition:* Because of lack of previous knowledge about the software functionality, it is difficult to find defects and classify them as faults in comparison to the required system functionality. In the experiment that Itkonen has conducted, the testers were reported to have encountered many trivial errors that they did not classify as defects [16].

*Test Logging:* It is not easy to log tests that have been executed [16]. Logging the tests consumes significant time and a lot more writing even when there are no defects found. It is hard to log each action while the tester explores the application. Defining the extent to which the test actions are to be logged is another notable challenge. Less logging affects the possibility of reproducing the identified defects [16].

*Tester's Skills:* The knowledge and experience needed for the ET tester should not be overlooked. Applying ET can be so challenging, if the tester is unfamiliar with the software under test [16]. ET requires experienced and skilled testers of relevant domain with appropriate knowledge of vulnerable areas and potential defects of the software.

*Feature Coverage:* When practicing ET, it is not easy to ensure complete coverage of the entire software features.

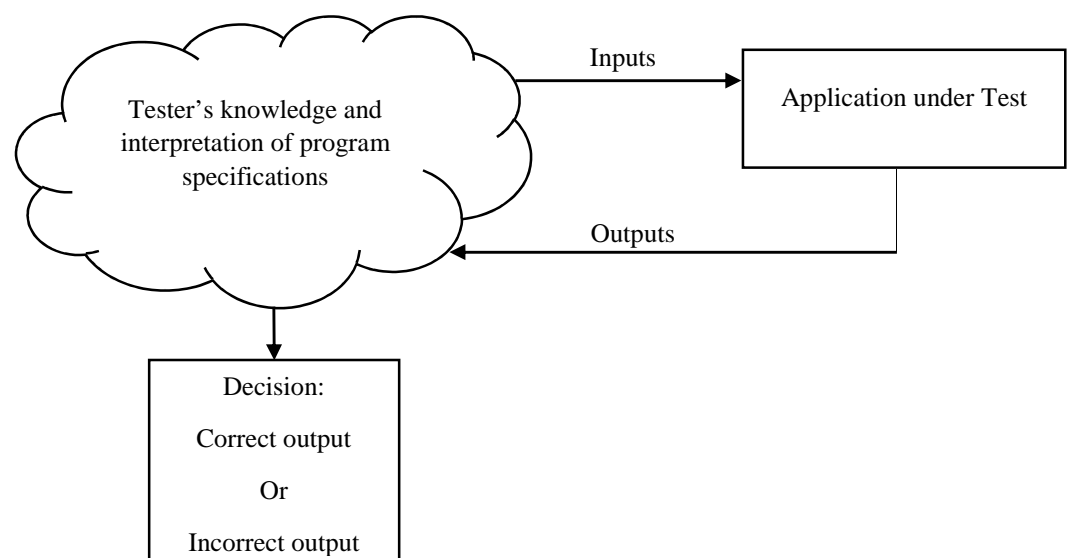
*Defect Reproduction:* This relates to test logging challenges. Reproducing the defects in ET is found to be difficult and it is not easy to analyze the cause of the observed failures while exploring the software [16].

### 1.2.11 TEST ORACLE

A test oracle is a mechanism frequently used to determine the success or failure of a system test. Some define it as “a reliable source of expected outputs” and it is used for verification of the test case results. Techniques of software testing depend on the test oracle availability [17].

In the context of ET, the test oracle is represented by human knowledge and intelligence. A test oracle separates a correct result from an incorrect result [18]. Finding a trusted or reliable oracle is a challenge known as “the oracle problem.” The failure identification is left as a human tester’s judgment [19]. Therefore, in ET the oracle problem is very much relevant and normally solved by applying tester’s personal knowledge, skills and various documents. Due to the fact that humans are fallible in nature, failures are not always detected by the exploratory testers even when revealed by the test case. This is one of the main drawbacks of relying on human knowledge.

A perfect test oracle should deliver a correct decision for a program execution. Generally, ET testers check the results using their personal experience in order to decide whether the application under test does what it is expected to do or not as in Figure 1.1.



**Figure 1.1:** Exploratory testing paradigm

# **CHAPTER 2**

## **LITERATURE REVIEW**

## 2.1 LITERATURE REVIEW

The literature on test oracles is a comparatively small part in software testing. A test case comprises of a set of inputs and their corresponding expected results for the AUT. Automatic generation of test inputs is reasonably stress-free, but it is quite difficult to generate the expected results. Confidence can be eradicated to execute extensive automated testing without the expected outputs. It can be a software specification or simply the knowledge of the programmer that verifies how a program should operate [17]. A perfect oracle would behave equally with the AUT and absolutely trusted. All the inputs specified for the AUT would be accepted and the correct results would be produced [17].

Expected outputs were generated by table of pairs [20] and relational models when the behaviour of the software is provided in tabular expressions. The formal descriptions were represented as logical expressions fulfilled by the software [21]. A temporal constraints based tableau algorithm was described that must not be violated, which executes the software [22]. The multilingual and multi paradigm specifications were provided for reactive systems in [23]. Evaluation of test results based on the expected outputs produced by a test oracle is extensively recognized as a vital aspect in the process of testing. There are numerous ways of developing test oracles using specifications, parallel implementations and documentation [23].

Derivation of test cases from a Z specification was applied as a model-based specification to define a template for the test framework [24]. Test oracle generation can be derived from formal models like formal specifications and relational program documentation [25]. The reduced set of expected outputs was generated by I/O relationship analysis [26]. An analysis of I/O was done to realise the outputs affected by the inputs. Reduced test cases set was then created manually. Finally, the expected outputs were produced on the basis of the reduced set of test cases and also generalized to make the remainder of the test cases available automatically.

A partial test oracle [27] verifies the correctness of the actual outputs in XML data. To deal with the actual outputs, the partial oracle was used to assess the accuracy of the test executions according to (1) a loose specification and (2), a set of predefined

software constraints. An automatic response about the accuracy of the AUT is given by the oracle with a certain degree of precision. A passive test oracle [28] does not reproduce the behaviour of a component but only checks its behaviour.

Wrapper approach [29] is a simple method used where the AUT is placed in a wrapper that is responsible for the behaviour checks. A similar syntactic interface is provided by a wrapper to give the results of the actual implementation. Other test oracles methods exploited the use of pattern recognition and artificial intelligence techniques. In a typical pattern recognition-based oracle implementation, the unit emulates the human tester's knowledge, intelligence and decision making process [30]. The oracle imitates an accurate tested module and measures the distance between the output patterns of oracles and tested module outputs.

The process of automating testing of an application system called test automation, involves generation of the inputs and expected outputs, running test suites with no manual intervention and assess the results [31]. The state machine analyses the log file to assess the SUT behaviour. A perfect oracle provides absolute pass or fail decision for any possible execution which is checked against the specifications of the intended behaviour [32]. Random Testing selects the test cases randomly to automatically match an operational profile [33], but requires other approaches to provide the expected outputs. Other methods as Boundary Value Analysis and Equivalence Partitioning [33] are not discussed as test oracles, but are rather more test case generation methods. The effectiveness of random testing is not sufficient even though the cost is significantly low [34].

I/O mapping test oracles are popularly addressed by Decision Tables and Cause-Effect Graphs [35]. Despite the tools used to produce the required structures based on software specifications, programs were written to generate the oracles automatically. To achieve the best test oracle, human observations and improvements are still necessary.

In [36], model transformation testing was discussed and how its oracle is provided. To generate oracle functions they used an example based technique. The more a transformation diverges from well-known examples of transformation, the more chances of being incorrect [37]. To put it otherwise, the oracle matches the base examples with the expected results and then gives a level of risk.

Genetic Algorithms were engaged to provide oracles for testing condition coverage [38]. Manual searching for the test cases to maximize condition coverage is challenging if the SUT has voluminous nested loops. Hence, an automated approach to effectively generate test cases covers more conditions. An automated Dynamic Test Generator using Genetic Algorithms [39] was introduced to identify test cases that are effective. The dynamic test generators inspect the internal structure of the SUT for collecting information and then use it for testing process optimization. The approach is unreliable for testing complex programs with multiple nested loops and it was platform-dependent. Artificial intelligence [40] was applied to auto-generate GUI expected outputs. The GUI actions and elements, the internal behaviour of the GUI was modelled so as to mine the normal state of the GUIs when executing every test case. Formal models consist of specifications and objects of the GUI which are derived from its attributes. Preconditions and their effects defines the actions of a GUI and uses the model to automatically generate expected states and the actions taken from the test cases.

Artificial neural networks were applied to automate oracle generation to test decision making structures [41]. A training dataset generated on the basis of software program specifications was used to model the decision rules. Then, a trained neural network was used for testing a software program, mutation testing was used to evaluate the approach. The oracle data was selected automatically to monitor the variables while testing to determine the expected value. In order to rank variables, mutation analysis was used to effectively find faults, therefore automatically select oracle data. It is an inexpensive method for producing small and effective oracle data [42].

Metamorphic testing [43] uses metamorphic relations and properties of the SUT represented in the form of relations amongst the inputs and the outputs of multiple executions to verify the accuracy of the program. Sufficient number of the appropriate metamorphic relations are identified even by novice testers with minimum training. Besides, it is cost-effective and can be improved using more diverse metamorphic relations.



Currently machine learning techniques [44] have been employed in the software testing field. However, the algorithms are difficult to discover the faults in certain applications as it is hard to find the test oracle for comparing the outputs for verification. A novel functional testing approach was attempted to verify the test oracles. The expected output for a given application is produced and verified by the oracle whether the SUT behaved correctly or not and issues a pass or fail verdict. A model-based method for generating oracle for unit testing was presented [45]. A fault model was developed based on the features of the core units to record the types of defects that may be encountered, and define how to generate automatically a passive, partial oracle from the agent design models.

Search-based test generation method can produce large volumes of test inputs automatically, even though, it is tough to define the test oracle for every test input. A mining approach [46, 47] was presented to build a decision tree model according to the generated test inputs from the Java bytecode, which was converted into Jimple representation. The resulting extracts predicates of the Jimple code are used as attributes for organizing training data to form a decision tree. Faults were injected to assess the performance of the technique. Artificial neural network [48] was presented for automatic generation of oracles. Also, heuristic test oracle fit for test automation was included. Both predication and classification abilities of the artificial neural network were appropriate for the generation of test oracles.

It is error prone and expensive to develop test oracles for software application with many test cases. To stand the test oracle difficulties, the semantic model of the GUI service specifications of the SUT was presented [49]. The service data domain knowledge capabilities and limitations are well defined by the model. Rules created simulate the expected behaviour of the SUT. Direct I/O relationships and interactions of service were captured for each service. Oracles of this nature are SUT implementation independent [49].

Backward-Slice-based Statistical Fault Localization (BSSFL) is a fault localization technique which assesses the statement suspected of a being defective by statistically analyzing the backward slices and the test cases results [50]. The test oracle is assumed to be in existence so as to decide the test case execution result as a pass or fail. BSSFL

can be impossible when test oracle is unavailable. As metamorphic testing has been extensively researched as a method to lessen the problems of the oracle. Therefore, metamorphic testing was used to apply BSSFL with the absence of test oracles. The result of execution of a test case is not required in this approach. Therefore, in application domains where no test oracle exists BSSFL can be used [50].

In [51], program comprehension techniques focused on the supportive building of the human understanding for an earlier unknown program and help the process of creating the test oracle. Machine learning techniques were used for automatic construction of the test oracles for reactive systems without explicit specifications. The intelligent Test Oracle Library was presented [52] to appropriately collect test traces. Program assertions or user guidance was used to collect the results of traces of the tests.

## **2.2 MOTIVATION OF THE WORK**

The motivation for this thesis done in ET, is its exponential adoption by many software development firms due to several benefits it provides [6], [8], [11]. Although a number of benefits motivated us to carry out research in this area, the quality of the software under test is highly dependent on the ability of the tester. The exploratory tester may not properly understand the requirements and specifications of the software and consequently endangers the quality of the product by judging correct results as wrong or wrong results as correct. Sometimes the tester may not be able to conclude whether the result is correct or not.

The question of how to improve the exploratory tester judgement or decision making is very crucial as a guideline to increase the quality of the product. In this thesis, we investigate the following issues:

- What ways can help the exploratory testers to determine whether the test execution result is correct or wrong?
- Can a multilayer perceptron neural network be applied as a test oracle to help exploratory testers?

### 2.3 AIM OF THE WORK

Our research goal is to use multilayer perceptron neural networks in a simulated application under test to predict the test oracle of the executed tests. Multilayer perceptron neural networks have not been applied to deal with different data formats in software development to facilitate learning and make decisions on test combinations executed.

In our study, we used multilayer perceptron neural network and evaluated different parameters which gives good results. Although not explicitly stated in the studies, partial knowledge of the internal functionality of the application under test need to be possessed by the exploratory testers. Testers should know which data types correspond to the input fields, their min-max length and integer valid ranges for the neural network to predict correctly. The results of the neural network allow the exploratory tester to judge the test results correctly.

# **CHAPTER 3**

## **PROPOSED METHODOLOGY**

### **3.1 INTRODUCTION TO TEST ORACLE PREDICTION**

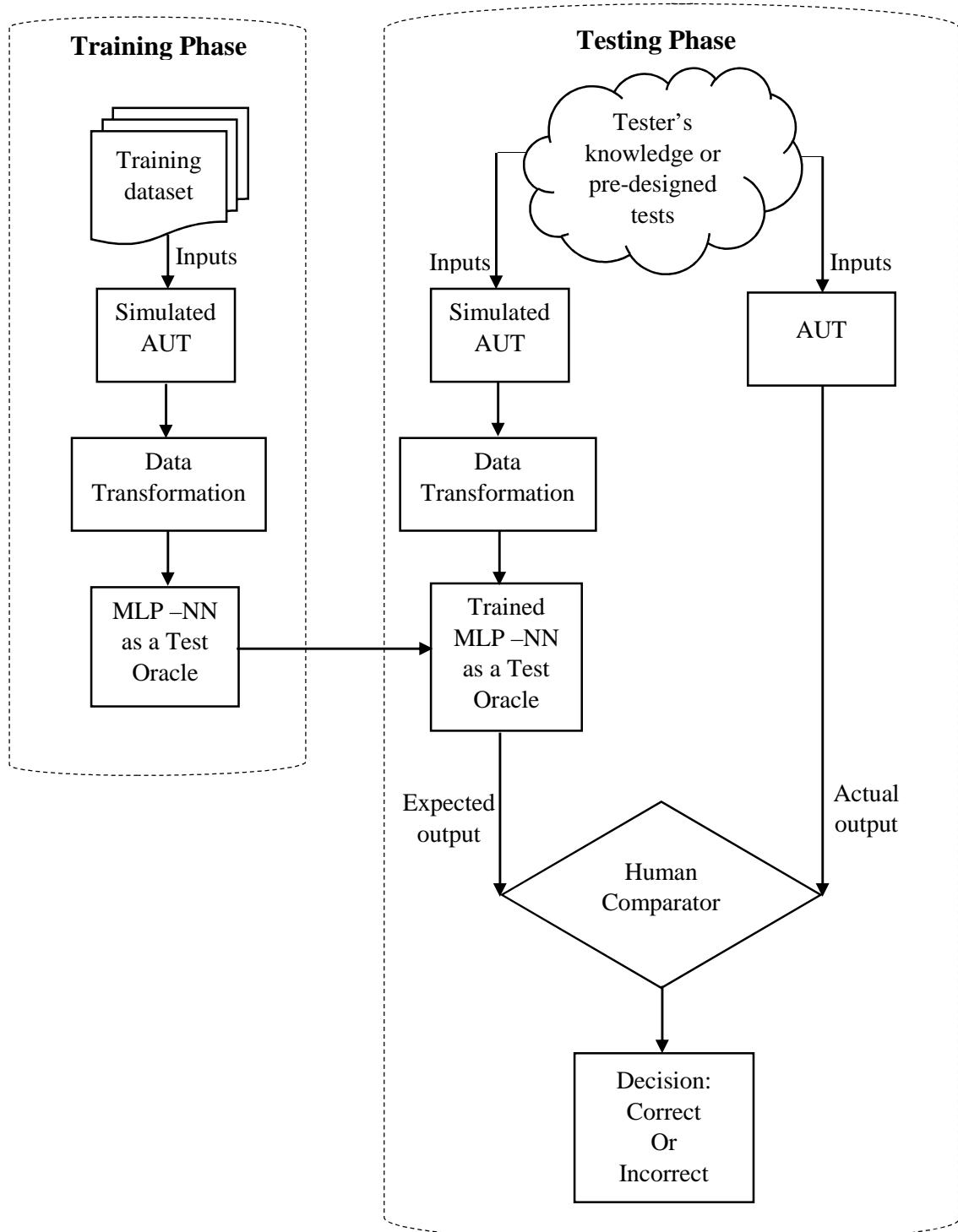
Our proposed methodology employs a multilayer perceptron neural network to facilitate learning of executing test cases and hence classify them as reliable test oracles. The multilayer perceptron neural network has to be trained first to be able predict the test oracle with minimum error using backpropagation algorithm. Ideally, a MLP-NN that performs best normally uses selected parameters from the experiments.

