5-12-2004

A Comparative Study Of Artificial Neural Networks And Info Fuzzy Networks On Their Use In Software Testing

Deepam Agarwal
University of South Florida

A Comparative Study Of Artificial Neural Networks And Info Fuzzy Networks On Their Use In Software Testing

by

Deepam Agarwal

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science & Engineering
College of Engineering
University of South Florida

Major Professor: Abraham Kandel, Ph.D.
Dewey Rundus, Ph.D.
Miguel Labrador, Ph.D.

Date of Approval:
May 12, 2004

Keywords: Automated Oracle, Regression Testing, ROC Analysis

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Abraham Kandel for his constant support and belief in my abilities, and for providing me with the opportunity to work on this project.

I am thankful to Dr. Mark Last who has been supervising me with patience and has been invaluable in enhancing both my technical and writing skills. Without his guidance my thesis wouldn't have been possible.

I extend my thanks to Dr. Dewey Rundus, Dr. Miguel Labrador, Dr. Rosina Weber for selflessly extending their support and help whenever it was needed.

I would like expressing my thanks to all my co-workers and staff at National Institute of Systems Test and Productivity and my friends, University of South Florida, for helping me on this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

A COMPARATIVE STUDY OF ARTIFICIAL NEURAL NETWORKS AND INFO FUZZY NETWORKS ON THEIR USE IN SOFTWARE TESTING

Deepam Agarwal

ABSTRACT

It is very important that the software being delivered to the user is reliable and fault free. This makes software testing one of the most important phases in the software development life cycle. The problem being faced by everyone is the time it takes to test the software, which is normally huge. An important part of the software testing process is running and evaluating test scenarios. The objective of this part is to evaluate how well the application under test conforms to its specifications. One of the ways to achieve this is to generate the test cases and make use of the test oracle (a human expert) to determine whether a given test case exposes a fault. This procedure consumes a lot of time. Using an automated oracle can contribute towards the reduction in software testing time which helps in the reduction of the cost of the testing process.

The use of Artificial Neural Networks (ANN) and Info-Fuzzy Networks (IFN) for test case selection and evaluation has already been explored. In this thesis these two approaches are compared on their use as an automated oracle.

An ROC Analysis is done to compare the two approaches. The execution times of both the approaches are also compared. For comparison, three applications have been used. The basic methodology behind the use of IFN or ANN is to train the network on randomly generated test cases executed with a stable version of the software. This trained network is then used as an oracle for evaluating the correctness of the output produced by new and possibly faulty versions of the stable software. The outputs from the oracle i.e. IFN or ANN and faulty versions of the software system are compared with that of the original version to evaluate the outputs generated by new version of the software.

CHAPTER 1

INTRODUCTION

As we know that creating a reliable and hence a fault free software is one of the major goals of a software developer. This makes software testing one of the most important and critical phases in the software development life cycle.

To plan and execute tests, software testers must consider the software and the function it computes, the inputs and how they can be combined and the environment in which the software will eventually operate. This difficult, time-consuming process requires technical sophistication and proper planning.

The process of software testing includes four phases [6] namely modeling the software's environment, selection of test cases, running and evaluating test cases and measuring testing progress. As the test case automation is increasing the volume of the test is growing. It makes the selection of test cases very difficult, making the test case reduction highly desirable. Test case reduction using algebraic constraints is covered in [21]. The use of ANN for test case reduction has been discussed in [2,3,25]. IFN has also been used for reduction of number of test cases in [19,20].

After selecting and running the test cases the tester has to evaluate whether the selected test cases expose a fault or not. A test oracle is needed to determine this.

Traditionally this step was done manually by the human tester, which required a lot of time. As the software systems are growing larger the burden on the human tester is increasing. Using an automated oracle to support the activities of human tester can reduce the cost of the testing process and hence the related maintenance costs [16]. The use of ANN and IFN as an automated oracle for tested systems has been explored in [16, 17] and [14] respectively.

Since both ANN and IFN have been used as an automated oracle it makes very important to compare these two approaches to evaluate the effectiveness of one methodology over the other. In this thesis a detailed comparison is performed between these two approaches. ROC Analysis is done to compare the accuracy of the two methodologies. The execution times are also calculated to compare the speed of both methods.

For the comparison between ANN and IFN experiments were performed using the following three applications, Pay Calculator, Risk Calculator and the Loan Calculator. As explained in [17] and [14] for using ANN and IFN as an automated oracle, both networks are trained according to their respective training methodology by using randomly generated input data. This random input data conforms to the program's specifications. The program here refers to the program under test. After this step the trained network

becomes a simulated model of the original program and hence can be used to test different faulty versions of the same program. When the new (faulty) versions have to be tested the same input set is executed on both the new version and the trained network. The outputs from both new version and trained network are compared and the validity of the output is determined. It is also assumed that there is no change in the original requirements in the newer versions of the program.

The organization of the thesis is as follows. In the next chapter a literature review is presented. In chapter 3 the testing methodology is explained in detail. In chapter 4 the experiments conducted on the three test beds are explained along with the complete description of the testbeds themselves. In Chapter 5 the results of the comparison between the Neural Network and Info Fuzzy networks are presented. Chapter 6 presents a discussion of the future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

In this chapter a basic description of ANN and IFN is presented. A literature survey is also presented which throws some light on the work already been done in the field of software testing using these methodologies.

2.2 Artificial Neural Networks (ANN)

An ANN is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, processes information. Either humans or other computer techniques can use neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, to extract patterns and detect trends that are too complex to be noticed. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.7

The key element of this paradigm is the novel structure of the information processing system. It consists of a number of processing units or neurons, which are connected to each other through, weighted links and each of these links has its own value

known as weight of the link [27]. These units (neurons) are organized into several layers, namely input layer hidden layer and output layer. The input layer receives an external activation vector and passes it via weighted connections to the units in the first hidden layer. These hidden layers compute their other activation and pass them to neurons in succeeding layers. From a distal point of view an arbitrary input vector is propagated forward through the network, finally causing an activation vector in the output layer. The entire network function that maps the input vector onto the output vector is determined by the connection weights of the net. The structure of a basic unit (neuron) is shown in the Figure 1 drawn below.

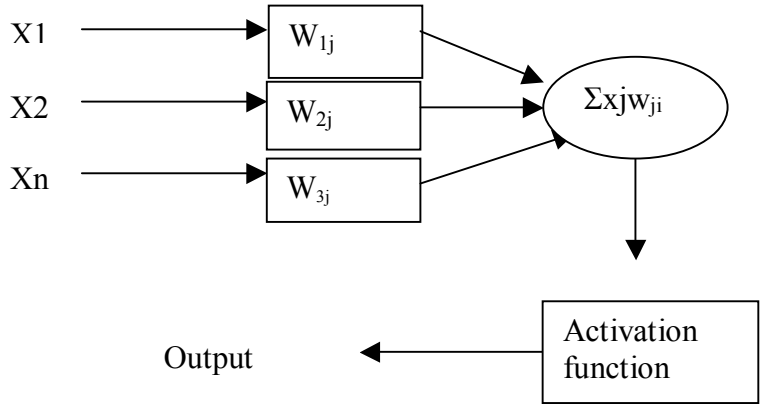

Figure 1 A Basic Neuron

As described above the neural network is composed of several such neurons. If the input signal to the neuron i is x_j, the weight value associated with neuron i and input xj is denoted by w_{ij}. The set of input signals is linearly combined as shown in (1) and the resulting value is then used to calculate the output of the activation function that consequently generates the neuron output as shown in (2). The activation function normally used is a sigmoid function and it generates the output between 0 and 1.

$$net = \Sigma xjwij \qquad (1)$$

$$output = 1/(1+e^{-net}) \quad (2)$$

The basic structure of a neural network is shown in the Figure 2. The diagram here shows the three layers mentioned above. All the neural networks have only one input layer corresponding to the inputs and one output layer corresponding to the output. The number of hidden layers can vary depending upon the target function which the network has to learn. It has been theoretically proven that three hidden layers are sufficient for approximating any given function [27].