### **3.2 ARCHITECTURE OF PROPOSED EXPLORATORY TEST ORACLE**

A simulated model of the AUT that perform as the AUT is required to generate approximate expected output for any possible input(s) combination. Figure 1 shows the procedure of using the proposed exploratory test oracle.

Test inputs are first passed to the simulated model for compliance, and mapped to numeric data to be handled by MLP-NN, then training takes place. Testing follows using the trained MLP-NN to generate reliable exploratory test oracles. The same inputs are given to the AUT and actual outputs are produced. Based on the actual outputs and expected outputs, exploratory tester determines whether the test passed or not.

The performance of MLP-NN will be evaluated based on correct classification percentage, which is the difference between the expected simulated results and actual outputs. The well trained MLP-NN should provide a higher percentage of correct classification with minimum false passed tests (false positives) and minimum false failed tests (false negatives).

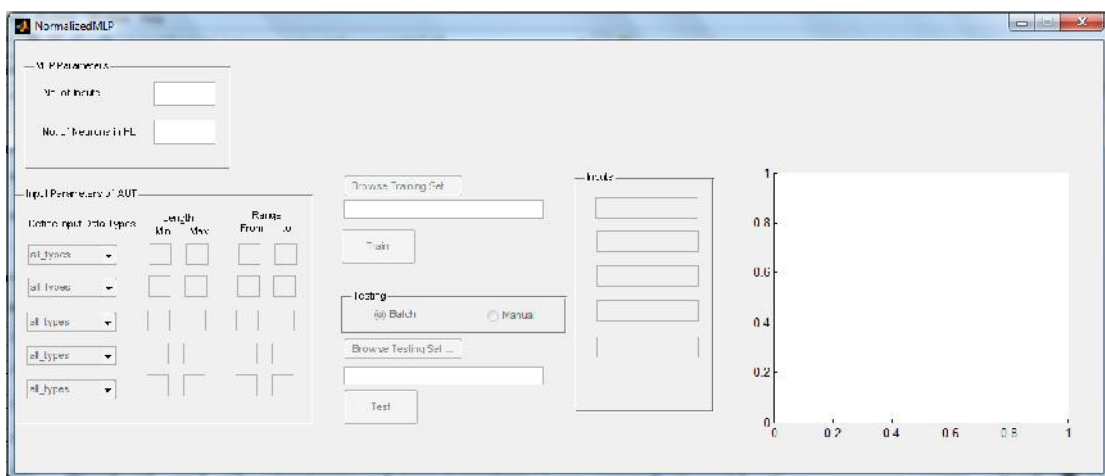


**Figure 3.1:** Proposed Exploratory Test Oracle Architecture

### 3.3 DESCRIPTION OF TOOL

We have developed exploratory test oracle based classifier using a multilayer perceptron neural network in MATLAB to predict the expected output of a given test case. The idea is to construct a MLP-NN that correspond to the input parameters of the test cases.

The tool accept a 2 dimensional array of test cases in CSV file format. The test case data comprises of different data types, i.e. string, alphanumeric, integer etc. The interface of the tool is shown in Figure 3.2.



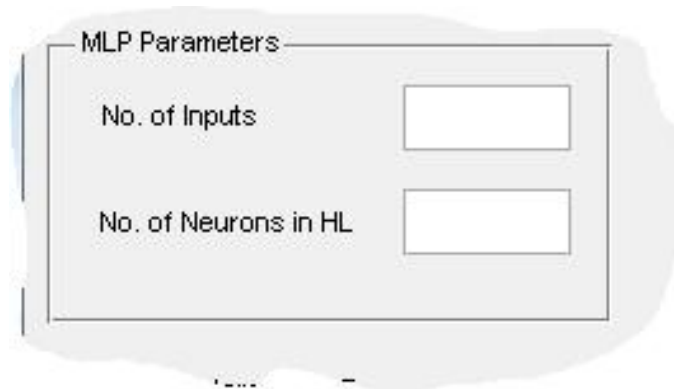
**Figure 3.2:** The Interface for Exploratory Test Oracle Prediction

Due to the size of our interface, everything may not be so clear. Every part of it has been shown separately in the subsequent sections.

#### 3.3.1 MULTILAYER PERCEPTRON NEURAL NETWORK PARAMETERS

The parameters of the MLP-NN are defined in Figure 3.3 to construct the network. The number of inputs defined should correspond to the number of inputs of the application under test. More than two hidden layers make the training process too complicated and make no improvements in the computing power. The tool has only one hidden layer, but however, determining the number of neurons in the hidden layer is not an easy task since there are no specific theorems or rules to comply with. The number of neurons in hidden layer has a substantial effect on the performance of the network; with too many

it will have training problems, also with too few neurons it might not learn the problem very well.



The image shows a dialog box titled "MLP Parameters". It contains two input fields: "No. of Inputs" and "No. of Neurons in HL". Both fields are currently empty.

**Figure 3.3:** MLP Parameters

### 3.3.2 INPUT PARAMETERS OF APPLICATION UNDER TEST

The input parameters of our tool are limited to 5. In Figure 3.4, the number of inputs of the AUT activated are equal to the number of inputs defined in the MLP-NN parameters in Figure 3.3. The AUT is simulated by defining the input data types, length (min and max) and range (min and max) for numerical inputs.



The image shows a dialog box titled "Input Parameters of AUT". It contains a table with five rows and four columns. The columns are labeled "Define Input Data Types", "Length", "Range", and "Range". The "Length" column has sub-columns "Min" and "Max". The "Range" column has sub-columns "From" and "to". Each row contains a dropdown menu with "all\_types" selected, and four empty input boxes for the "Length" and "Range" sub-columns.

Define Input Data Types	Length		Range	
	Min	Max	From	to
all_types ▼	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
all_types ▼	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
all_types ▼	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
all_types ▼	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
all_types ▼	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

**Figure 3.4:** Input Parameters of AUT



### 3.3.3 DATA LOADING AND TRANSFORMATION

In order to train the MLP-NN, we have designed training dataset in Table 3.3. When the MLP-NN parameters are defined, the Browse Training Set button in Figure 3.5 gets activated. Selecting or clicking that button allows for selection of the training dataset in CSV file format. When selected, the path and name of the file would be displayed in the text box shown in Figure 3.5.



**Figure 3.5:** Load Dataset and Train the MLP-NN

The Train button performs most of the work. Test cases cannot be directly fed into the MLP-NN for training or testing. The datasets may not be represented in an appropriate form for the MLP-NN. They may be in various formats such as strings, characters, alphanumeric or other forms.

Data transformation is needed for the format of MLP-NN since MLP-NNs can only learn from numerical values. For example, a string name “akashi”, can be mapped into numeric domain by converting to ASCII equivalence values as shown below:

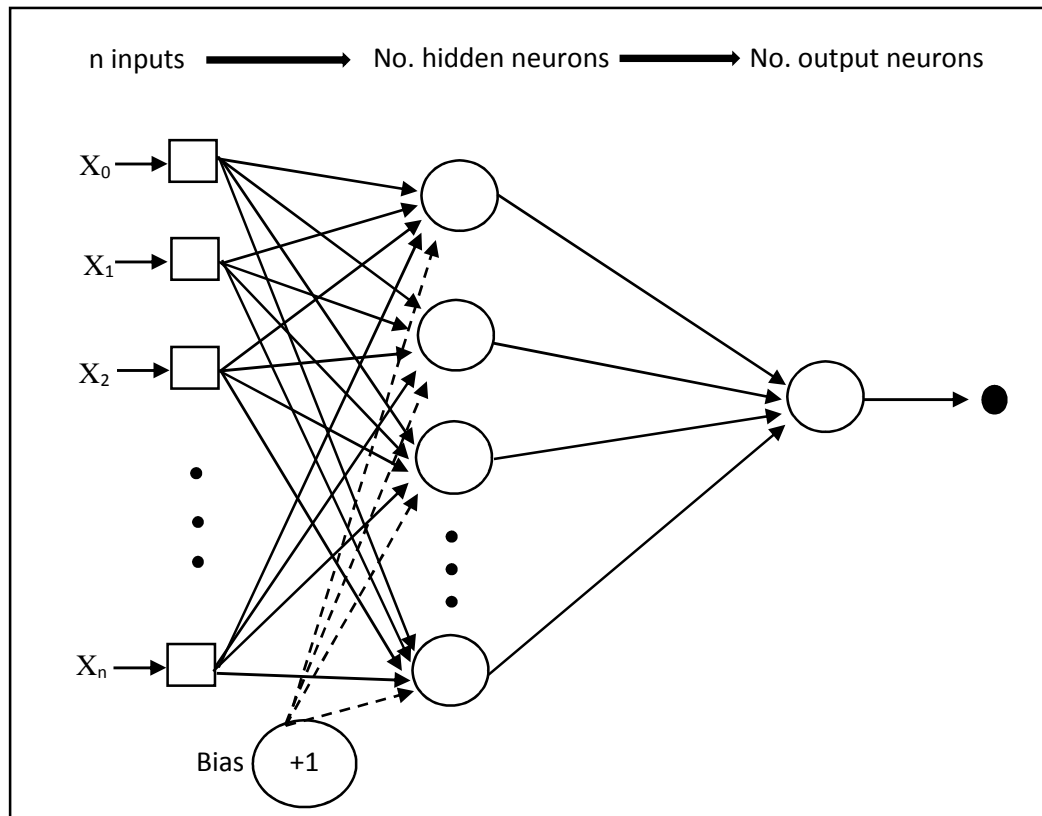
```
string = ASCII
akashi = 97  107  97  115  104  105
           or
a     k     a     s     h     i
97   107   97   115   104   105
```

Each character is converted to its ASCII equivalence, but the MLP-NN cannot handle all the values once as a single input. So, we summarized the ASCII codes to be taken as a single input.

```
sum_string_ascii =  ASCII
```

### 3.3.4 FEEDFORWARD MULTILAYER PERCEPTRON NEURAL NETWORK

A MLP-NN have layers of nodes with each node in a layer connected to each subsequent node. All nodes of hidden layer and output layer are associated with an activation function. The design of a feed forward network is such that signals are given straightforward through the whole network as in figure 3.6 below.

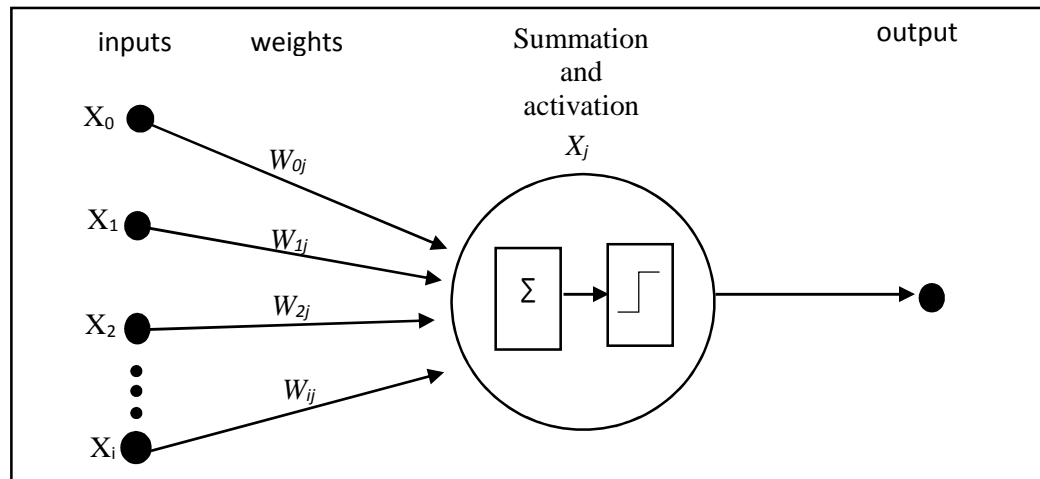


**Figure 3.6:** Multilayer Perceptron Neural Network

This network structure comprises of an input layer with  $n$  inputs, a single hidden layer with  $n$  neurons and the output layer with a single neuron. The input layer is represented by one or more input nodes in the multilayer architecture.

### 3.3.4.1 THE ARTIFICIAL NEURON

A neural network consists of a number of nodes, denoted by  $x_i$ . Every node accepts weighted inputs from all preceding nodes, processes and passes them to all subsequent nodes as shown in Figure 3.7. At every moment there is an activity denoted by  $x_i$  in each neuron. Neurons are connected by directed weight connections  $w$ , where the weight connection of node  $i$  to node  $j$  is denoted as  $w_{ij}$ .



**Figure 3.7:** An Artificial Neuron

The inputs are strengthened by the weights and summarized inside a neuron as shown in equation (3).

$$\bar{x}_{ij} = \sum_{i=1}^n x_i w_{ij} + w_0 \quad (1)$$

where  $w_0$  is a bias weight.

The value of the summarized weighted inputs is taken as an argument by the activation function. Equation (2) denotes the activating function with  $f$  and the result of applying it, which is  $x_j$ .

$$x_j = f(\bar{x}_j) \quad (2)$$

and the equation (3) shows a comprehensive formula with summarized activity.

$$x_j = f\left(\sum_{i=1}^n x_i w_{ij} + w_0\right) \quad (3)$$

### 3.3.4.2 ACTIVATION FUNCTIONS

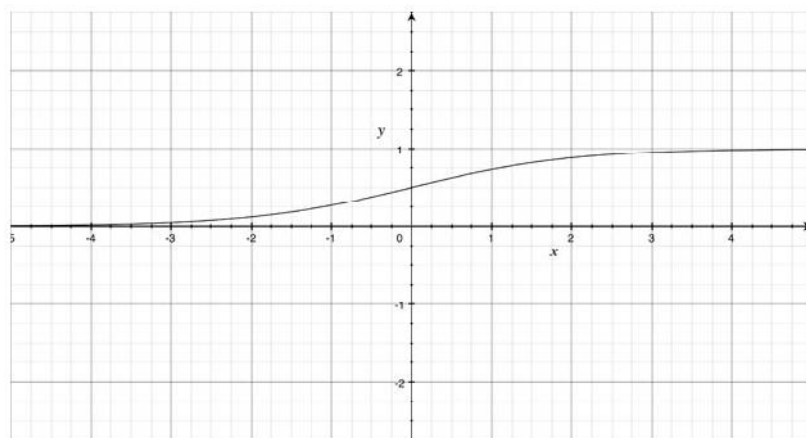
The manner in which neurons and network process signals are determined by the activation functions. When the neuron should be active is determined by the activation function depending on whether the threshold value is reached or not. We applied sigmoid function, hyperbolic tangent function and pure linear function. The choice of the activation function requires consideration of some important aspects. In the training process, experiments were carried out to evaluate the activation functions that address the problem better.

#### 3.3.4.2.1 SIGMOID ACTIVATION FUNCTION

A sigmoid activation function is defined as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (4)$$

Sigmoid means curled in twofold directions, like the alphabet character “S.” The sigmoid function is shown in Figure 3.8.



**Figure 3.8:** The Sigmoid Function

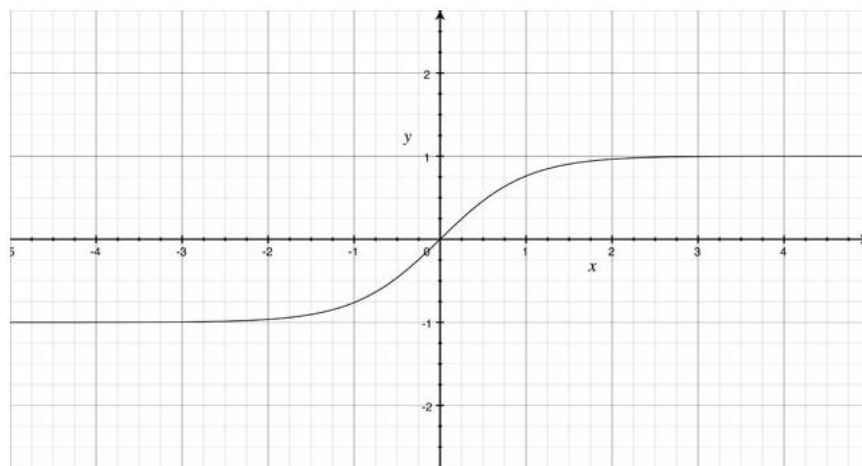
The positive values only are returned by the function, and it is often applied in recurrent and feedforward networks. If you need negative values to be returned by the neural network, the sigmoid function will be inappropriate.

## 3.3.4.2.2 HYPERBOLIC TANGENT ACTIVATION FUNCTION

Equation (5) defines the hyperbolic tangent function.

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (5)$$

It is a negative and positive attuned form of the sigmoid function and appears more difficult than the sigmoid activation function. It takes input of both positive and negative values. The hyperbolic tangent function is graphically provided in Figure 3.9.



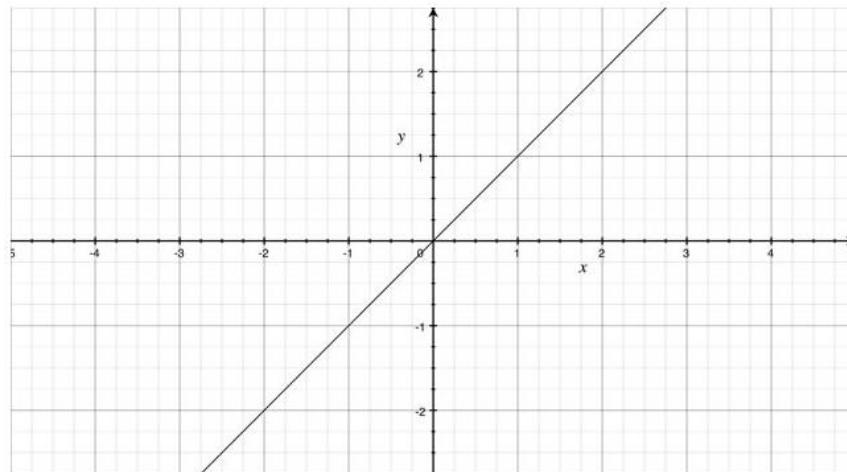
**Figure 3.9:** The Hyperbolic Tangent Function

## 3.3.4.2.3 PURE LINEAR ACTIVATION FUNCTION

A linear activation function is perhaps the least used activation function. We considered experimenting to check whether it gives good results or not. The Equation (6) defines the pure linear activation function.

$$f(x) = x \quad (6)$$

The linear activation function may be suitable in cases when the all range of numbers to be shown is needed. Generally, it looks like your neurons are active or non-active. It is graphically shown in Figure 3.10.



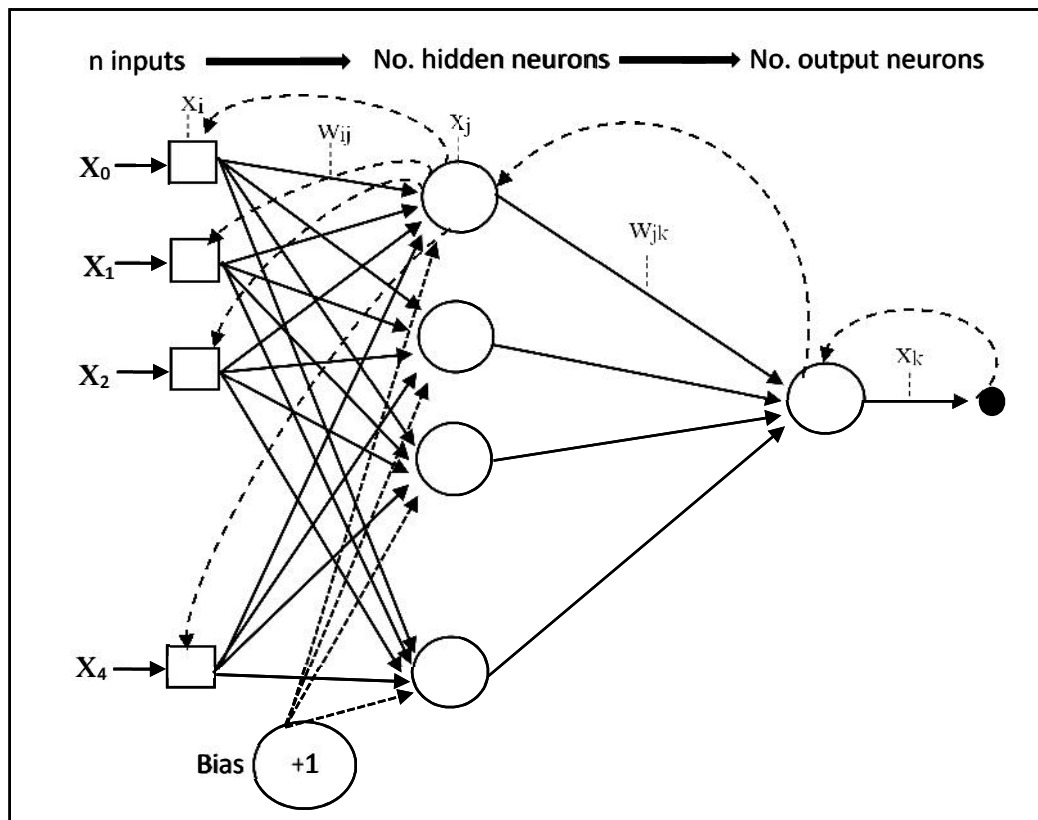
**Figure 3.10:** The Pure Linear Activation Function.

### 3.3.5 TRAINING PHASE

The ability to learn from its environment is the most crucial aspect of a neural network for it to be useful and improves its performing capabilities through learning. The learning procedure is accomplished by modifying the weights and threshold values. These weight modifications alters the manner in which the network deals with signals and the resulting output signal too. Even though diverse training algorithms are there, solving a difficult problem requires a proper selection of an algorithm. The training dataset relative to the environment that the network operates is needed to stimulate the network during the learning process.

#### 3.3.5.1 BACKPROPAGATION

The backpropagation learning algorithm was used in the multilayer perceptron neural network. It is a supervised training algorithm. The whole learning procedure includes computing the aggregate error that make up the output and adjust the weights in reverse order. The aggregate error in output is computed of all inputs and for all the output neurons. The overall error in output denotes the summation of the differences squared between the actual and the desired output.



**Figure 3.11:** The Backpropagation Algorithm in MLP-NN

The Figure 3.11 shows a two layered network comprising of four input units, four hidden neurons and one output neuron. Network nodes are labelled starting with  $x_i, x_j$  and  $x_k$ .

If the desired output for a given input vector  $x$  is  $d_k = (d_{k1}, d_{k2}, d_{k3})$  and the observed output is  $x_k = (x_{k1}, x_{k2}, x_{k3})$ . Then, the equation (7) for measuring the error is given below.

$$E_k = (d_{k1} - x_{k1})^2 + (d_{k2} - x_{k2})^2 + (d_{k3} - x_{k3})^2 \quad (7)$$

and in general, if we had  $m$  outputs, the measure of error would be:

$$E_k = \sum_{i=1}^m (d_{ki} - x_{ki})^2 \quad (8)$$

The sum of squared differences of the entire output nodes  $m$  are defined.

So, to obtain the overall differences for the whole dataset, we sum the output over all input vectors where the total input number is denoted by  $p$ .

$$E_k = \sum_{k=1}^p \sum_{i=1}^m (d_{ki} - x_{ki})^2 \quad (9)$$

$$E_p = \sum_p (d_k - x_k)^2 \quad (10)$$

The output obtained is clearly influenced by the inputs, weights and the error. Suppose we have  $n$  weights, the error will be denoted by:

$$E = f(w) = f(w_1, w_2, \dots, w_n) \quad (11)$$

As we consider the effect of a change of a single weight  $w$  to the error  $E$ , where  $f$  is a continuous and differentiable function similar to partial differentiation of  $E$  with respect to  $w_i$ ,  $\partial/\partial w_i$ . Therefore, for node  $i$  an error term  $\bar{\epsilon}_i$  is defined as

$$\bar{\epsilon}_i = \frac{\partial^+ E_p}{\partial \bar{x}_i} \quad (12)$$

Applying the chain rule,  $\bar{\epsilon}_i$  can be defined as

$$\text{node } \bar{\epsilon}_i = \begin{cases} -2(d_i - x_i) \frac{\partial x_i}{\partial \bar{x}_i} = -2(d_i - x_i)x_i(1 - x_i) & \text{if node } i \text{ is an output} \\ \frac{\partial x_i}{\partial \bar{x}_i} = \sum_{j,i < j} \frac{\partial^+ E_p}{\partial \bar{x}_j} \frac{\partial \bar{x}_j}{\partial x_i} = x_i(1 - x_i) \sum_{j,i < j} \bar{\epsilon}_j w_{ij} & \text{otherwise, (13)} \end{cases}$$

where the weight connection between node  $i$  to  $j$  is denoted by  $w_{ij}$ . The weight update equation  $w_{ki}$  is shown below.

$$\Delta w_{ki} = -\eta \bar{\epsilon}_i x_k \quad (14)$$

where  $\eta$  is the learning rate, which affect on the speed of convergence and weight stability during learning.



Many epochs are processed until the error attain a predefined threshold value, meaning that the error rate is low enough and all updated weights will be saved as shown below.

$$w_{new} = w_{old} + \Delta w \quad (15)$$

The absolute error is estimated as the average error rate as shown in equation 18.

$$E = \frac{1}{k} \sum_{i=1}^k E_i \quad (16)$$

The input combination or a test case would be given to the MLP-NN, forward pass is performed then backward propagation adjusting weights, if the error threshold is not reached, the same input combination is given again until the error threshold is reached, and then the next input combination is given. The process is repeated until an acceptable error is achieved and all training sets have been fed to the MLP. Updated weight matrix will be saved at the end of training.

Since elements in the input sets may have different data formats, therefore this requires a uniform transformation to allow equal impact of input variables to the MLP-NN. The principle of executing a test follows that all the inputs given to the software under test must be valid to be considered as successful. Otherwise, if any of the inputs is incorrect, the test would be considered as invalid or failed.

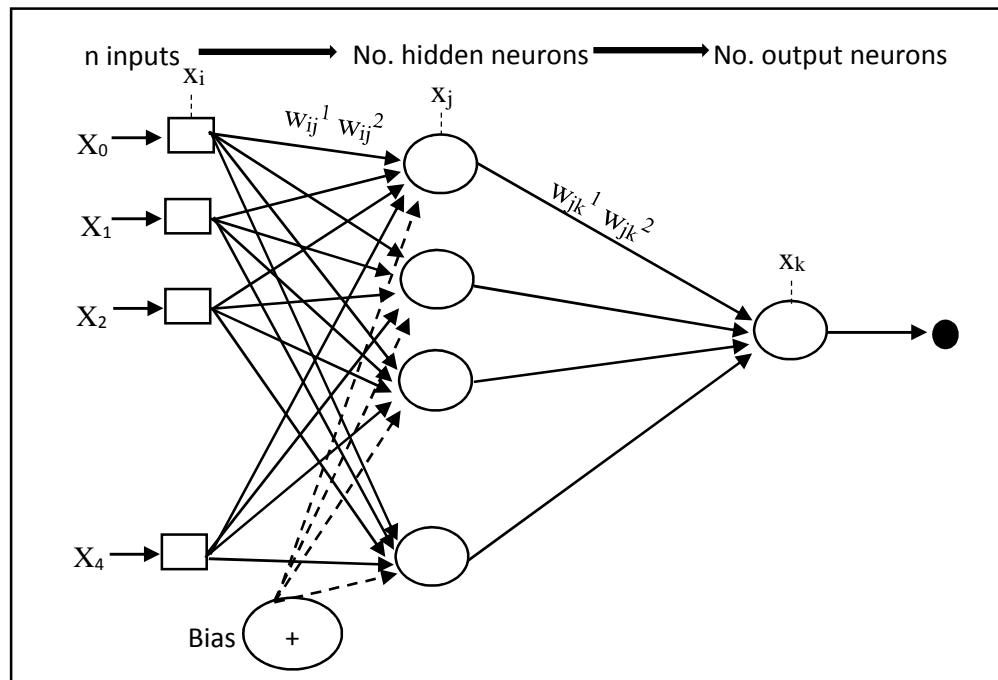
It is difficult to differentiate the ASCII values that are valid from invalid ones. For example, a field that takes strings only without special characters,

```

amit-kaushik = 1224
amitzkaushik = 1301
amit = 427

```

Therefore, the MLP-NN also fails to classify such data. We then considered the validation aspect of software under test, that is, the valid inputs must be associated with weights that differ from weights associated with the invalid inputs. This has given rise to the modification of MLP-NN in Figure 3.12.



**Figure 3.12:** Two Weighted MLP-NN

This allows the MLP-NN to be able to classify invalid and valid input combinations. A larger weight is selected when the input is valid in order to approximate towards the desired class 1 and smaller weight when the input is invalid. A combination of the valid inputs should be classified as correctly executed, class 1 and invalid inputs or the combination of the valid and invalid inputs should be classified as wrongly executed in class 0.

Making the choice of the structure of MLP-NN is a more challenging task. How many neurons in hidden layer are required for specific problem needs cautious analysis. To find the appropriate parameters, a number of experiments were carried out varying the parameters. The training dataset is crucial for supervised artificial neural network learning. Selection of training dataset is the other difficult task that requires careful consideration. After the MLP-NN is well trained, it becomes possible to predict the expected output for new input data sets.

### 3.3.6 TESTING PHASE

After proper training of the MLP-NN, it becomes feasible to be used as an exploratory test oracle. The procedure of applying the MLP-NN based test oracle to generate expected outputs follows: get the test cases as testing data, execute them on AUT

producing the actual output; meanwhile, the same inputs are given to the trained MLP to generate the approximate expected output. Human intelligence is employed as a comparator to make a comparison of the outputs. The deviation between the outputs is considered as a possible AUT defect. Testing can be batch testing or manual testing.

### 3.3.6.1 BATCH TESTING

In batch testing, the entire testing set is presented first in CSV file format, then actual testing begins. The testing set should be selected in Figure 3.13.



**Figure 3.13:** Batch Testing

### 3.3.6.2 MANUAL TESTING

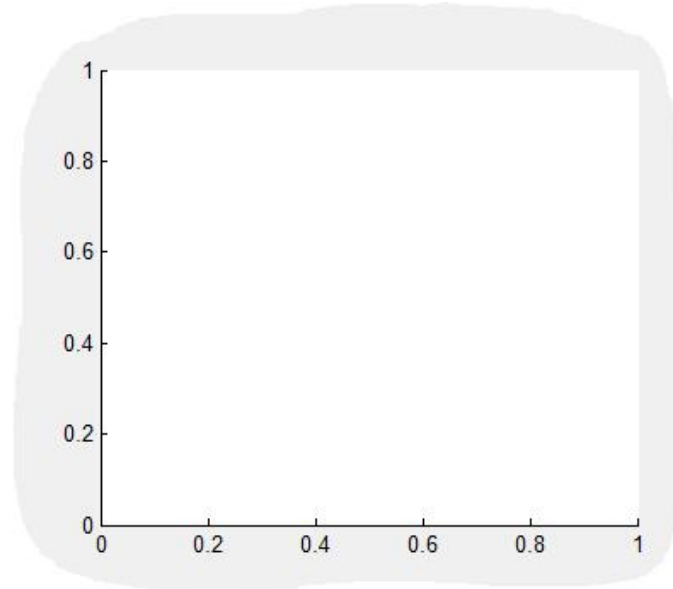
When testing is manual, testing becomes exploratory where test cases are designed at the same time executed one by one. Figure 3.14 shows part of the interface where the exploratory tester performs tests on the MLP-NN.



**Figure 3.14:** Exploratory Testing

### 3.3.7 OUTPUT DISPLAY

The outputs of the batch and manual testing will be graphically represented in Figure 3.15. The expected passed tests will be plotted above 0.5 and failed tests below 0.5.



**Figure 3.15:** The MLP-NN Testing Outputs

In this thesis, the same dataset was used to perform the evaluation of MLP-NN classification with 3 different activation functions.

**Table 3.1:** Combination of Activation Functions

Code No.	Network Code	Hidden Layer	Output Layer
1	sig-sig	Sigmoid	Sigmoid
2	sig-pul	Sigmoid	Pure Linear
3	sig-tanh	Sigmoid	Hyperbolic Tangent
4	pul-pul	Pure Linear	Pure Linear
5	Pul-sig	Pure Linear	Sigmoid
6	pul-pul	Pure Linear	Hyperbolic Tangent
7	tanh-tanh	Hyperbolic Tangent	Hyperbolic Tangent
8	tanh-pul	Hyperbolic Tangent	Pure Linear
9	tanh-sig	Hyperbolic Tangent	Sigmoid

The test case data have two output classes, namely, class 0 that denotes failed tests and class 1 that denotes passed tests. The network was trained and tested varying the MLP-

NN parameters and the following are the parameters that were evaluated: Activation functions, Number of neurons in Hidden Layer, Error threshold, E and Learning rate,  $\eta$ . The parameters that give the best performance were chosen.

### 3.4 SUMMARY OF EXPLORATORY TEST ORACLE

Exploratory test oracle generates the approximate expected outputs. The following steps summarized exploratory test oracle:

**Step 1:** Define parameters of the neural network.

**Step 2:** Simulate AUT.

**Step 3:** Load test cases for training.

**Step 4:** Perform data transformation i.e. map non-numeric data into numeric.

**Step 5:** Randomly initialize the weights.

**Step 6:** Perform feedforward pass in the MLP.

**Step 7:** The neural network trains using back propagation algorithm.

**Step 8:** In the case that the stopping criterion is unsatisfied, step 6 and 7 are repeated till all inputs of the AUT are fed to the MLP, otherwise the weights are kept and proceed to step 8.