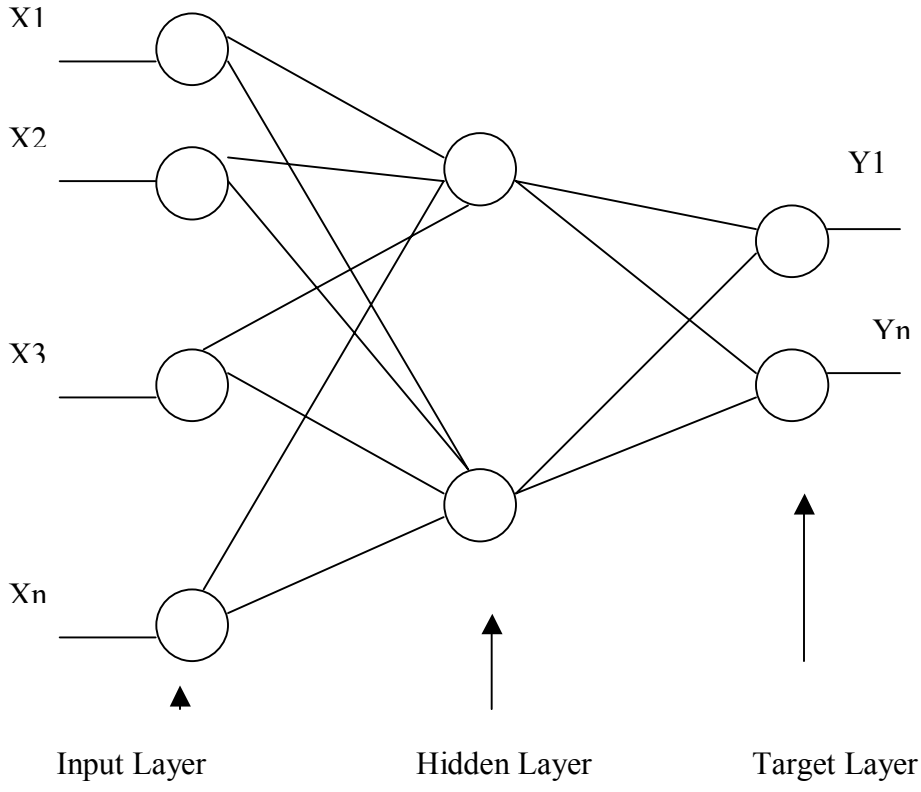

Figure 2 Structure Of A Neural Network

Each node in the hidden layer is fully connected to the inputs. That means what is learned in a hidden node is based on all the inputs taken together. This hidden layer is where the network learns interdependencies in the model.

This weighted sum is performed for each hidden node and each output node and that is how interactions are represented in the network. Using this value as the input the output the neural network is calculated by an activation function

After the calculation of the output, comes the next phase called training explained in the next section.

2.2.1 ANN Training

The main phases before the application of ANN to any kind of prediction problem is it's training. Training in layered neural networks refers to the modification of internal network parameters, so as to bring the function implemented by the network as close as possible to a desired function. It may be viewed as an optimization of the parameter set with respect to a set of training examples instancing the underlying rule. In this phase the neural network goes through incremental adaptation of connection weights that transport information between simple processing units. All learning methods used for adaptive neural networks can be classified into two major categories

Supervised Learning: which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. An important issue concerning supervised learning is the problem of error convergence, i.e. the minimization of error between the

desired and computed unit values. The goal is to determine a set of weights, which minimizes the error.

Unsupervised Learning uses no external teacher and is based upon only local information. It is also referred to as self-organization learning.

Neural network training composes of two parts forward pass and the backward pass. In the forward pass first each input pattern I_p is presented to the network and propagated forward to the output. The outputs are computed using sigmoid thresholding of the inner product of the corresponding weight and input vectors. As shown in Figure 2 all the outputs of one layer are connected as inputs for the other following layer. After this comes the backward pass in which the output values obtained from the forward pass are compared with the actual output to compute the value of some predefined error-function By various techniques the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to minimize the total error on the patterns in the training set. These weights are changed in proportion to the negative of an error derivative with respect to each weight. This method is known as gradient descent.

In the forward pass the step described above takes place i.e. evey node in the hidden unit and the output unit calculates its output using the activation function.This adjustment process can be done after the presentation of each record known as online or stochastic learning or after the the presentation of the whole epoch which is known as offline or batch learning [26,27]. Epoch means presentation of all the records in the training file to the neural network. The training procedure is concluded when either the

neural network acheives the desired accuracy or it runs for sufficiently large number of epochs. If one is satisfied with the accuracy then it is said that the network has *learned* a certain target function and hence the network is called a trained neural network. For online learning most common method used for training is the backpropagation algorithm. It uses gradient descent approach to find the weights that will yield the global minimum error.

The unique nature of neural networks to learn certain functionality makes them suitable to be applied to solve many classification problems. Neural networks have been used in the past to handle several aspects of software testing. Experiments have been conducted to evaluate the effectiveness of generating test cases capable of exposing potential faults [2], find faults in the failure data [3], reduce the number of test cases [19,20]. Neural networks have also been compared to other fault exposing techniques in [25]. The use of neural networks as an automated oracle is evaluated in [16, 17].

2.3 Info-Fuzzy Networks (IFN)

IFN is an approach developed for knowledge discovery and data mining. The interactions between the input and the target attributes of any type (discrete and continuous) are represented by an information theoretic connectionist network [18]. Like the structure of the ANN, IFN has changeable number of hidden layers and a single target layer. However, each hidden layer of an IFN represents an input attribute and not a weighted sum of input values. IFN also has a root node, which is not present in a neural network. An *l-th* hidden layer consists of nodes representing all possible combinations of first *l* input attributes. These nodes are associated with different equivalence classes

(values or intervals) of the corresponding input attribute. The target layer has a concept of category nodes just like in neural networks and decision graphs. Since a node in any hidden layer represents a combination of equivalence classes, each test case can be associated with one and only one node in every hidden layer, according to its own values of the input attributes. The network looks like as Figure 3

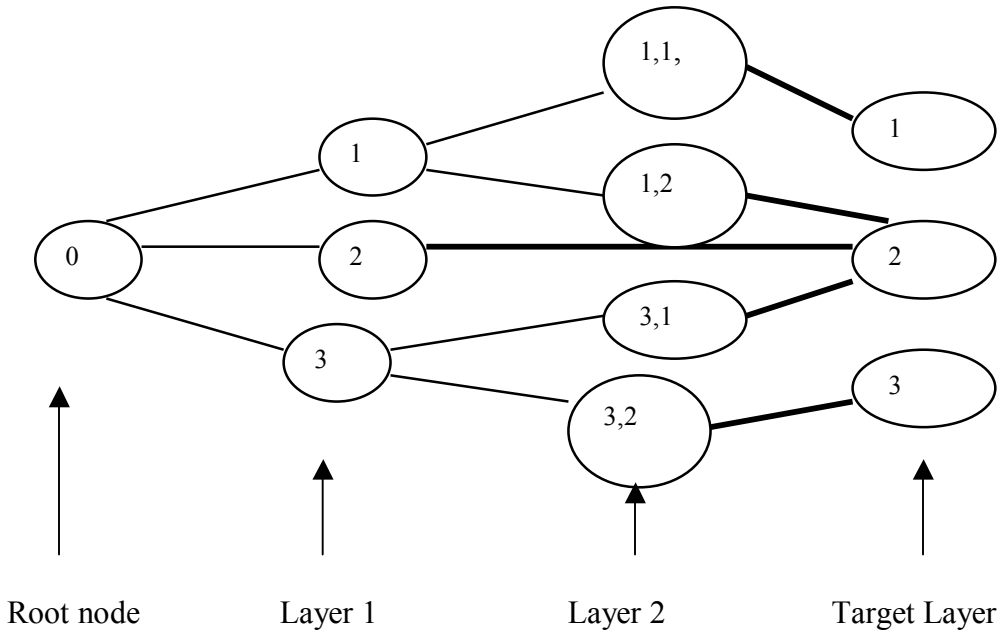

Figure 3 Structure Of An Info Fuzzy Network

Here the first layer corresponds to the first input variable, which has 3 equivalence classes and the second layer is for second input variable having two equivalence classes. As mentioned before the second layer consists of all the possible combinations of the nodes in the previous layers and the nodes in the second layer. As described in [11] the final nodes represent non-redundant combination of input variable, which produce a distinct output. In this case the final nodes are (1,1) (1,2), (2) (3,1).