**Step 9:** Obtain the testing inputs of the AUT and give to the neural network. Get the estimated output and determine if there is a failure or not.

**Step 10:** If a different module need to be tested, go back to step 1 or else, the process finishes.

### 3.5 PERFORMANCE EVALUATION

Evaluation of MLP-NN performance will be based on correct classification percentage, which is the difference between the expected simulated results and actual outputs. Equation 16 defines the correct classification.

$$\% \text{ Correct Classification} = \frac{\text{Total of Correct Data}}{\text{Total Number of Data}} \times 100\% \quad (17)$$

### 3.6 TEST CASES

A dataset with 160 test cases was created to be used in the multilayer perceptron neural network based on the AUT specifications in Table 3.2. It comprises of invalid test cases that fall in class 0 and valid ones that fall in class 1. Each data set has four features and its corresponding class. The features are Username, Password, Employee id and security access level.

**Table 3.2:** Specifications of the AUT

	Data Type	Length		Range	
		Min	Max	Min	Max
<b>Username</b>	String	8	12		
<b>Password</b>	All types	8	15		
<b>Employee ID</b>	Integer	5	5	98000	99999
<b>Access Level</b>	Integer	1	1	1	5

The data was divided into two, 60 for training and 100 for testing.

#### 3.6.1 TEST CASES FOR TRAINING

The detailed data for training is described in the Table 3.3 below. This dataset contains 50% valid test cases and 50% invalid test cases.

**Table 3.3:** Test Cases for Training.

Username	Password	Emp_ID	Access Level	Class
				0
jones				0
3jones				0
jones-clerk				0
234				0
jonhsonsbright				0
jonhson				0
jonhsons				0
jonhsons	vj			0
jonhsons	>34			0
jonhsons	1232125			0

jonhsons	leejones12345678			0
jonhsons	jonhsons1234			0
jonhsons	jonhsons1234	124RFD		0
jonhsons	jonhsons1234	234@FHG		0
jonhsons	jonhsons1234	34E.454		0
jonhsons	jonhsons1234	-333		0
joneslee	jonhsons1234	98121	0	0
joneslee	jonhsons1234	98125	100	0
joneslee	jonhsons1234	98128	4.5	0
haltordy	dvd343fdcc	100000	4	0
siltordy	suv343fdcc	98403	two	0
warordy	nbvb343fdcc	98403	3+3	0
hawardy	hhf343fdcc	98403	3*3	0
hardman	sda343fdcc	98403	2&3	0
rexmanns	snbv343fdcc	34359+32	2	0
wiltordy	mnb43fdcc	34360?	2	0
wiltordy	fgh343fdcc	34361sa	3	0
wiltordy	sxcv343fdcc	343 62	2	0
wiltordy	zxsd343fdcc	34*363	3	0
joneslee	jonhsons1234	98131	3	1
marylean	mary1234	98144	1	1
jameslean	james1234	99123	2	1
kelvinlean	kelvin1234	98576	5	1
marialean	maria1234	98432	4	1
amitlean	amit1234	98345	2	1
mayanklean	mayank1234	99231	3	1
ankitlean	ankit1234	99323	2	1
piyushlean	piyush1234	99878	2	1
hiteshlean	hitesh1234	99434	1	1
rajeshlean	rajesh1234	99545	4	1
ankullean	ankul1234	99090	5	1
maneeshlean	maneesh1234	99121	4	1
punitlean	punit1234	99586	3	1
brijeshlean	brijesh1234	99999	2	1
tejnalean	tejna1234	99002	3	1
deeptilean	deepti1234	99004	2	1
kritilean	kriti1234	98999	1	1
rechnalean	rechna1234	98095	4	1
ivyleans	ivy12345	98765	4	1
savenance	fhg34jfff6	99023	5	1
samarahm	were367fkf	99878	1	1
alvisdani	ty5639ujft	99999	4	1
albajodi	t58jif3xeqw	99854	2	1
patrickas	3udj4jd43ea	98576	2	1

phiniasy	4r5fi5id4iw	98403	1	1
douglasto	dou45glas	98543	5	1
selinasky	sli875sky	99586	3	1
malvinsir	sir1malvin	99248	4	1
hulkrules	champion1	99385	3	1

### 3.6.2 TEST CASES FOR TESTING

The detailed data for testing is described in the Table 3.4 below. This dataset contains 50% valid test cases and 50% invalid test cases.

**Table 3.4:** Testing Datasets

Username	Password	Emp_ID	Access Level
jonhsons	jonhsons1234	342+87	
jonhsons	jonhsons1234	456;	
jonhsons	jonhsons1234	ten	
jonhsons	jonhsons1234	4532"SDS"	
jonhsons	jonhsons1234	78>2	
jonhsons	jonhsons1234	345 453	
joneslee	jonhsons1234	1	
joneslee	jonhsons1234	23	
joneslee	jonhsons1234	234	
joneslee	jonhsons1234	1234	
joneslee	jonhsons1234	123456	
joneslee	jonhsons1234	98121	
joneslee	jonhsons1234	98121	-1
joneslee	jonhsons1234	98122	6
jonhsons	vj		
jonhsons	>34		
jonhsons	1232125		
jonhsons	leejones12345678		
jonhsons	jonhsons1234	Gtr453	
jonhsons	jonhsons1234	./iy	
jonhsons	jonhsons1234	124RFD	
jonhsons	jonhsons1234	234@FHG	
jonhsons	jonhsons1234	34E.454	
jonhsons	jonhsons1234	-333	
joneslee	jonhsons1234	98127	50
joneslee	jonhsons1234	98129	2,5
joneslee	jonhsons1234	98130	"1"
welshl	welsh1234	99231	1
2welshl	welsh1235	99231	1



wel@shl	welsh1236	99231	1
welshlee	welsh1237	99231	1
welshl	welsh 1238	99231	1
welshl	welsh1239?	99231	1
joneslee	jonhsons1234	98125	100
joneslee	jonhsons1234	98128	4.5
wiltordy	suv343fdcc	100000	4
wiltordy	suv343fdcc	98403	two
wiltordy	suv343fdcc	98403	3+3
welshl	welsh-1240	99231	1
welshl	welsh1241	992310	1
welshl	welsh1242	9923	1
welshl	welsh1243	seven	1
welshl	welsh1244	99.231	12
wiltordy	suv343fdcc	34359+32	2
wiltordy	suv343fdcc	34360?	2
wiltordy	suv343fdcc	34361sa	3
wiltordy	suv343fdcc	343 62	2
wiltordy	suv343fdcc	34*363	3
welshl	welsh1245	99.231	8
welshl	welsh1246	99.231	`1
sandeeplean	sandeep1234	98343	2
prashitlean	prashit1234	99827	1
sanchitlean	sanchit1234	98943	4
sakshilean	sakshi1234	99878	4
shivilean	shivi1234	99434	2
abmanyulean	abmanyu1234	99545	3
rajendralean	rajendra1234	99090	2
sachinlean	sachin1234	99121	4
joellean	joel1234	99586	3
phiniasy	4r5fi5id4iw	98403	1
douglasto	dou45glas	98543	5
selinasky	sli875sky	99586	3
malvinsir	sir1malvin	99248	4
hulkrules	champion1	99385	3
jonesonlean	joneson1234	99999	2
douglaslean	douglas1234	99002	3
easterlean	easter1234	99878	2
theresalean	theresa1234	99434	4
willardlean	willard1234	99434	4
ivyleans	ivy12345	98765	4
savenance	fhg34jfjf6	99023	5
samarahm	were367fkf	99878	1
alvisdani	ty5639ujft	99999	4

albajodi	t58ijf3xeqw	99854	2
elizabethlean	elizabeth123	99434	5
irenelean	irene1234	99121	5
beautylean	beauty1234	99586	5
denfordlean	denford1234	99999	4
marialean	maria1234	98432	4
amitlean	amit1234	98345	2
mayanklean	mayank1234	99231	3
ankitlean	ankit1234	99323	2
piyushlean	piyush1234	99878	2
stewardlean	steward1234	99002	5
riannlean	riann1234	99878	4
blancolean	blanco1234	99434	4
danuellean	danuel1234	99545	3
kurtlean	kurt1234	99090	3
marylean	mary1234	98144	1
jameslean	james1234	99123	2
kelvinlean	kelvin1234	98576	5
marialean	maria1234	98432	4
amitlean	amit1234	98345	2
micheallean	micheal1234	99121	4
wadelean	wade1234	99586	2
keenenlean	keenen1234	99999	1
alvislean	alvis1234	99002	1
messilean	messi1234	99545	2
albalean	alba1234	99090	1
alvinlean	alvin1234	99121	2

Table 3.5 describes the data division summary.

**Table 3.5:** Summary of testing and training datasets

	Training Data	Testing Data
Valid Test Cases	30	50
Invalid Test Cases	30	50
Total Test Cases	60	100

# **CHAPTER 4**

## **EXPERIMENTAL RESULTS**

## 4.1 INTRODUCTION

The aim of this thesis was to predict the exploratory test oracle using MLP-NN. In order to choose the optimal parameters that give good results, experiments were done differing parameter values and evaluate the results. The excellent networks give the best percentage of correct classification and outputs that are close to 0 or 1, if the test failed or passed respectively. Accuracy of the MLP-NN was evaluated for selection of the parameters suitable to correctly classify the test oracles.

## 4.2 MULTILAYER PERCEPTRON NEURAL NETWORK

Figure 4.1 shows the output of a MLP-NN with a single weight on each connection. The first 50% of the test cases were invalid and the remaining valid. The output shows the expected outputs to be in the same class which means 50% are falsely failed tests (false negatives) and the correct classification was 50%.

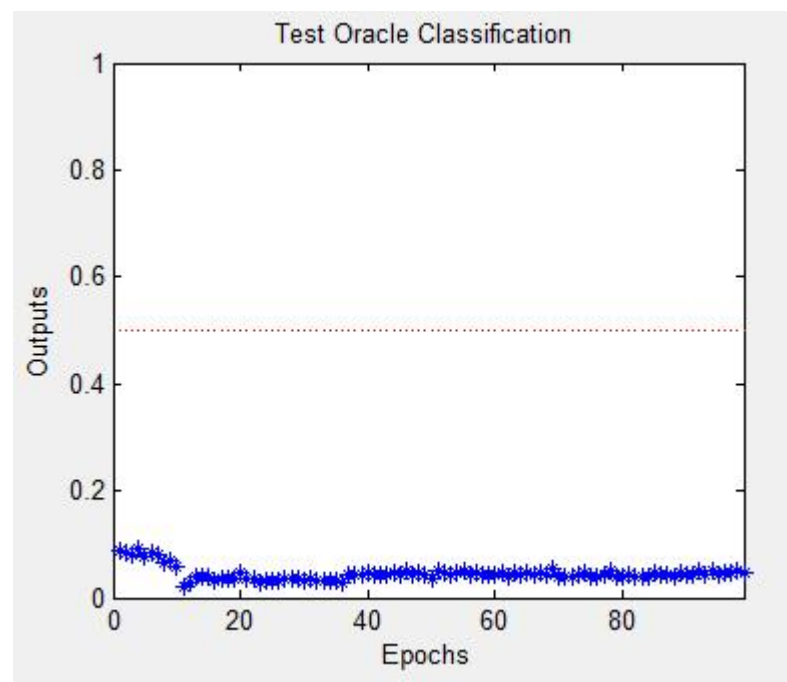


Figure 4.1: Output of a Common MLP-NN

### 4.3 MODIFIED MULTILAYER PERCEPTRON NEURAL NETWORK

The improved MLP-NN with two weights on each connection perfectly generated approximate exploratory test oracles. 99% correct classification and 1% false expected passed tests or false positives was achieved as shown in Figure 4.2.

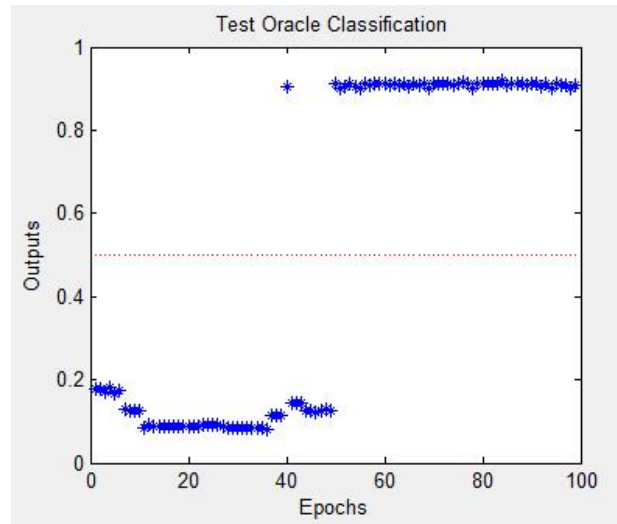


Figure 4.2: Output of a Modified MLP-NN

#### 4.3.1 CHANGING ACTIVATION FUNCTIONS

Activation functions were being changed in the hidden and output layer. A dataset with 100 test cases (samples) was used during the testing phase. Figure 4.3 shows a Sigmoid Activation function which was applied in both layers, and it correctly classified the test oracles, but some of the points were a little bit far from 0, in the case of failed tests.

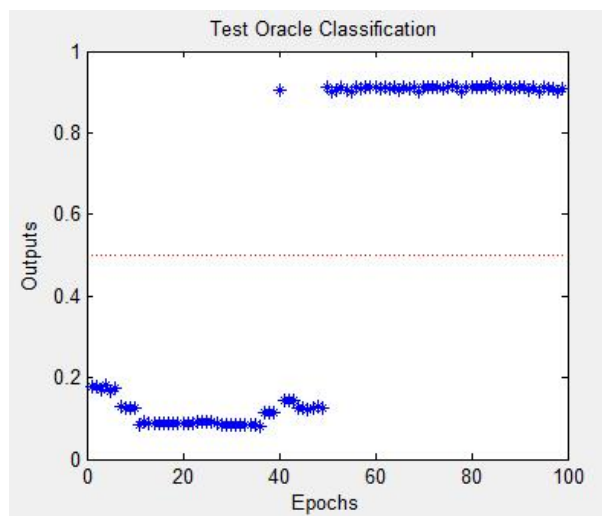
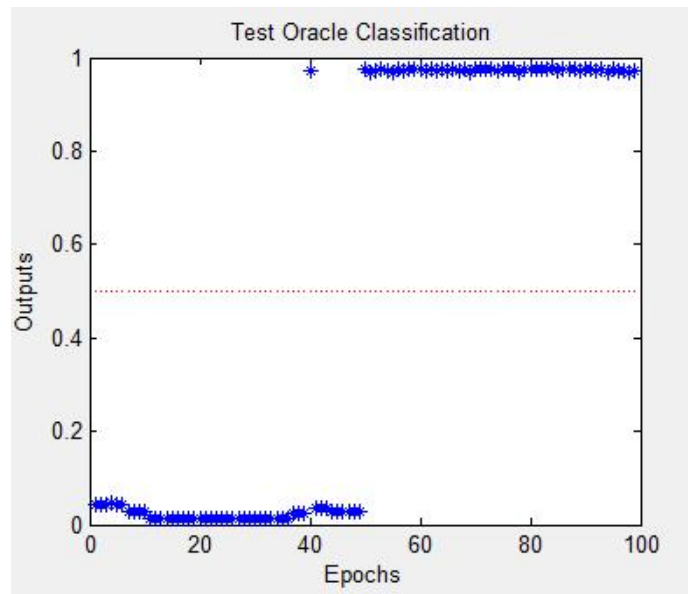


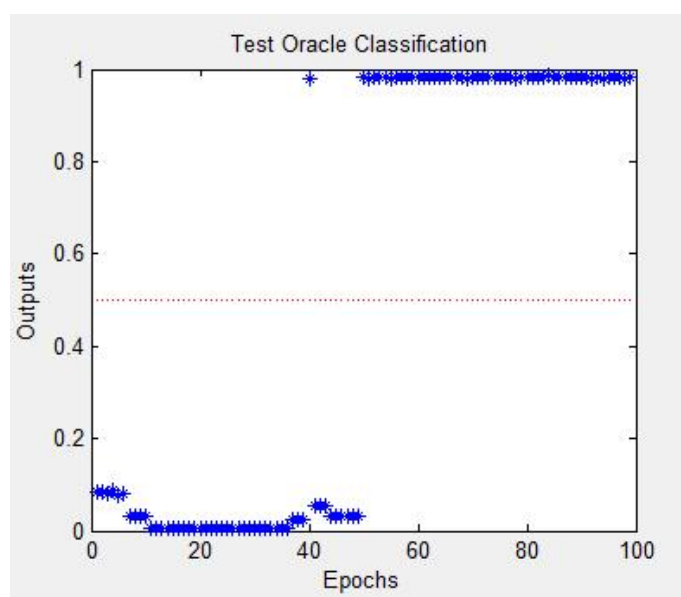
Figure 4.3: Sigmoid Activation function in both Output Layers

In Figure 4.4, the output of the sigmoid activation function in hidden layer and hyperbolic tangent in the output Layer is shown. The test oracles were correctly classified, where outputs in class 1 were too close to 1 and in class 0 were too close to 0 as well.