These networks can be used to predict unknown values of target values in a manner similar to decision trees. But there is a difference between the two approaches. At

the first place IFN has a built-in feature selection capability by using the same input attribute at all nodes of each layer, which makes them "function graphs", and secondly IFN has interconnections between the terminal i.e. unsplit nodes and the target nodes. It has been shown that each discrete function has a unique (up to isomorphism) reduced function graph representation [14]. A detailed description and comparative evaluation of the info-fuzzy algorithm is provided in [12, 18].

The IFN methodology handles a selection of the most relevant features, extraction of informative rules and patterns, and post-processing of the extracted knowledge .The construction and the structure of the network depends on the amount of knowledge available from the data domain for example a single layer network or a double layer network can be constructed if detailed prior knowledge is available about the attributes and there dependency, a multi layer network has to be constructed if nothing in known about the attributes beforehand. For our experiments we will be using multilayer networks since we assume that nothing is known beforehand. The detailed description of the network construction procedure from the data available is presented in chapter 3. The next section provides an overview of the IFN construction procedure.

2.3.1 Network Construction

The learning procedure starts from a single node i.e. the root node. The single node in the network represents an empty set of input attributes. Like in decision trees, a node is split if there is a significant decrease in the conditional entropy of the target

attribute. If no input attribute is found to decrease the total conditional entropy of the target attribute across all nodes of the final layer then the network construction stops.

The unconditional and the conditional entropy of the target attribute are estimated by using the frequency estimators of conditional and unconditional probabilities of the target attribute values. The frequency estimators are calculated as proportions of records (cases) having certain values in a training set. The significance of decrease in the conditional entropy at a hidden node is measured by a likelihood ratio test explained in detail in Chapter 3.

IFN has been used in the past to reduce the number of test cases by the automated identification of relationships between inputs and outputs [11]. It has also been used as an automated oracle in [14] . IFN is also used for critical feature selection in [12] .

CHAPTER 3

DESCRIPTION OF THE METHODOLOGY

3.1 Overview

In this chapter the methodology of using the ANN and IFN as an automated oracle is explained. There are two separate stages, the first one being the training phase in which the ANN or the IFN are trained to be an oracle for the original tested system and second one being the evaluation phase in which the oracle is used to provide outputs, which together with the outputs of the original tested system are compared in the evaluation phase. The following sections provide the description of both ANN as well as the IFN.

3.2 ANN

For our application multi layer networks will be used. A separate network is created for every output. As described above the whole process is divided into two stages training and evaluation and the following subsection provide a description of both the stages in detail.

3.2.1 Training Stage

Supervised method of learning is used for our purpose i.e. the input vector to the network must have a corresponding output vector in order for it to learn the relationship.

All these input vectors are generated randomly according to the program specifications. The generation is done randomly to avoid clustered input vectors although random generation cannot ensure a complete coverage. Clustered inputs lead to bad training of the neural network. This randomly generated input vectors are executed on the tested program, which produces the corresponding output vectors for the inputs. After the generation, these input and output vectors are preprocessed. The preprocessing process of the input and outputs is covered in detail in Chapter 4. ANN parameters like the number of hidden layers, no of units in each layer, learning rate, number of epochs and no of output units in case of the continuous output have to be selected for training. ANN parameters to be selected depend upon the algorithm selected for training. There are several training algorithms [7,8,11] available for the neural network training but here for our experiments we have used the most common training algorithm, backpropagation. The algorithm is explained in Appendix A.

Trained ANN becomes a simulated model of the tested program. When a set of inputs is provided to the trained neural network, it provides the output according to tested program's specifications. This trained network is used as an automated oracle. Figure 4 summarizes the training process.

After training phase there is evaluation phase in which the effectiveness of the neural network as an auomated oracle is evaluated.

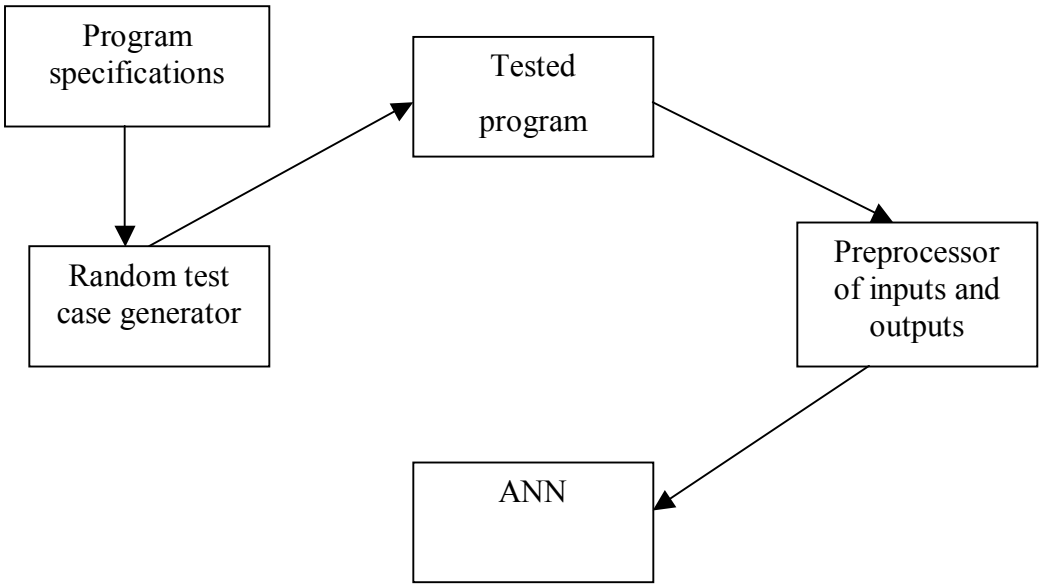

Figure 4 Training Phase Of ANN

3.2.2 Evaluation Phase

In the evaluation phase some faults are injected in the original tested program to create various mutated or faulty versions . Fault injection and the type of faults injected is covered in detail in chapter 4.After the injection of faults a separate set of input vectors is created the same way as the input vectors for the training file were created .These input vectors are now given to the faulty version of the program which results in a set of input and outputs. This set of inputs and outputs is preprocessed the same way as was done for the training file.The same set of preprocessed inputs is given to the trained ANN which results in a set of outputs of the network. The outputs of the ANN together with the output of the faulty version and a distance value calculated by the ANN are used to detect whether the output given by the faulty version is actually faulty or not. The outputs of the

faulty application and the neural network can lie in the following categories as shown in [16,17]

Table 1: Classification Of Output

ANN Output		Faulty Application Output	
		Correct	Wrong
	Correct	True Negative	True Positive
	Wrong	False Positive	False Negative

Since we are concerned with detection of faults, positive here means detection of faults by the ANN in the faulty output and hence we have the following meaning

True positive: Incorrect records classified as Incorrect

True negative: Incorrect records classified as Correct

False Positive: Correct records classified as Incorrect

False negative: Correct records classified as Correct

For our experiments we will be focusing on true positive and the false positive values.