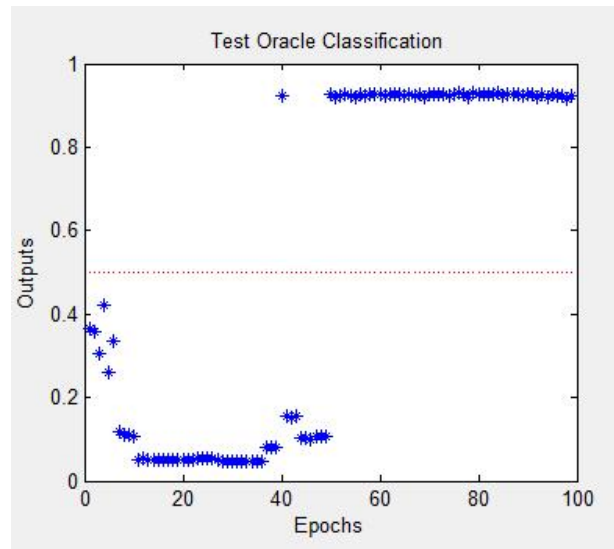
**Figure 4.4:** Sigmoid in the Hidden and Hyperbolic Tangent in the Output Layer

Figure 4.5 shows the output of the hyperbolic tangent activation function in both layers (hidden and Output). The test oracles were correctly classified, where outputs in class 1 were too close to 1 and in class 0 some were a bit far from 0.



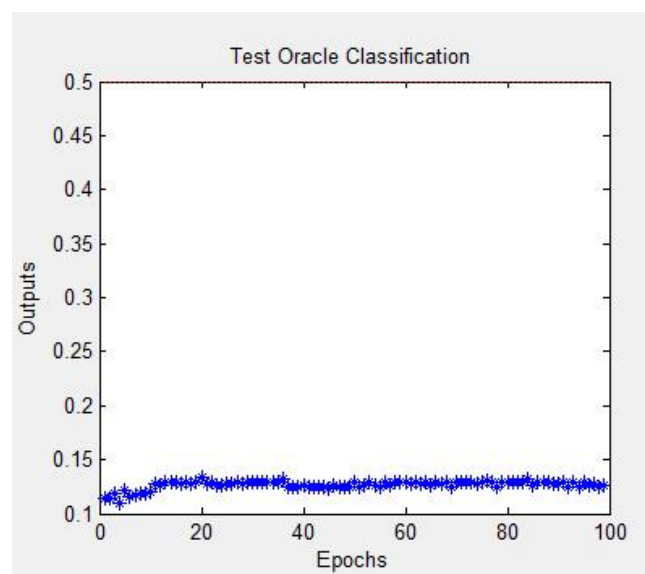
**Figure 4.5:** Hyperbolic Tangent in both layers - Hidden and Output

Figure 4.6 shows the output of the hyperbolic tangent activation function in the hidden layer and the sigmoid activation function in the output layer. The test oracles were correctly classified, where outputs in class 1 were not so close to 1 and in class 0 as well, some were even close to the boundary 0.5.



**Figure 4.6:** Hyperbolic Tangent in the Hidden Layer and Sigmoid in the Output Layer

In Figure 4.7, the output of the combination of the hyperbolic tangent function and the pure linear function classified all test oracles into class 0. All outputs were in the range between 0.1 and 0.15. It shows that the pure linear function cannot model the brain of a human being. In some cases where pure linear was in the hidden layer, the MLP executed infinitely i.e. did not reach the stopping criteria.



**Figure 4.7:** Hyperbolic Tangent in the Hidden Layer and Pure Linear in the Output Layer

Table 4.1 shows a summary of the outputs of combinations of activation functions in the hidden layer and output layer.

**Table 4.1:** Evaluation of Activation Functions

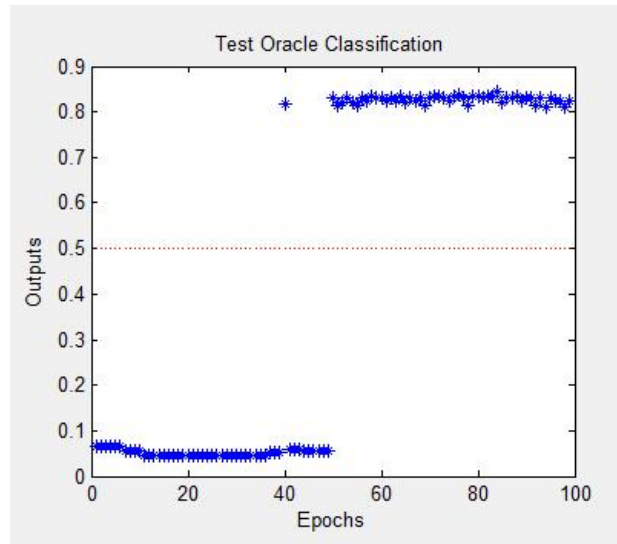
Number Code	Network Code	Class 0		Class 1	
		Min	Max	Min	Max
1	sig-sig	0.091078	0.17542	0.85837	0.87864
2	sig-tanh	0.014543	0.04371	0.91718	0.93957
3	sig-pul	Infinite training			
4	pul-pul	Infinite training			
5	Pul-sig	Infinite training			
6	pul-pul	Infinite training			
7	tanh-tanh	0.0076123	0.085012	0.92917	0.9534
8	tanh-pul	0.067344	0.093526		
9	tanh-sig	0.073464	0.41424	0.86547	0.89706

The results have shown that using sigmoidal activation function in the hidden layer and hyperbolic tangent activation function in the output layer yields the best results. The pure linear activation function is the waste of all in our situation as it resulted in infinite training execution. It executed finitely when the network code was tanh-pul, but classified all test oracles in failed class. Hence it is not useful for training a neural network. Therefore, in the subsequent experiments the sigmoid activation function was applied in the hidden layer and the hyperbolic tangent activation function in the output layer.



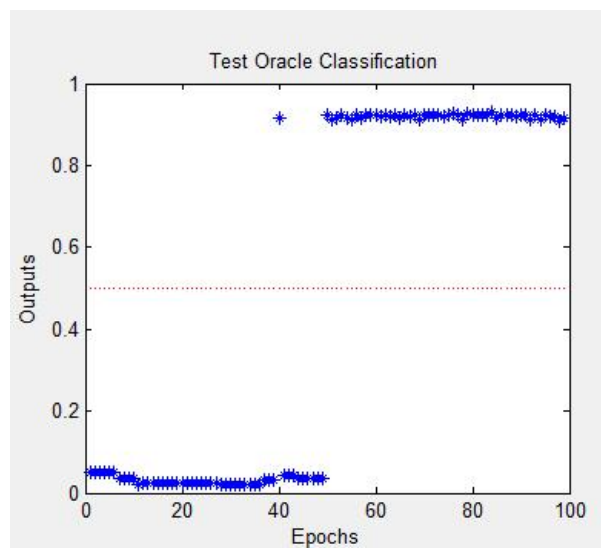
### 4.3.2 CHANGING THE NUMBER OF NEURONS IN THE HIDDEN LAYER

In this section we investigate the effects of the number of neurons in hidden layer to the outputs in the MLP. The aim is to identify the number of neurons in hidden layer that gives the best outputs. Figure 4.8 shows outputs when there is 1 neuron in the hidden layer. Summarized data values are provided in Table 4.2.



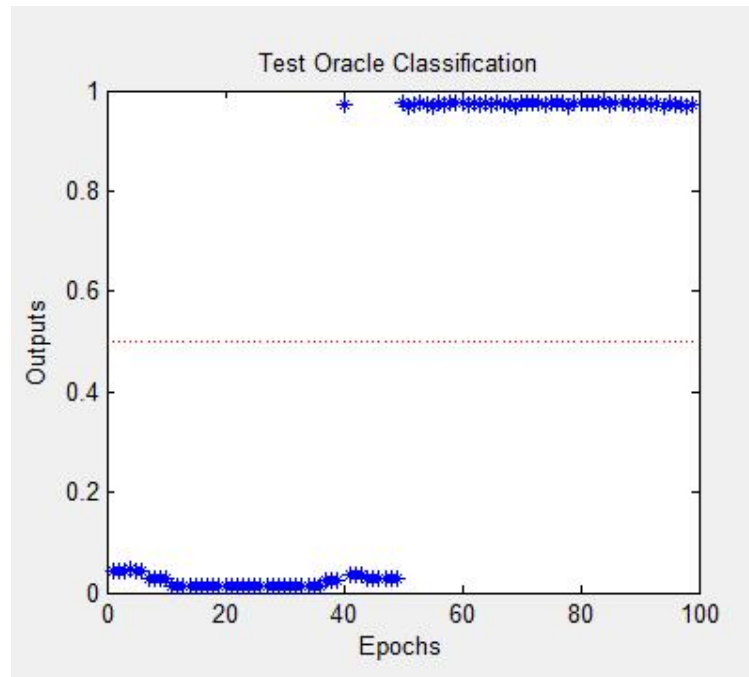
**Figure 4.8:** 1 Neuron in the Hidden Layer

In Figure 4.9, outputs are shown when 2 neurons in the hidden layer were used in a MLP network. Summarized data values are provided in Table 4.2.



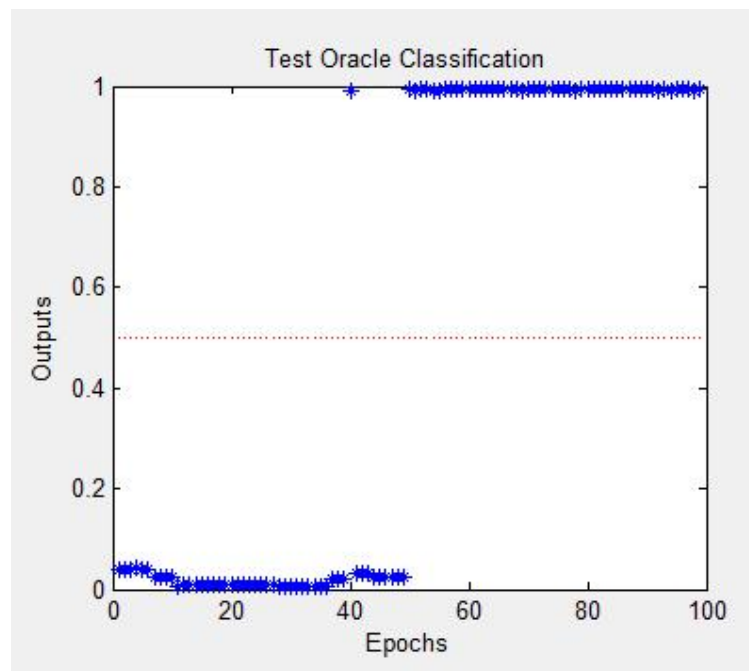
**Figure 4.9:** 2 Neurons in the Hidden Layer

Outputs are shown when 3 neurons in the hidden layer were used in a MLP network in Figure 4.10. A summary of its data values is provided in Table 4.2.



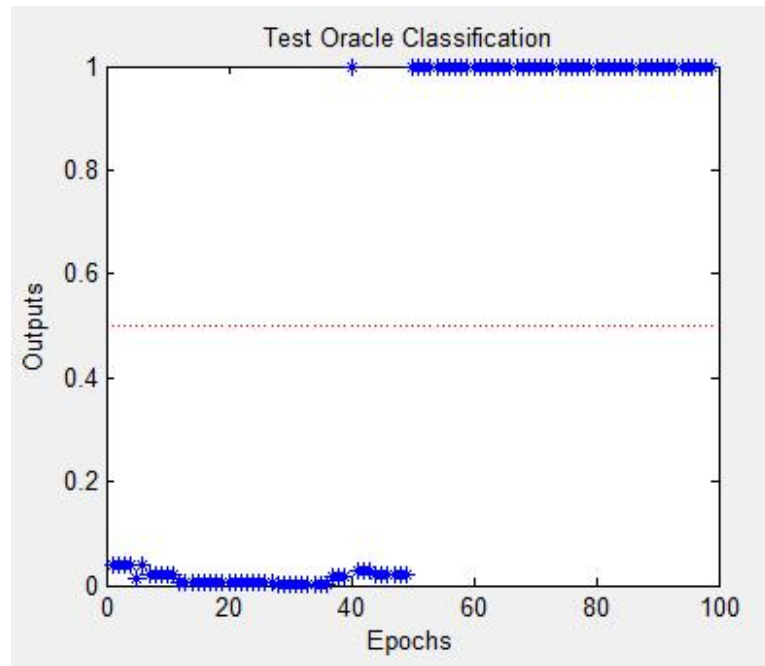
**Figure 4.10:** 3 Neurons in the Hidden Layer

Figure 4.11 shows outputs when 4 neurons in the hidden layer were used in a MLP neural network. Summarized data values are provided in Table 4.2.



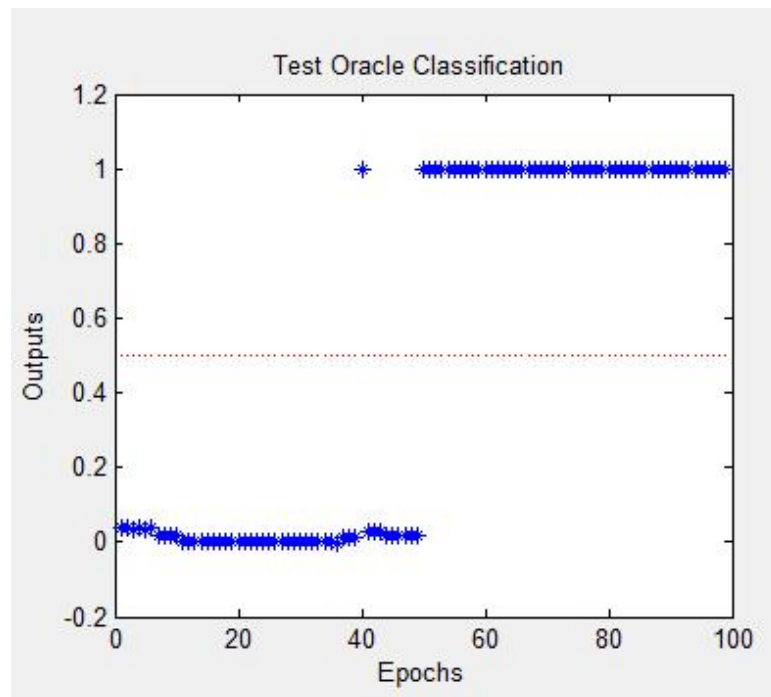
**Figure 4.11:** 4 Neurons in the Hidden Layer

In Figure 4.12, outputs are shown when 5 neurons in the hidden layer were used in a MLP network. Summarized data values are provided in Table 4.2.



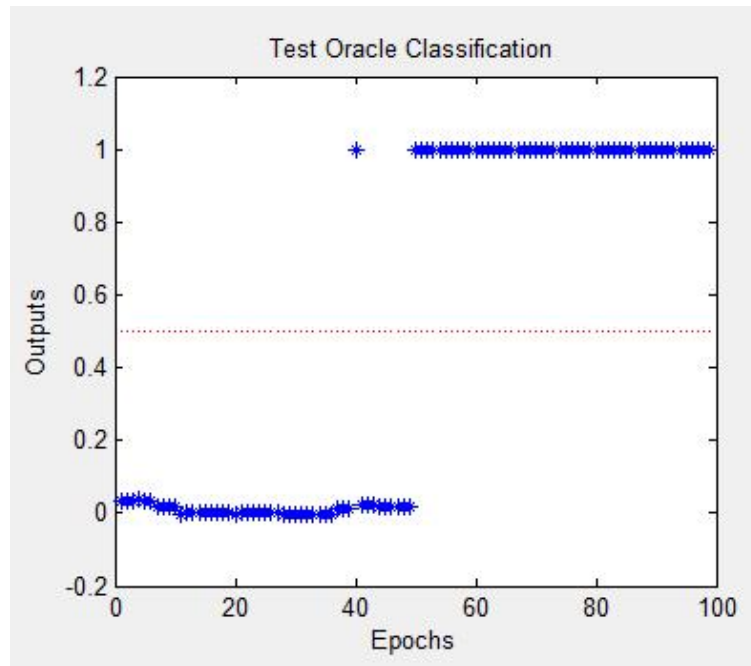
**Figure 4.12:** 5 Neurons in the Hidden Layer

Outputs are shown when 6 neurons in the hidden layer were used in a MLP network in Figure 4.13. A summary of its data values is provided in Table 4.2.