Distance value is calculated between the neural network output and the corresponding application output. As explained in [16] the distance value will always be between (0.0, 1.0). This distance value along with the ANN output and the faulty application output is used to calculate whether the faulty application output is actually faulty or not. Figure 5 shows the summary of the evaluation phase.

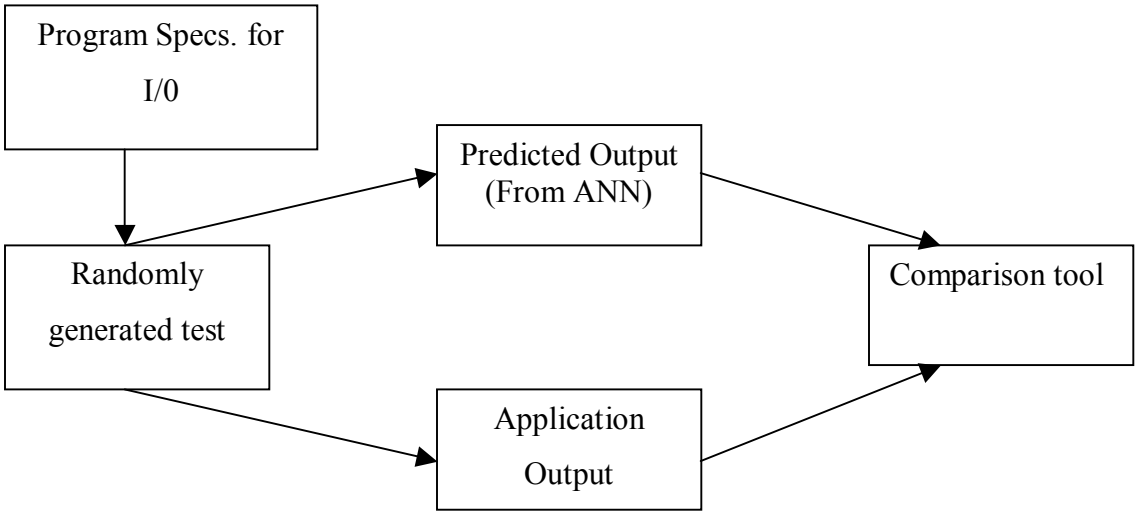

Figure 5 Evaluation Phase Of ANN

The comparison tool is hence used as an alternative to the human being who used to compare the results of the faulty versions and the original versions to tell whether the output is faulty or not. Table II provides a summary of how an output of the faulty application is classified as faulty or correct.

Table 2 Classification Of Faulty Application Output By The Neural Network

Condition	Oracle classification of the faulty application output
Oracle output = Faulty output and 0 < distance < L.T.	Correct
Oracle output != Faulty output and H.T. < distance < 1	Incorrect
Oracle output = Faulty Output and L.T. < distance < H.T.	Incorrect
Oracle output != Faulty Output and L.T. < distance < H.T.	Correct

L.T here stands for low threshold and H.T. stands for high threshold. There are no specific values for these thresholds already defined. These threshold values are determined experimentally to give the best overall results.

When the distance is between 0 and L.T and when it is between H.T. and 1 the neural network output is considered to be reliable otherwise it is considered to be unreliable and in both the cases the network output is likely to be correct and reliable enough to evaluate the correctness of the application output.

The training algorithm used i.e. the backpropogation algorithm has some problems associated with it. In the first place it is difficult to choose the learning rate appropriately as a good choice depends on the shape of the error function, which obviously changes with the learning task. A small learning rate results in long convergence time on a flat error function, where as a large learning rate will possibly lead to oscillations [15]. Another problem with backpropogation is the influence of the partial derivative on the size of the weight step. If the error function is shallow then the derivative is small resulting in small weight steps. On the other hand in the presence of

deep error function large derivatives lead to large weight steps due to which the algorithm is taken to a completely different weight space.

One of the ways to get around the problem is the introduction of a momentum term. Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low-pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network may get stuck in a shallow local minimum. With momentum a network can slide through such a minimum. Momentum can be added to backpropagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the backpropagation rule. A momentum constant, μ, mediates the magnitude of the effect that the last weight change is allowed to have which can be any number between 0 and 1. When the momentum constant is 0, a weight change is based solely on the gradient. When the momentum constant is 1, the new weight change is set to equal the last weight change and the gradient is simply ignored. The updated weight change formula looks like this

$$\Delta wij\ (t) = -E\ \delta E/\delta wij\ (t) + \mu\ \Delta wij\ (t-1)$$

But the optimal value of the momentum μ is also problem dependent, also it has been pointed out in [15] that this technique works well for many learning tasks, but is not general.

But in spite of all these problems in backpropogation it is shown in [16,19] that it performs well for the software testing applications. In this work neural network with momentum is used.

3.3 Info-Fuzzy Networks

We will be using multi layer information fuzzy networks. As in the case of neural network a new network is created for every output. Here also the output units are divided into equal frequency intervals as done for the neural networks. The process of making the IFN function as an automated oracle is divided into two parts the parts being the network construction and evaluation.

3.3.1 Network Construction Phase

We will be using supervised method of learning to train Information fuzzy networks. As opposed to online learning used for neural network training, in the case for IFN offline learning is used.

The overview of the training process for IFN is shown in Figure 6. The generation of inputs is again done randomly. These inputs are then given to the tested application and hence a file is obtained having both the set of inputs and the corresponding outputs. This file is used for building the info fuzzy network.

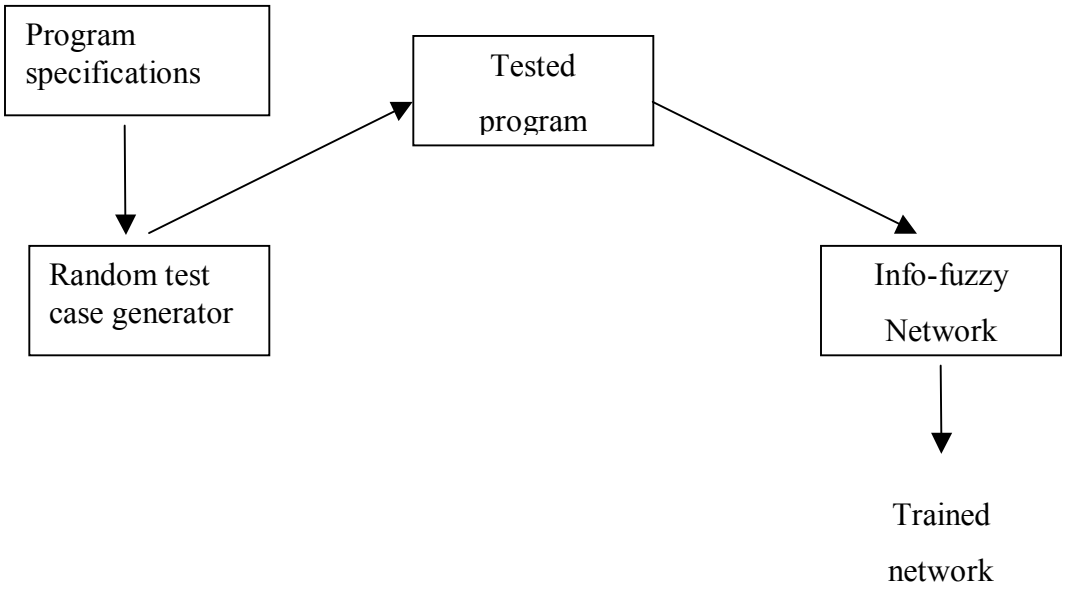

Figure 6 Network Construction Phase

The network induction algorithm is based on a pre-pruning approach i.e. when no attribute causes a statistical significant decrease in the entropy the network construction is stopped. The algorithm performs discretization of continuous input attributes on the fly unlike the neural networks where no discretization of continuous inputs is performed. The algorithm is presented in Appendix A [12].