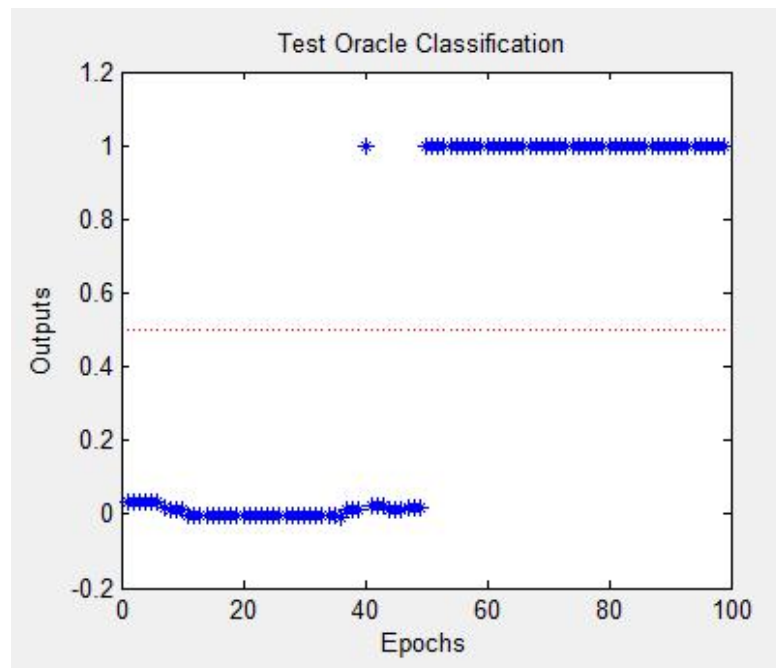
**Figure 4.13:** 6 Neurons in the Hidden Layer

Figure 4.14 shows outputs when 7 neurons in the hidden layer were used in a MLP neural network. Summarized data values are provided in Table 4.2.



**Figure 4.14:** 7 Neurons in the Hidden Layer

In Figure 4.15, outputs are shown when 8 neurons in the hidden layer were used in a MLP network. Summarized data values are provided in Table 4.2.



**Figure 4.15:** 8 Neurons in the Hidden Layer

**Table 4.2:** Summary of MLP Outputs for Various Neurons in the Hidden Layer

No. of Neurons in hidden layer	Class 0		Class 1	
	Min	Max	Min	Max
1	0.48082	0.066778	0.77831	0.81652
2	0.024467	0.050567	0.86396	0.8944
3	0.014543	0.04371	0.91718	0.93957
4	0.0092992	0.040164	0.9585	0.97214
5	0.0055741	0.037807	0.98195	0.989
6	0.0033354	0.036227	0.99252	0.99588
7	0.0013707	0.034938	0.99694	0.99847
8	-0.00010363	0.033971	0.99875	0.99944

Changing the number of neurons in the hidden layer showed that few neurons give desirable outputs, but the network learns very slowly. Many neurons results in outputs with large error, but learn much faster. We therefore choose to use 4 neurons in the hidden layer. Since we are predicting the test oracle, we are expecting an output from a single pass as passed or failed; that is why we have one output neuron in the output layer.

### 4.3.3 VARYING THE NUMBER OF ITERATIONS

An investigation of the stability of the MLP was shown in this section. The objective is to evaluate the MLP, if it continues to give correct results when dealing with datasets of different sizes. Figure 4.16 shows the outputs of a dataset with 100 test samples.

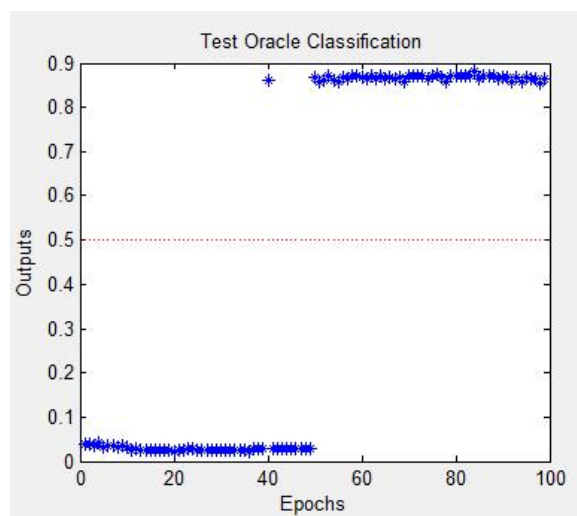
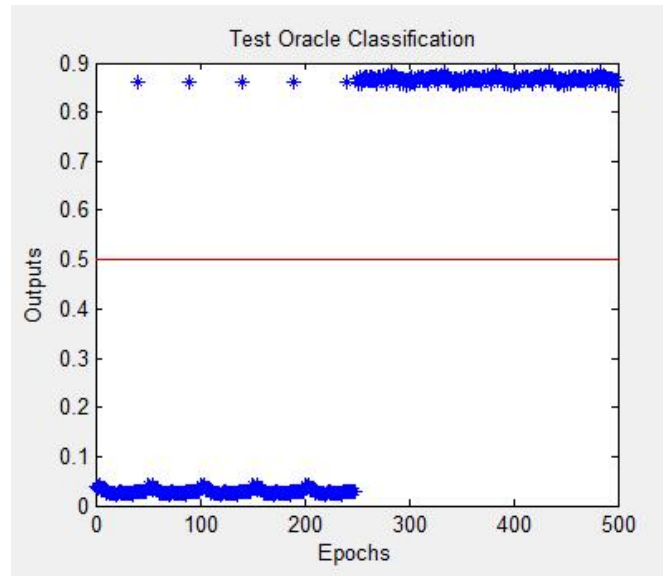
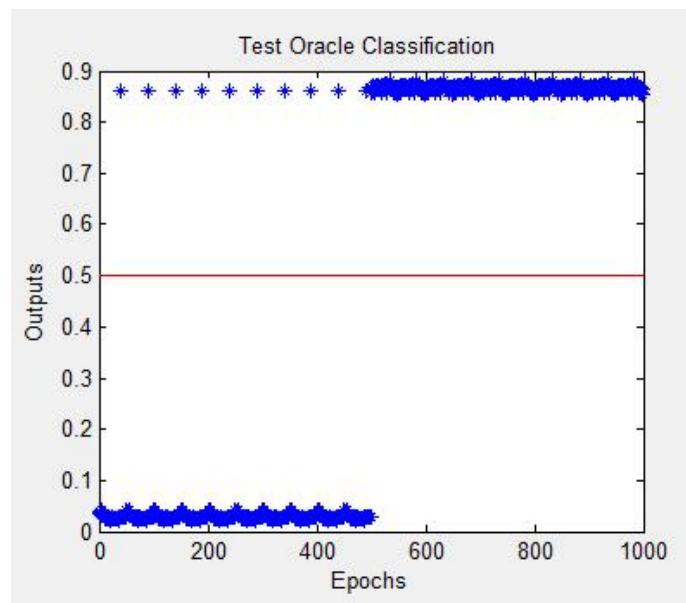
**Figure 4.16:** Testing done with 100 test cases

Figure 4.17 shows outputs of a dataset with 500 test samples. The 100 test samples in Figure 4.16 has been replicated 4 times, starting with the first 50, just after the 50<sup>th</sup> test sample four times, then the same to the last 50 as well.



**Figure 4.17:** Testing done with 500 test cases

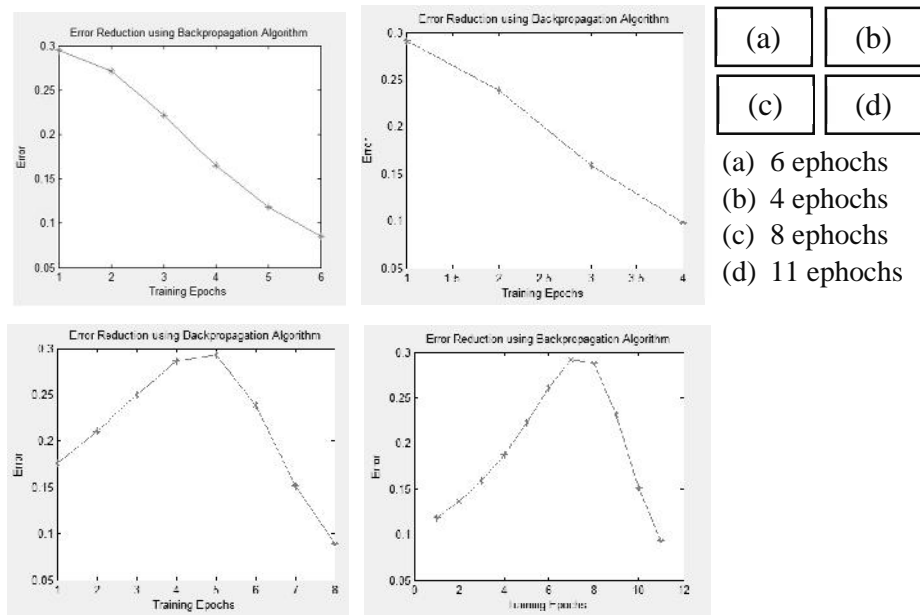
The outputs of a dataset with 1000 test samples is shown in Figure 4.18. The 500 test samples in Figure 4.17 has been replicated once on the same dataset.



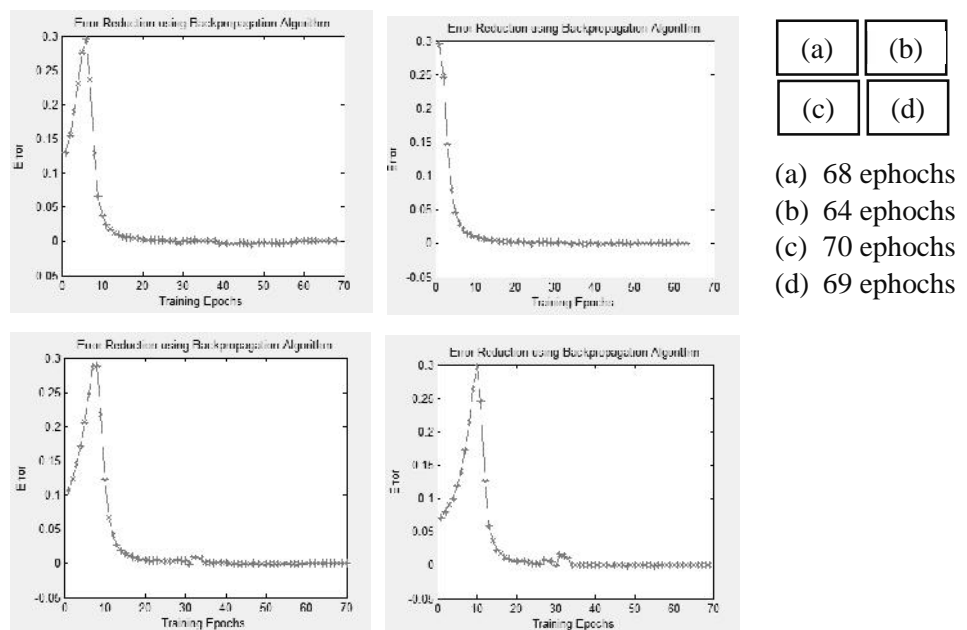
**Figure 4.18:** Testing done with 1000 test cases

### 4.3.4 ERROR THRESHOLD VARIATIONS

In this section we analyzed how the error gets reduced based on the error threshold value when a single test sample is given. The number of iterations differs due to the random initialization of weights. Figure 4.19 (a), (b), (c), (d) shows four different graphs from the same input set when the error threshold was 0.1. The average absolute error of a single training sample was 0.18751 and of the whole training set in Figure 4.20 (a), (b), (c), (d) was 0.01705 for training 4 times.

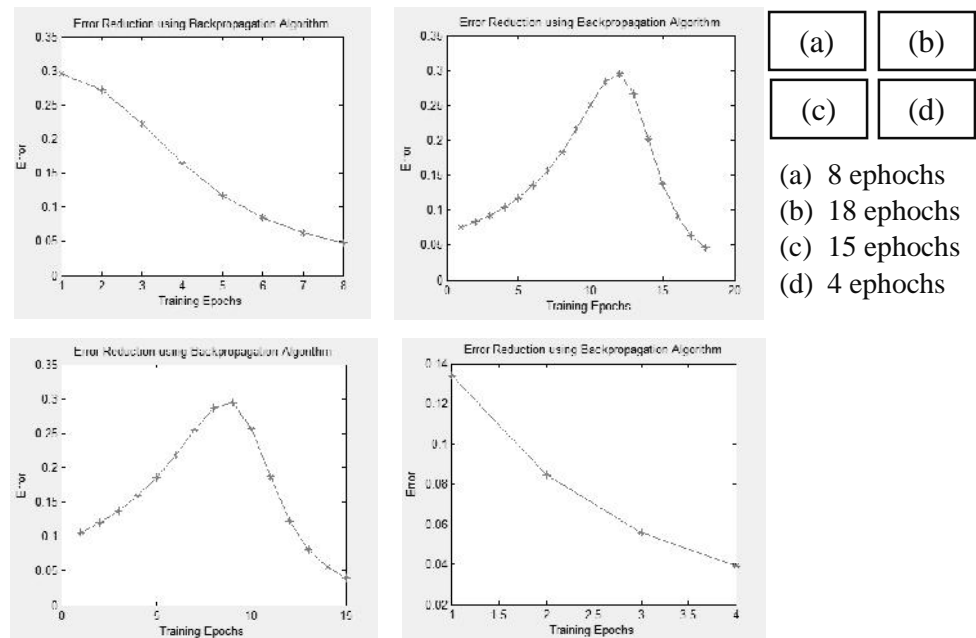


**Figure 4.19:** Error Reduction with 1 Sample when the Error Threshold is 0.1

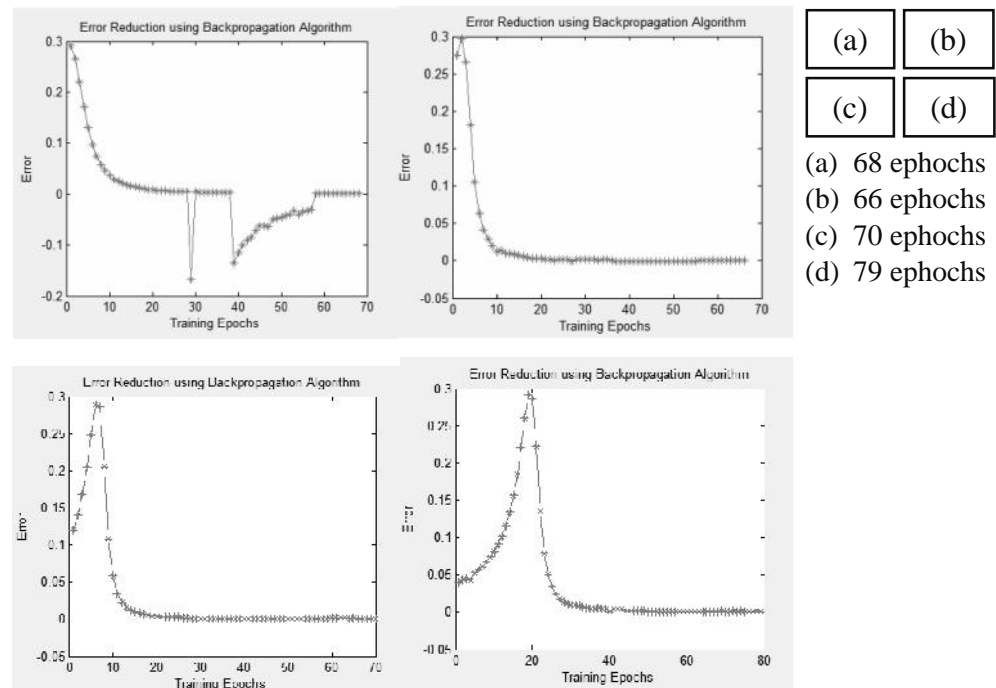


**Figure 4.20:** Error reduction with 60 samples when the Error Threshold is 0.1

Figure 4.21 (a), (b), (c), (d) and Figure 4.22 (a), (b), (c), (d) shows error reduction outputs when the error threshold was 0.05. The average absolute error of a single training sample was 0.17255 and of the whole training sets was 0.015182 for training 4 times.