3.3.2 Evaluation Phase

This phase is similar to the evaluation phase in the case of neural networks .The validation file generated in the case of neural network is given to the trained IFN and the faulty application and then both outputs are compared to decide whether the application output is valid or not. Like the neural networks here also reliability value is calculated. IFN gives as output two types of reliability values Rel_p and Rel_v. These values are always between [0,1] and are defined, as the degree of certainty that predicted value of the IFN is reliable. Rel_p value is calculated by the following formula [18].

$$t_k[R_i] = 2 / (1 + e^{\beta * d_{ik}}) \qquad (3)$$

Where

$t_k[R_i]$ stands for reliability value of target attribute no. i for tuple number k.This value is also termed as rel_p.

β – exponent coefficient expressing the sharpness of the reliability function. A low value of beta for example 1 makes the reliability formula behave like the sigmoidal function providing a continuous range of reliability values between 0 and 1.Higher values like 10 or 20 make it behave like a step function assigning a value of 0 if the actual value is not equal to target value. For our experiments we have kept the value of β to be 1.

d_{ik} – distance between the predicted and the actual output value in a training case k. It is calculated by the following formula.

$d_{ik} = \log P(v_{ij*}/z)/P(v_{ij}/z)$

Where

$P(v_{ij*}/z)$ = Estimated probability of the predicted value j^*, given node z.

$P(v_{ij}/z)$ = estimated probability of the actual value j, given a model z.

Rel_v value is calculated by the formula given in [10] .Also it is calculated only for the continuous target attributes and not for discrete

$$t_k[R_i] = 2 / (1 + e^{\alpha * d_{ik}}) \qquad (4)$$

$t_k[R_i]$ stands for reliability value of target attribute no. i for tuple number k.This value is also termed as rel_v.

α – As mentioned in [10] it is the exponential coefficient expressing the user perception of unexpected data. The above formula behaves like a sigmoidal function providing a gradual change in the reliability degree between 0 and 1 within the attribute range.

Higher value of α makes it behave like a step function assigning a value 0 to all value which are different than the expected value.

d_{ik} – It is the measure of the distance between the actual value and the predicted value of the output attribute. It is given by the formula

$d_{ik} = abs(t_k[Ai] - Pred_{iz}^{*})/Range_i$

Here

$t_k[Ai]$ = predicted value of target attribute A_i in tuple k.

$Pred_{iz}^{*}$ = predicted value of target attribute A_i in a tuple k here Z* refers to the terminal node.

$Range_i$ = it is the difference between the maximum and the minimum values of the target attribute A_i.

The comparison tools is then used to tell whether the application output is faulty or no. If the reliability value calculated like (3) is less than a threshold value then the output is classified as faulty else it is classified as correct. This threshold value is also experimentally determined to give the best overall results.

The next chapter explains the experiments conducted in detail.

CHAPTER 4

DESCRIPTION OF THE EXPERIMENTS

4.1 Overview

In this chapter experiments conducted to compare the two approaches i.e. neural network and the info fuzzy networks are explained. We use three tested applications to compare the two approaches. The applications used are Pay calculator, Loan approval program and a Heart disease probability calculator [29]. As mentioned in [16,17] the experiment is divided into three parts: tested application, neural network or IFN and tool comparison. For the training of Neural Network to function properly the set of inputs and the corresponding outputs are normalized to a value between 0 and 1 before training the network. This preprocessing is not needed for IFN that performs dynamic discretization of input attributes. The preprocessing of the inputs is done by the following formula.

$$\text{Normalized } X = (X_{\text{original}} - X_{\min}) / (X_{\max} - X_{\min}) \qquad (4)$$

Where X_{Min} and X_{Max} are the minimum and maximum values of input X in the whole training file. The outputs are treated in a different way. For continuous output attributes the intervals are decided based on equal frequency i.e. each interval is selected in such a way that every interval has same number of records. The number of intervals is

decided before classifying them in the intervals. For binary output it is either given a value 0 or 1, the number of intervals being 2 always in this case. For the IFN the inputs need not be preprocessed as the algorithm takes care of it. The outputs however have to be preprocessed which is done in a similar way as for the neural network.

After preprocessing, this file is used for training both the networks and hence we obtain a trained network i.e. it is able to emulate the behavior of the program under test to a desired accuracy. After the training evaluation is done in which various kinds of faults are introduced into the original application to create a faulty version of the application.

A random set of input vectors also known as validation file is then given to both the faulty version and the trained network. Using the output from both the tested system as well the neural network or IFN it is decided whether the corresponding output generated by the faulty version is correct or not.

4.2 Description Of The Applications Used For Experiments

The following section discusses about some more details of the applications used for our experiments. The pseudo code of the three systems is presented in appendix B.

4.2.1 Pay Calculator

This is a small application to calculate the pay of a given employee given his/her age, rate of pay and number of hours worked. The continuous output (Gross Pay) was discretized to 10 equal-frequency intervals.

A more detailed description of the inputs and outputs of the program is given in the table 3.1.

Table 3 Description Of Pay Calculator

Attribute name	Attribute type (Input/Output)	Attribute data type	Attribute type (Continuous/Discrete)	Attribute description
Regular hours	Input	Integer	Continuous	Represents the number of hours worked by the employee.
Age	Input	Integer	Continuous	Age of the employee..
Rate of Pay	Input	Double	Continuous	Rate of pay of the employee.
Gross Pay	Output	Double	Continuous	Gross pay of the employee.

In table IV, a sample input file is displayed, values in column 1, 2 and 3 are generated randomly according to the program specifications. The output is then calculated accordingly from the program.

Table 4 Sample Data Before Preprocessing

Age (Input)	Rate Of pay (Input)	No. of hours worked (Input)	Gross Pay (Output)
55	57	49	1851.281250
38	37	59	1355.289062
18	27	43	467.883026
57	53	64	2534.063965
44	45	43	1195.700928

As described this file is not used as a training file directly. It has to be preprocessed before being used as a training file for the neural networks and the IFN. As mentioned earlier, for IFN the inputs are not preprocessed. After preprocessing the data in table IV looks like what is shown in Table V.

Table 5 Inputs And Values After Preprocessing

Age (Input)	Rate Of pay (Input)	No. of hours worked (Input)	Gross Pay (Cont. output)
0.897436	1	0.285714	{0,0,0,1,0}
0.512821	0.333333	0.761905	{0,0,1,0,0}
0	0	0	{1,0,0,0,0}
1	0.866667	1	{0,0,0,0,1}
0.666667	0.6	0	{0,1,0,0,0}

The values are between 0 and 1 for all the input attributes. The way these values are calculated is discussed in the following paragraph.

We take column age as an example to demonstrate the preprocessing procedure. On each value in column age formula 3 is applied. It is assumed that the maximum and minimum for age column is 57 and 18 respectively. Similarly for other input attributes the values are preprocessed. The number of intervals is determined for the output attributes. In this case let us assume that the output has to be divided into 5 intervals. The next step is to calculate the interval boundaries for equal frequency. The whole set of test cases are divided so that there is equal number of cases in every interval.

After the calculation of the interval boundary the output value is placed into the correct interval using it. Rest of the values are set to 0 and the value in which the output is placed is set to 1. In the above case suppose we calculate the range size 1st interval boundary as 468 second as 1196 third as 1356 fourth as 1852 then 467 will fall in the 1st interval while 1355 will fall in 3rd interval. Note that the intervals need not be of the same size. Accordingly the output vector is constructed as {1,0,0,0,0} and {0,0,1,0,0}.

4.2.2 Loan Calculator

Depending upon certain parameters: yearly income, age, loan amount, number of years, the loan is requested. This application decides whether the loan should be approved or no. If the loan is approved it needs to be decided how much loan should be approved which is less than equal to the amount of loan applied for. The continuous output (loan amount approved) was discretized to 10 equal-frequency intervals.
A small description of all the attributes is discussed in the following section.