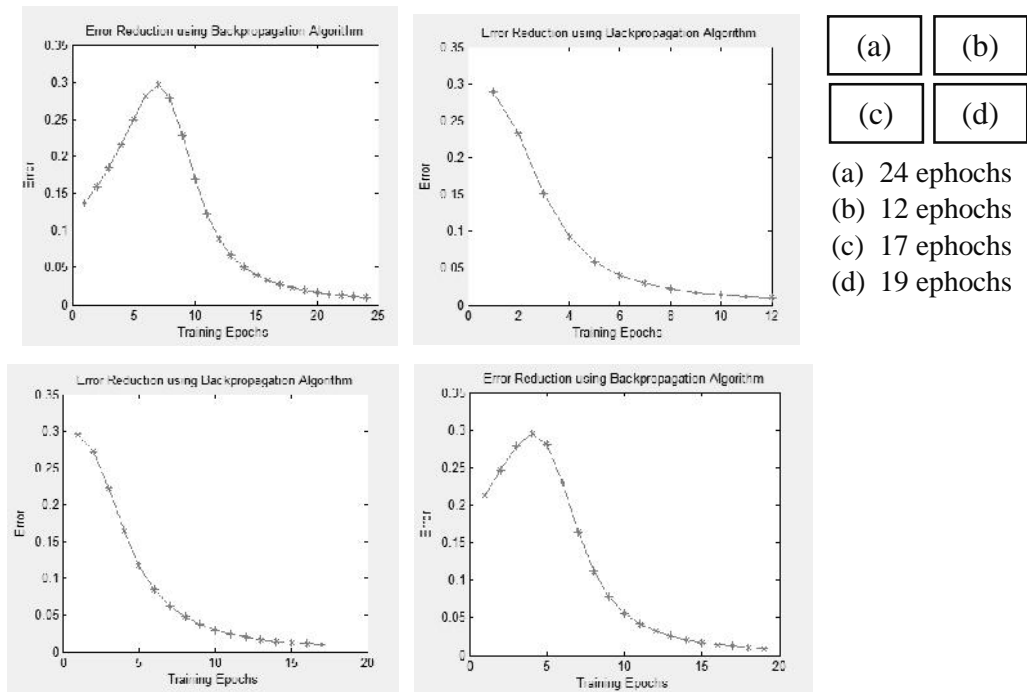
**Figure 4.21:** Error Reduction with 1 Sample when the Error Threshold is 0.05



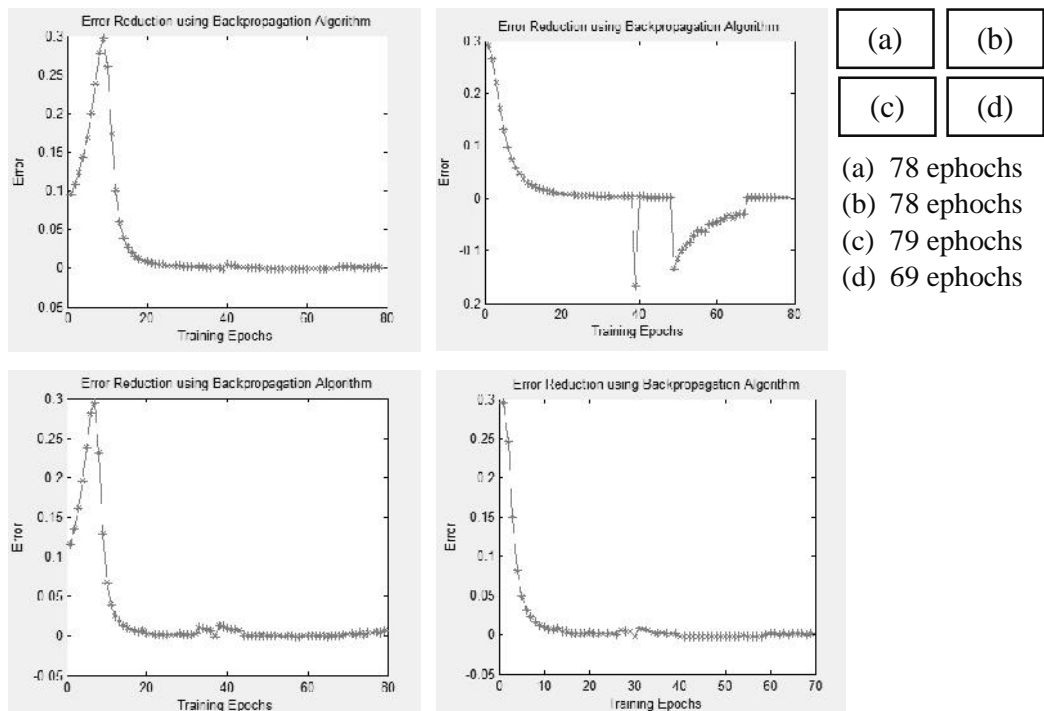
**Figure 4.22:** Error Reduction with 60 Samples when the Error Threshold is 0.05



Error reduction outputs are shown in Figure 4.23 (a), (b), (c), (d) and Figure 4.24 (a), (b), (c), (d) when the error threshold was 0.01. The average absolute error of a single training sample was 0.12048 and of the whole training set was 0.013723 for training 4 times.

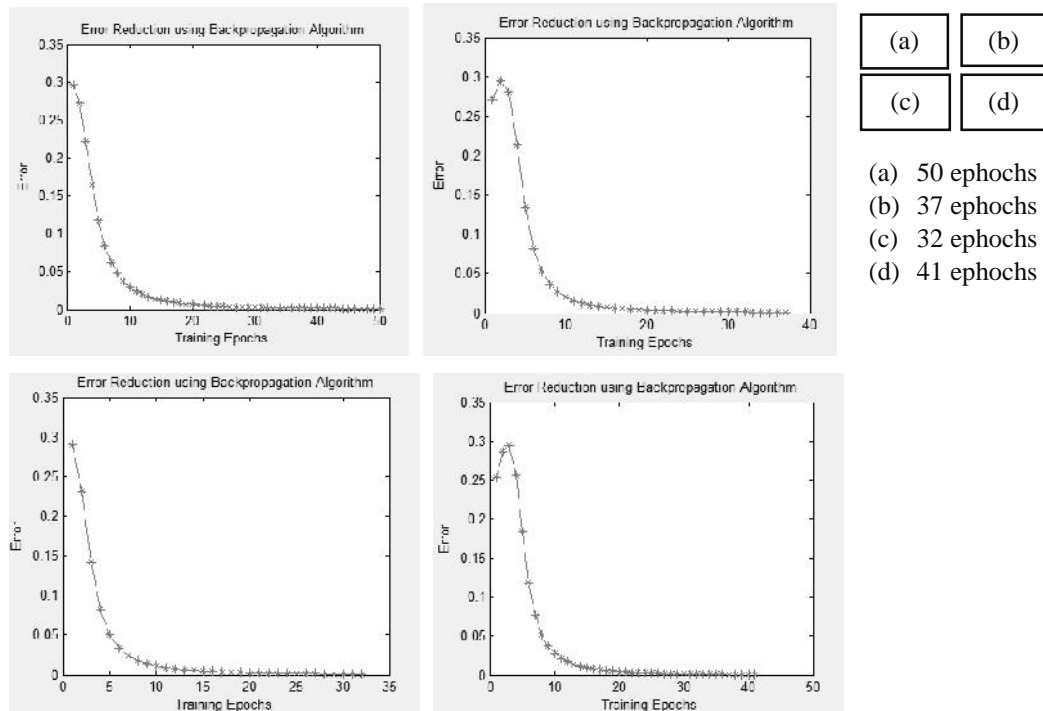


**Figure 4.23:** Error Reduction with 1 Sample when the Error Threshold is 0.01

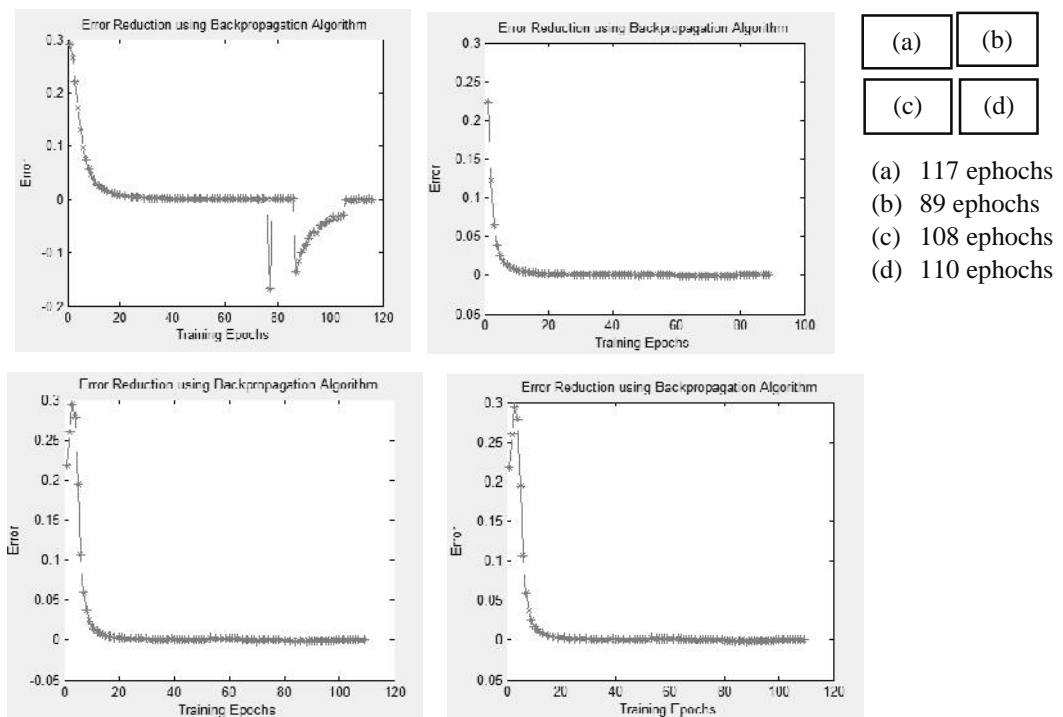


**Figure 4.24:** Error Reduction with 60 Samples when the Error Threshold is 0.01

The outputs of reducing errors are shown in Figure 4.25 (a), (b), (c), (d) and Figure 4.26 (a), (b), (c), (d) when the error threshold was 0.001. The average absolute error of a single training sample was 0.046066 and of the whole training set was 0.010186.



**Figure 4.25:** Error Reduction with 1 Sample when the Error Threshold is 0.001



**Figure 4.26:** Error Reduction with 60 Samples when the Error Threshold is 0.001

The Table 4.3 below summarizes the effects of giving a single training sample. Based on the different error threshold values that were experimented, it shows that a higher error threshold value results in fewer iterations executed to reach a shopping criteria and a higher percentage of the absolute error.

**Table 4.3:** Summary of Effects of Error Threshold of 1 training sample.

<b>Error Threshold</b>	<b>Average Absolute Error</b>	<b>Average Epochs</b>
0.1	18.751%	7
0.05	17.255%	11
0.01	12.048%	18
0.001	4.6066%	40

The highest average absolute error 18.751% was obtained when the error threshold was 0.1 and the least average number of epochs 7, while with the smallest error threshold 0.001, the MLP performed an average of 40 iterations to reach the stopping criteria while the absolute error was 4.6066%.

60 training samples were given to train the MLP on different error threshold values. The calculated absolute error and number of epochs performed are shown below in Table 4.4. Based on the results shown in Table 4.3 and in Table 4.4, it shows that the greatest error values are reduced by the first training sample.

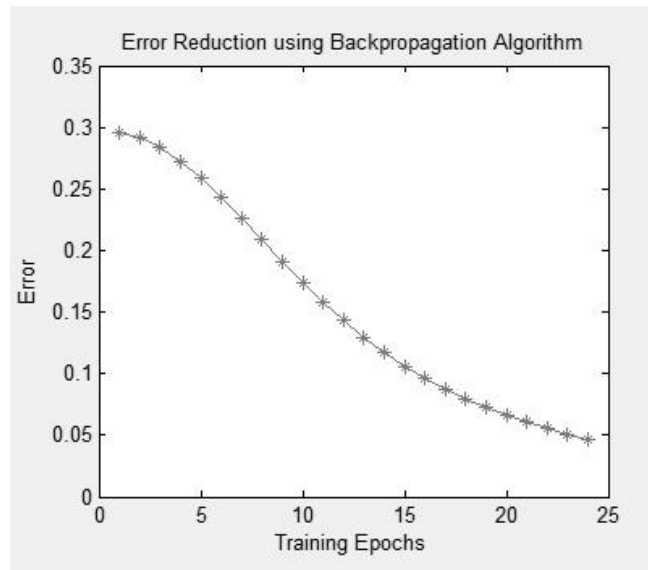
**Table 4.4:** Summary of Effects of Error Threshold of 60 training samples.

<b>Error Threshold</b>	<b>Average Absolute Error</b>	<b>Average Epochs</b>
0.1	1.705%	68
0.05	1.5182%	71
0.01	1.3723%	76
0.001	1.0186%	106

Therefore, 0.1 error threshold was chosen to be used in the MLP network because of the highest average absolute error percentage and the least average number of epochs to reach the stopping criteria.

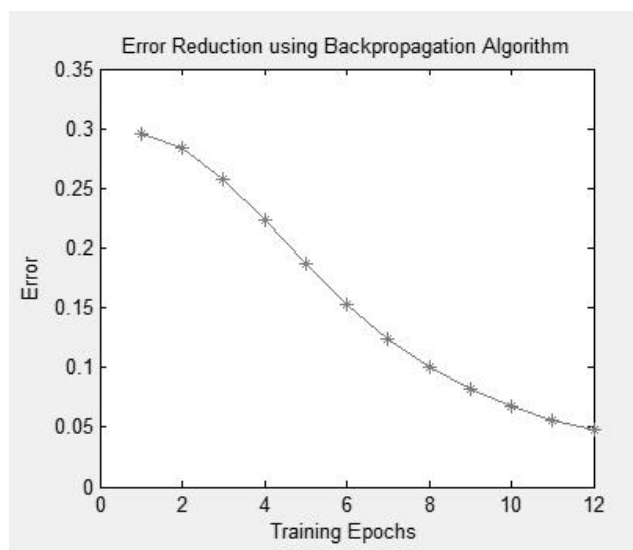
### 4.3.5 LEARNING RATE VARIATIONS

The learning rate determines how much the connection weights can be modified based on the direction change and change rate. In order to evaluate the effects of changing the learning rate, we have kept the weights constant. Figure 4.27 shows how error gets reduced using the backpropagation algorithm when the learning rate is 0.1. The first input set iterates 24 times to reach the error threshold value.



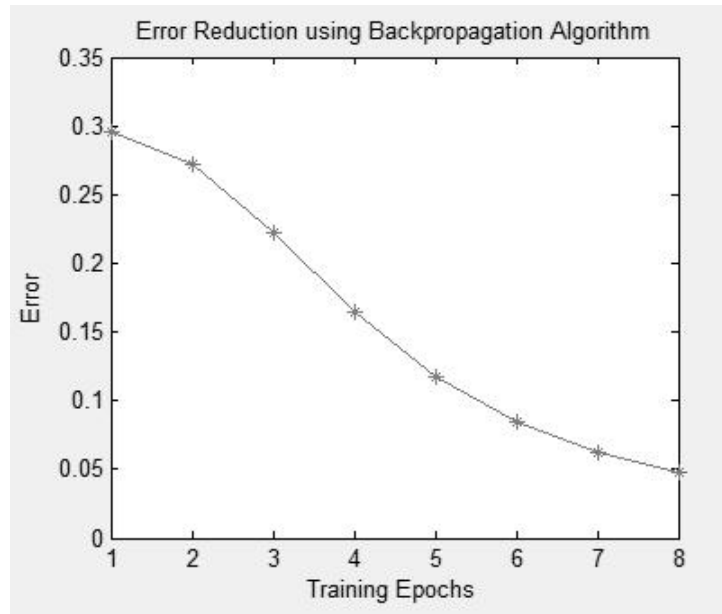
**Figure 4.27:** Error reduction with 1 Sample when the Learning Rate is 0.1

Output of error reduction using the backpropagation training algorithm is shown in Figure 4.28 when the learning rate is 0.2. The first input set iterates 12 times to reach the error threshold value.



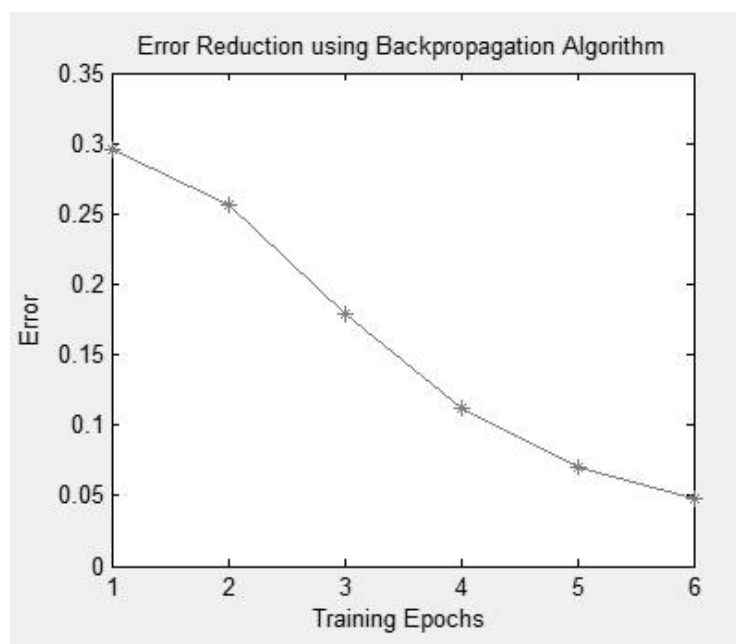
**Figure 4.28:** Error reduction with 1 Sample when the Learning Rate is 0.2

In Figure 4.29, output of error reduction using a backpropagation training algorithm is shown when the learning rate is 0.3. The first input set iterates 8 times to reach the error threshold value.