Table 6 Description Of Loan Calculator

Name	(Input/Output)	Data type	(Continuous/Discrete)	Description
Annual Income	Input	Double	Continuous	Represents the yearly salary of the person applying for loan.
Loan	Input	Double	Continuous	Loan amount requested for.
No of years	Input	Integer	Continuous	Time the loan is requested for.
Preferred Customer	Input	Integer	Discrete	0:Not a preferred Customer 1:Preferred Customer
Age	Input	Integer	Continuous	Age of the person.
Approved	Output	Integer	Discrete	0: Not approved 1: Approved
Loan amount approved	Output	Double	Continuous	Amount of loan approved.

4.2.3 Risk Calculator

This program calculates the chance to get coronary heart disease, based on sex, age, and systolic blood Pressure diastolic blood pressure smoking total cholesterol HDL cholesterol diabetes. The continuous output (Risk Value) was discretized to 5 equal-frequency intervals. Below is the description given for all the input and the output attributes.

Table 7 Description Of The Risk Calculator

Name	(Input/ Output)	Data type	Continuous/ Discrete	Description
Sex	Input	Integer	Discrete	0: Represents male 1: Represents female
Age	Input	Integer	Continuous	Age of the person
Systolic Blood pressure	Input	Double	Continuous	Systolic blood pressure of the person
Diasystolic Blood pressure	Input	Double	Continuous	Disystolic blood pressure of the person
Smoking	Input	Integer	Discrete	0: Represents non smoking 1: Represents smoking
Total cholesterol	Input	Double	Continuous	Total cholesterol of the person
HDL Cholesterol	Input	Double	Continuous	HDL cholesterol of the person
Diabetes	Input	Integer	Discrete	0: non diabetic 1:diabetic
Lvh	Input	Double	Continuous	Left ventricular hypertrophy
Years	Input	Integer	Continuous	No of years for which the risk is being examined
Risk	Output	Double	Continuous	Risk value, which is on the scale of 0 to 100.

Once the networks are trained the next step is the evaluation in which the performance of the IFN and the Neural network are compared. For this step, faults are introduced in the three applications and then see how well the trained networks perform as an oracle. The next section explains the type of faults injected in the three applications.

4.3 Fault Injection

To understand the nature of the faults injected please refer to the pseudo code of the three systems presented in appendix B.

As mentioned earlier after the training phase comes the testing phase in which we introduce faults in the program. After the introduction of the faults, methodology described in section 2 is followed.

Table 8 Description Of Injected Faults For Pay Calculator

Fault no.	Line no.	Original line	Injected fault	% of faulty cases	Error type
1	5	if (sex= 0)	if (sex!= 0)	25.2	Operator change
2	10	Gross_pay = reg_pay + ot_pay	Gross_pay = reg_pay – ot_pay	11.76	Operator Change
3	12	If(a > 18 && a < 70)	if(a > 18 || a < 70)	7.68	Operator change
4	16	if (gross > 272 || hw >= 30)	if (gross > 272 && hw >= 30)	30	Operator change
5	24	if (gross >= 300 && gross < 400)	if (gross >= 300 || gross < 400)	62.74	Operator change
6	26	If (gross >= 400 && gross < 500)	if (gross >= 300 || gross < 500)	65.16	Operator change
7	28	If (gross >= 500 && gross < 600)	if (gross >= 500 || gross < 600)	19.44	Operator change

Table 9 Description Of Injected Faults For Loan Calculator

Fault no	Line no	Original line	Injected fault	% of faulty cases for Output 1	% of faulty cases for Output 2	Fault type
1	3	Yearly_installment = loan/num_years	yearly_installment = loan/num_years+3	10.9	10.9	Argument change
2	8	Yearly_installment = yearly_installment + monthly_installment * 12	yearly_installment = yealry_installment + monthly_installment * 10	16.52	16.52	Argument change
3	12	if (yearly_installment <= bimonthly_income && minamt >= monthly_installment)	if (yearly_installment <bimonthly_income && minamt > monthly_installment)	28.18	28.18	Operator change
4	12	. if (yearly_installment <= bimonthly_income || minamt >=monthly_installment)	if (yearly_installment <= bimonthly_income || minamt >=monthly_installment)	21.36	21.36	Operator change
5	26	monthly_income = annual_income/12	monthly_income = annual_income/12 – 100	5.88	5.88	Argument change
6	25	bimonthly_income = (1.0/2.0) * annual_income	bimonthly_income = (1.0/4.0) * annual_income	10.3	10.3	Argument change
7	34	if(pref_customer == 0)	if(pref_customer == 1)	0%	8.86	Argument change
8	12	. if (yearly_installment <= bimonthly_income && minamt >=monthly_installment)	. if (yearly_installment > bimonthly_income && minamt <monthly_installment)	75.54	75.54	Operator change

Table 10 Description Of Injected Faults For Risk Calculator

Fault no.	Line no.	Original line	Injected fault	% of faulty cases	Error type
1	1	A = 11.1122 - 0.9119 * log(sbp) - 0.2767 * smoking - 0.718 * log(tc /hdl) - 0.5865 * lvh;	a = 11.1122 - 0.9119 * log(sbp) + 0.2767 * smoking - 0.7181 * log(tc /hdl) - 0.5865 * lvh	14.52	Operator change
2	5	If (sex!=0)	If (sex==0)	20.16	Operator change
3	8	c = (a - 5.8549) + 1.8515 * log(age/74) * log(calcArray[1]/74) - 0.3758 * diabetes;	c = (b – 5.8549) + 1.8515 * log(age/74) * log(calcArray[1]/74) - 0.3758 * diabetes;	12.78	Argument change
4	27	if (j > i)	if(j<i)	39.48	Operator change
5	17	f = exp(-0.3155 - 0.2784 * c);	f = exp(-0.3155 + 0.2784 * c);	28.5	Operator change
6	1	a = 11.1122 - 0.9119 * log(sbp) - 0.2767 * smoking - 0.718 * log(tc /hdl) -0.5865 * lvh;	a = 11.1122 - 0.9119 * log(sbp) - 0.2767 * smoking - 0.718 * log(tc /hdl)	12.54	Argument change
7	8	c = (a - 5.8549) + 1.8515 * log(age/74) * log(calcArray[1]/74) - 0.3758 * diabetes;	c = (a - 5.8549) + 1.8515 * log(age/55) * log(calcArray[1]/74) - 0.3758 * diabetes	10.8	Argument change

The next section explains the results when the trained neural networks and IFN are used as an automated oracle.

CHAPTER 5

RESULTS

5.1 Overview

To evaluate the performance of Artificial Neural Networks and IFN, series of experiments including the ROC (Receiver Operator Characteristic) analysis were performed.

The first series of experiments were conducted to investigate, the number of training records sufficient to train both the networks. The Neural Network and Info Fuzzy Networks were trained on a training set varying from 50 to 5000 records. After we get the trained networks the predictions from both networks were compared to the actual output values for a set of 5000 validation cases using the Root Mean Square Error (RMSE). The results from all the three test beds are shown below.

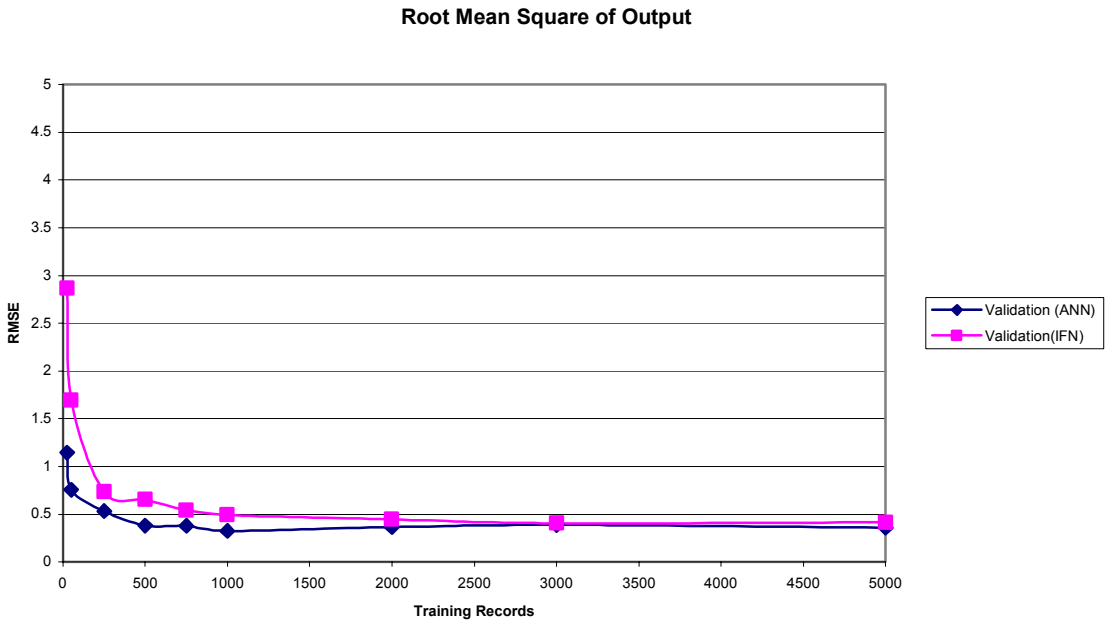

Figure 7 Root Mean Square Error Of Output For Pay Calculator

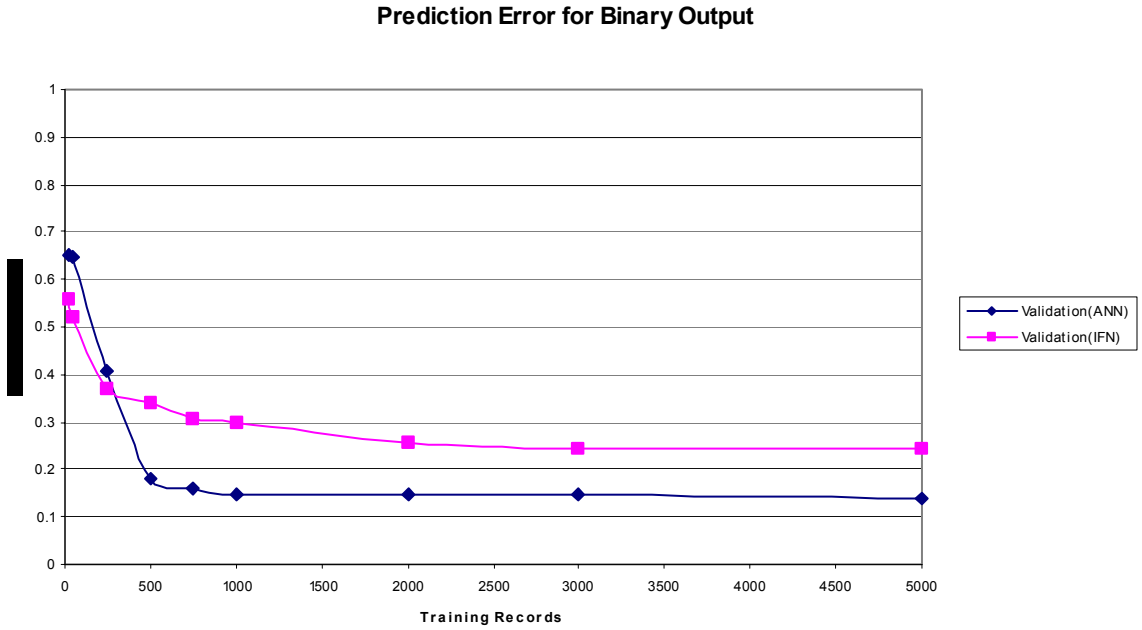

Figure 8 Root Mean Square Error Of Output For Loan Calculator (Output 1)

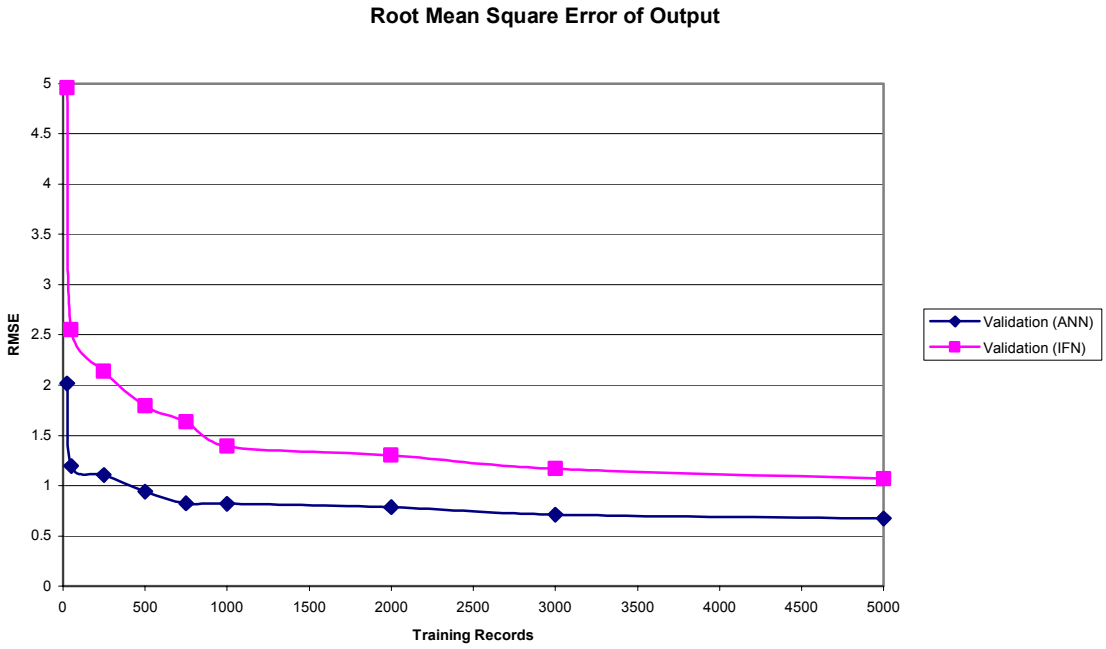

Figure 9 Root Mean Square Error Of Output For Loan Calculator (Output 2)