**Figure 4.29:** Error reduction with 1 Sample when the Learning Rate is 0.3

Figure 4.30 shows how error gets reduced using a backpropagation algorithm when the learning rate is 0.4. The first input set iterates 6 times to reach the error threshold value.



**Figure 4.30:** Error reduction with 1 Sample when the Learning Rate is 0.4

---

The summary of learning rate is shown in Table 4.5. When the learning rate is high (maximum of 1.0), the network trains faster. However, the chances of being trained to reach a local minimum solution are higher.

**Table 4.5:** Learning Rate effects on Epochs.

Learning Rate	Epochs
0.1	24
0.2	12
0.3	8
0.4	6

Therefore, 0.3 was chosen as the learning rate in our MLP network as it provides optimum performance and has a better chance to stabilize at the most optimal global solution.

#### 4.4 SUMMARY OF EXPERIMENTAL RESULTS

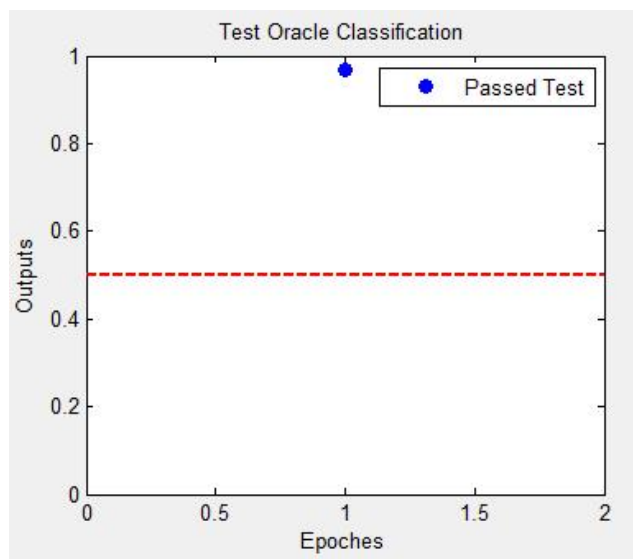
Based on the experiments done in this chapter, the following parameters have been chosen to be used in the MLP network for predicting exploratory test oracles.

- The Sigmoid Activation Function in the Hidden Layer
- The Hyperbolic Tangent Activation Function in the Output Layer
- 4 neurons in the hidden layer
- 0.1 error threshold value
- 0.3 learning rate

With these chosen parameters, the MLP networks trains and tests well when provided with suitable datasets (test cases) that corresponds to the simulated application under test.

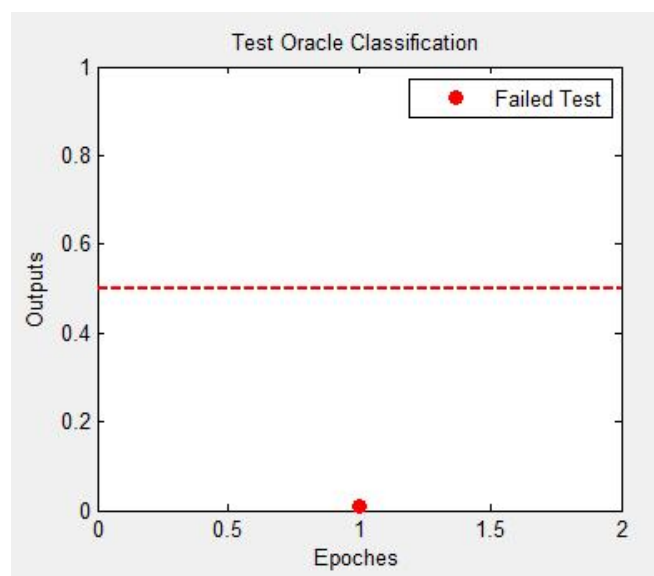
## 4.5 EXPLORATORY TEST ORACLE

When all experiments are carried out and the optimal Exploratory Test Oracle based MLP-NN parameters are chosen, then the network can be used to generate exploratory test oracles. Figure 4.31 shows the expected output, which is a Passed Test when a set of the correct inputs are given. The dotted red line is the threshold demarcating the passed from the failed tests.



**Figure 4.31:** Expected Passed Test

The expected failed test run is shown in Figure 4.32 which reflect a set of incorrect inputs given or a faulty in a program.



**Figure 4.32:** Expected Failed Test

Figure 4.31 and Figure 4.32 may represent false expected passed test (False Positive) and false failed test (False Negative) respectively. This happens when there is a programming error or when the AUT has not been simulated properly.



# **CHAPTER 5**

## **CONCLUSION AND FUTURE WORK**

## CONCLUSION AND FUTURE WORK

The thesis work has been focused on application of the multilayer perceptron neural network as an exploratory test oracle to generate the expected outputs of the AUT. The AUT with different data types was used for experimental purposes and showed that it cannot learn properly from summed ASCII codes as correct and wrong inputs cannot be distinguished easily by the MLP-NN. This is because the value of the correct and wrong can be similar or they may be more or less the same or fall in the same range.

The common MLP-NN with one weight on each connection failed to classify the transformed data correctly and hence we modified it to two weighted MLP-NN. The improved MLP-NN with two weights on each connection perfectly generated reliable exploratory test oracles. The MLP-NN selects a greater weight for a correct input and smaller weight for incorrect input. These weights were adjusted by backpropagation algorithm to an acceptable error value.

Based on the experiments, the following parameters have been chosen to be used in the MLP-NN for predicting the exploratory test oracles; Sigmoid activation function in the Hidden Layer, Hyperbolic Tangent activation function in the Output Layer, 4 neurons in the hidden layer, 0.1 error threshold and 0.3 learning rate. With these chosen parameters, the MLP-NN trains and tests well when provided with the suitable datasets (test cases) that corresponds to the simulated AUT. The exploratory testers tests the AUT to produce the actual outputs while the exploratory test oracle produce the approximate or partial expected outputs. Using tester's knowledge and interpretation, the expected and actual outputs are compared and a fault report is produced.

The future scope of this research work is to explore on how exploratory test oracle based MLP-NN can be used to generate oracles in integration and system testing. The comparison of actual and partial output need to be automated to fully reduce reliance on human decisions. Thus, the exploratory tester would concentrate more on mundane tasks of designing and executing tests

***This thesis has been converted into a research paper submitted on the 20<sup>th</sup> of June 2014 to IEEE Software Engineering Journal for review.***

---

---

## REFERENCE

1. Fewster, M., Graham, D., 1999. Software test automation. Addison-Wesley Harlow, Essex, UK.
2. Everett, G., McLeod, R., 2007. Software testing: testing across the entire software development life cycle. Wiley-IEEE Computer Society Pr.
3. Itkonen, J., Mantyla, M., Lassenius, C., 2007. Defect detection efficiency: Test case based vs. exploratory testing. In: Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. pp. 61–70.
4. Itkonen, J., Rautiainen, K., 2005. Exploratory testing: a multiple case study. In: Empirical Software Engineering, 2005. 2005 International Symposium on. p. 10 pp.
5. Bach, J., 2000. Session-based test management. Quardev Laboratories.
6. Naseer, A., Zulfiqar, M., 2010. Investigating exploratory testing in industrial practice, Blekinge Institute of Technology, COM/School of Computing.
7. Myers, G. J., Feb. 1979. The Art of Software Testing, 1st Edition. John Wiley & Sons.
8. Tinkham, A., Kaner, C., 2003. Exploring Exploratory Testing. In: STAR East conference, [www.testingeducation.org](http://www.testingeducation.org).
9. Bach, J., Kaner, C., March 2001. Demonstrating Exploratory Techniques. In: Workshop on Heuristic & Exploratory Techniques (WHET 2). Front Royal, VA.
10. Bolton, M., September 2009a. Exploratory approaches in agile projects. An online Webinar with Chad Wathington from ThoughtWorks.
11. Itkonen, J., Rautiainen, K., 2005. Exploratory testing: a multiple case study. In: Empirical Software Engineering, 2005. 2005 International Symposium on. p. 10 pp.
12. L. Shoaib, A. Nadeem, and A. Akbar, “An empirical evaluation of the influence of human personality on exploratory software testing,” in Proceedings of IEEE 13th International Multitopic Conference, 2009 No. 1-6.
13. Kalman, S., May 2007. Session-based test lite. url <http://www.quardev.com/content/whitepapers/SBTMLitesamkalman.pdf>.

- 
- 
14. Lyndsay, J., 2003. A Positive View of Negative Testing. Workroom Productions Ltd.
  15. Kaner, C., 2006c. Exploratory testing after 23 years. In: Conference of the Association for Software Testing.
  16. Itkonen, J., Mantyla, M. V., Lassenius, C., 2009. How do testers do it? An exploratory study on manual testing practices. In: Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on. pp. 494–497.
  17. Itkonen, J.; Mantyla, M.V.; Lassenius, C., "The Role of the Tester's Knowledge in Exploratory Software Testing," Software Engineering, IEEE Transactions on , vol.39, no.5, pp.707,724, May 2013
  18. A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp, Guide to the Software Engineering Body of Knowledge 2004 Version. Los Alamitos, CA, USA: IEEE Computer Society, 2004.
  19. L. Baresi and M. Young, "Test oracles," University of Oregon, Dept. of Computer and Information Science, Eugene, Oregon, U.S.A., Tech. Rep. Technical Report CISTR- 01-02, Aug. 2001.
  20. Peters, D., and Parnas, D.L.: 'Generating a test oracle from program documentation', in Editor (Ed.)^(Eds.): 'Book Generating a test oracle from program documentation' (ACM, 1994, edn.), pp. 58
  21. Bousquet, L.d., Ouabdesselam, F., Richier, J.L., and Zuanon, N.: 'Lutess: a specification-driven testing environment for synchronous software'. In Proceedings of the 21st International Conference on Software Engineering (May 1999), ACM Press, pp. 267-276.
  22. Dillon, L.K., and Ramakrishna, Y.S.: 'Generating oracles from your favorite temporal logic specifications', SIGSOFT Softw. Eng. Notes, 1996, 21, (6), pp. 106-117.
  23. Richardson, D.J., Aha, S.L., and O'Malley, T.O.: 'Specification-based Test Oracles For Reactive Systems', in Editor (Eds.): 'Book Specification-based Test Oracles For Reactive Systems' (1992, edn.), pp. 105-118.
  24. Hall, P.A.V.: 'Relationship between specifications and testing', Information and Software Technology, 1991, 33, (1), pp. 47-52

- 
- 
25. Stocks, P., and Carrington, D.: 'A framework for specification-based testing', *IEEE Transactions on Software Engineering*, 1996, 22, (11), pp. 777-793
  26. Schroeder, P.J., Faherty, P., and Korel, B.: 'Generating expected results for automated black-box testing', in Editor (Eds.): 'Book Generating expected results for automated black-box testing' (2002, edn.), pp. 139-148.
  27. Kim-Park, D.S.; de la Riva, C.; Tuya, J., "A Partial Test Oracle for XML Query Testing," *Testing: Academic and Industrial Conference - Practice and Research Techniques*, 2009. TAIC PART '09. , vol., no., pp.13,20, 4-6 Sept. 2009.
  28. Shukla, R.; Carrington, D.; Strooper, P., "A passive test oracle using a component's API," *Software Engineering Conference*, 2005. APSEC '05. 12th Asia-Pacific , vol., no., pp.7 pp.,, 15-17 Dec. 2005.
  29. S. H. Edwards, M. Sitaraman, B. W. Weide, and J. Hollingsworth, "Contract-checking wrappers for C++ classes," *IEEE Transactions on Software Engineering*, vol. 30, no. 11, pp. 794-810, 2004.
  30. I. Gondra, "Applying machine learning to software fault-proneness prediction," *J. Syst. Softw.*, vol. 81, no. 2, pp. 186–195, Feb. 2008.
  31. Binder Robert V., *Testing Object-Oriented Systems-Models, Patterns, and Tools*. Addison Wesley 1999.
  32. J. H. Andrews and Y. Zhang, "General Test Result Checking with Log File Analysis," *IEEE Transactions on Software Engineering*, vol. 29(7), pp. 634-648, July 2003.
  33. Spillner, A., Linz, T., Schaefer, H.: *Software Testing Foundations*, 2nd edn. Rocky Nook Inc, Santa Barbara (2007).
  34. Ntafos, S.C.: On comparisons of random, partition, and proportional partition testing. *IEEE Trans. Softw. Eng.* 27(10), 949–960 (2001).
  35. Jorgensen, P.C.: *Software Testing: A Craftsman's Approach*, 2nd edn. CRC Press LLC, Boca Raton (2002)
  36. Pfleeger, S.L., Hatton, L.: Investigating the influence of formal methods. *Computer* 30(2), 33–43 (1997).
  37. Kessentini, M., Sahraoui, H., Boukadoum, M.: Example-based model-transformation testing. *Autom. Softw. Eng.* 18(2), 199–224 (2011b).

- 
- 
38. Woodward, M.R., Hennell, M.A.: On the relationship between two control-flow coverage criteria: all JJpaths and MCDC. *Inf. Softw. Technol.* 48(7), 433–440 (2006).
  39. Michael, C.C., McGraw, G., Schatz, M.A.: Generating software test data by evolution. *IEEE Trans. Softw. Eng.* 27(12), 1085–1110 (2001)
  40. Memon, A.M., Pollack, M.E., Soffa, M.L.: Automated test oracles for GUIs. *SIGSOFT Softw. Eng. Notes* 25(6), 30–39 (2000).
  41. Shahamiri, S.R.; Kadir, W.M.N.W.; bin Ibrahim, S., "An automated oracle approach to test decision-making structures," *Computer Science and Information Technology (ICCSIT)*, 2010 3rd IEEE International Conference on , vol.5, no., pp.30,34, 9-11 July 2010.
  42. Staats, M.; Gay, G.; Heimdahl, M.P.E., "Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing," *Software Engineering (ICSE)*, 2012 34th International Conference on , vol., no., pp.870,880, 2-9 June 2012.
  43. Huai Liu; Fei-Ching Kuo; Towey, D.; Tsong Yueh Chen, "How Effectively Does Metamorphic Testing Alleviate the Oracle Problem?," *Software Engineering, IEEE Transactions on* , vol.40, no.1, pp.4,22, Jan. 2014.
  44. Monisha, T.R.; Chamundeswari, A., "Automatic verification of test oracles in functional testing," *Computing, Communications and Networking Technologies (ICCCNT)*, 2013 Fourth International Conference on , vol., no., pp.1,4, 4-6 July 2013.
  45. Padgham, L.; Zhiyong Zhang; Thangarajah, J.; Miller, T., "Model-Based Test Oracle Generation for Automated Unit Testing of Agent Systems," *Software Engineering, IEEE Transactions on* , vol.39, no.9, pp.1230,1244, Sept. 2013.
  46. Weifeng Xu; Tao Ding; Hanlin Wang; Dianxiang Xu, "Mining Test Oracles for Test Inputs Generated from Java Bytecode," *Computer Software and Applications Conference (COMPSAC)*, 2013 IEEE 37th Annual , vol., no., pp.27,32, 22-26 July 2013.
  47. Weifeng Xu; Hanlin Wang; Tao Ding, "Mining Auto-generated Test Inputs for Test Oracle," *Information Technology: New Generations (ITNG)*, 2013 Tenth International Conference on , vol., no., pp.89,94, 15-17 April 2013.

- 
- 
48. Hu Jin; Yi Wang; Nian-Wei Chen; Zhi-Jian Gou; Shuo Wang, "Artificial Neural Network for Automatic Test Oracles Generation," Computer Science and Software Engineering, 2008 International Conference on , vol.2, no., pp.727,730, 12-14 Dec. 2008.
  49. Xiaoying Bai; Kejia Hou; Hao Lu; Yao Zhang; Linping Hu; Hong Ye, "Semantic-Based Test Oracles," Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual , vol., no., pp.640,649, 18-22 July 2011.
  50. Yan Lei; Xiaoguang Mao; Tsong Yueh Chen, "Backward-Slice-Based Statistical Fault Localization without Test Oracles," Quality Software (QSIC), 2013 13th International Conference on, vol., no., pp.212-221, 29-30 July 2013.
  51. Kanstren, T., "Program Comprehension for User-Assisted Test Oracle Generation," Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on, vol., no., pp.118-127, 20-25 Sept. 2009.
  52. Farn Wang; Li-Wei Yao; Jung-Hsuan Wu, "Intelligent Test Oracle Construction for Reactive Systems without Explicit Specifications," Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on , vol., no., pp.89,96, 12-14 Dec. 2011.