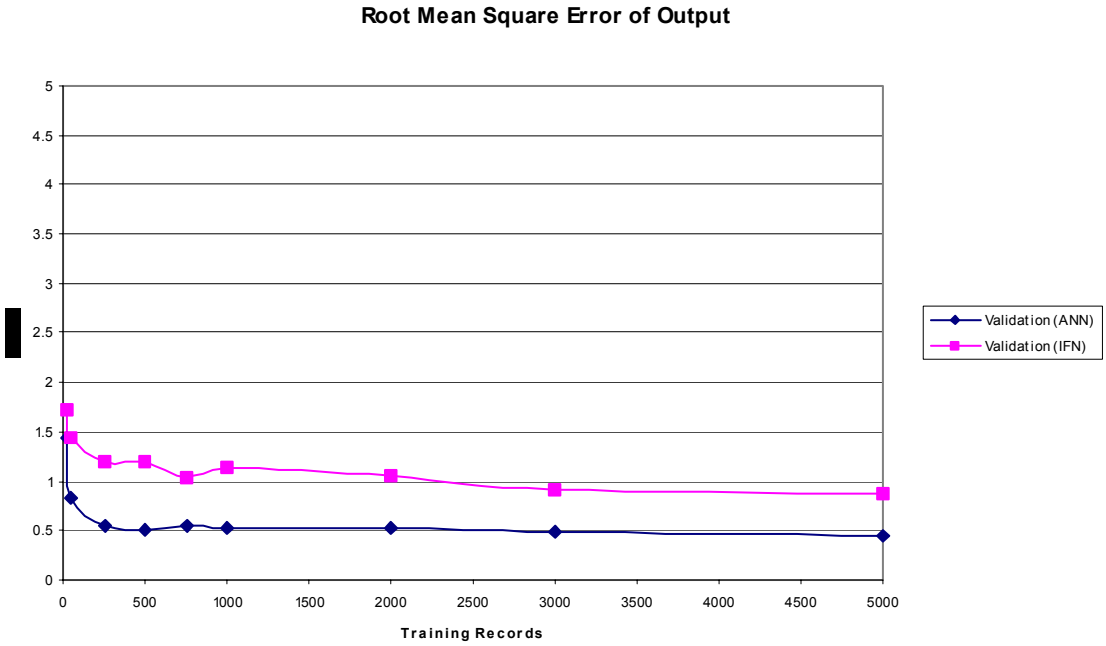

Figure 10 Root Mean Square Error Of Output For Risk Calculator

All values are in generic units

From the above figures it can be seen that there is a steep decrease in the root mean square error when the training records are between 50 and 1000 cases. After 1000 cases the relative error decreases very slowly. This suggests that 1000 cases are sufficient to train both neural networks and the info fuzzy networks. These results support the results obtained in [14] for IFN.

It is interesting to note that incase of less number of training records(50-1000) the RMSE of neural network is much lower than IFN. This suggests that neural network is a better classifier, incase of less number of training records. From Figure 8 it can be seen that the RMSE for output no.1 of loan calculator, which is a binary output is much lower than what we get for other continuous outputs for the both the methods.This is understandable, as for binary output the neural network and info fuzzy network have two choices to select from where as for continuous output there are more choices depending upon the choice of number of discretization intervals. It can also be seen that in this case IFN has lower RMSE than ANN when the number of records are lesser as opposite to the case of continuous output where ANN has much lower RMSE than IFN.

We have also plotted the results of overall validation error of the ANN and the IFN model for each faulty version in each testbed to investigate how the trained networks perform on different faulty versions. For the faulty versions the RMSE should be greater than what it is for the version without fault.

For these series of experiments both the networks were trained on 5000 records . These simulated models or in other words the trained networks are used to predict the outputs for all the faulty versions as well as correct version on a separate set of 5000 validation cases. It can be seen from figures 10,11 that the RMSE for the faulty versions is higher than the version without fault which shows that the both Neural network and Info fuzzy network are able to detect faults. We can also see that more the number of faults more is the RMSE. Figure 12 also suggests that both the methodologies, particularly IFN don't perform that well on the third test bed i.e. the risk calculator since there is considerable less difference in the RMSE of un-faulty and the faulty version.

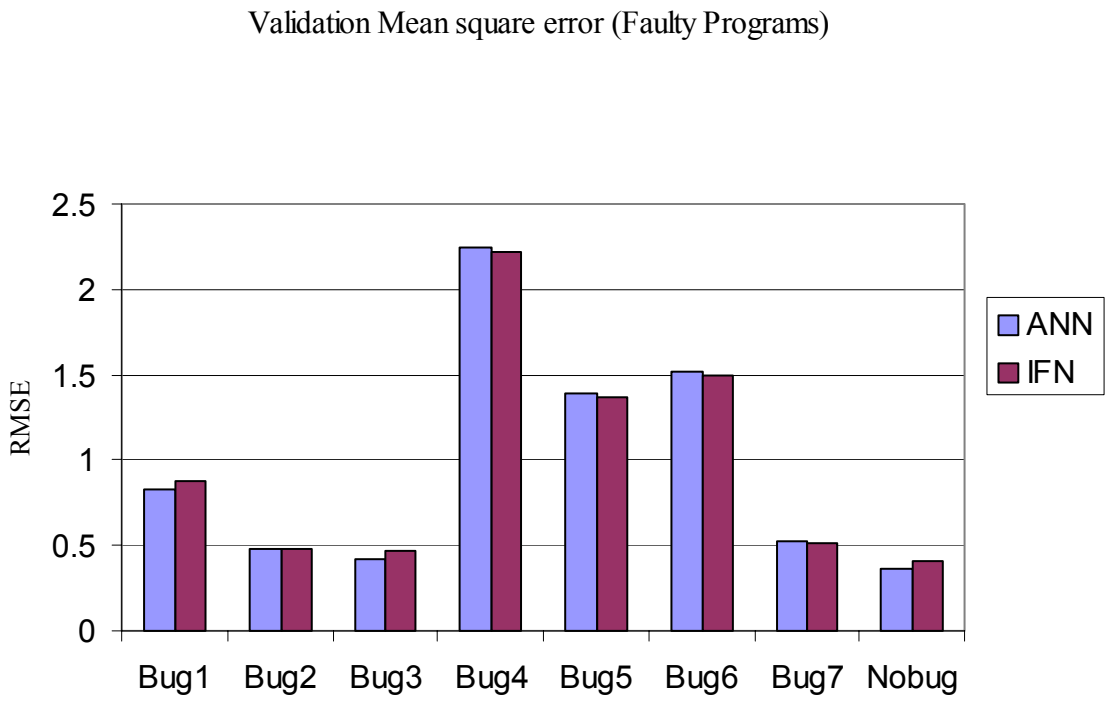

Figure 11 Root Mean Square Error For Faulty Version Of Pay Calculator

All values are in generic units

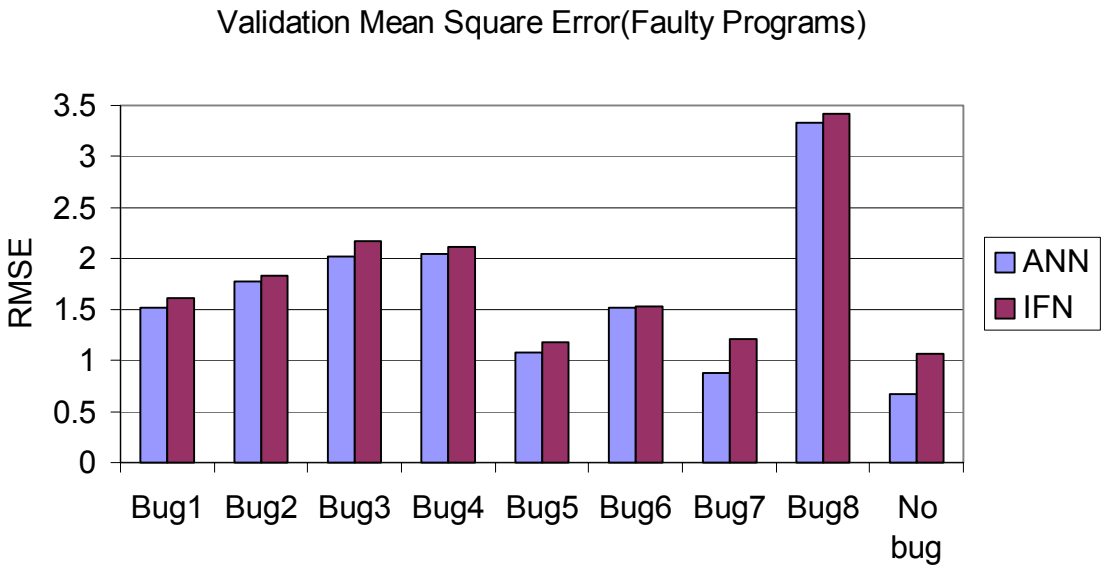

Figure 12 Root Mean Square Error For Faulty Version Of Loan Calculator

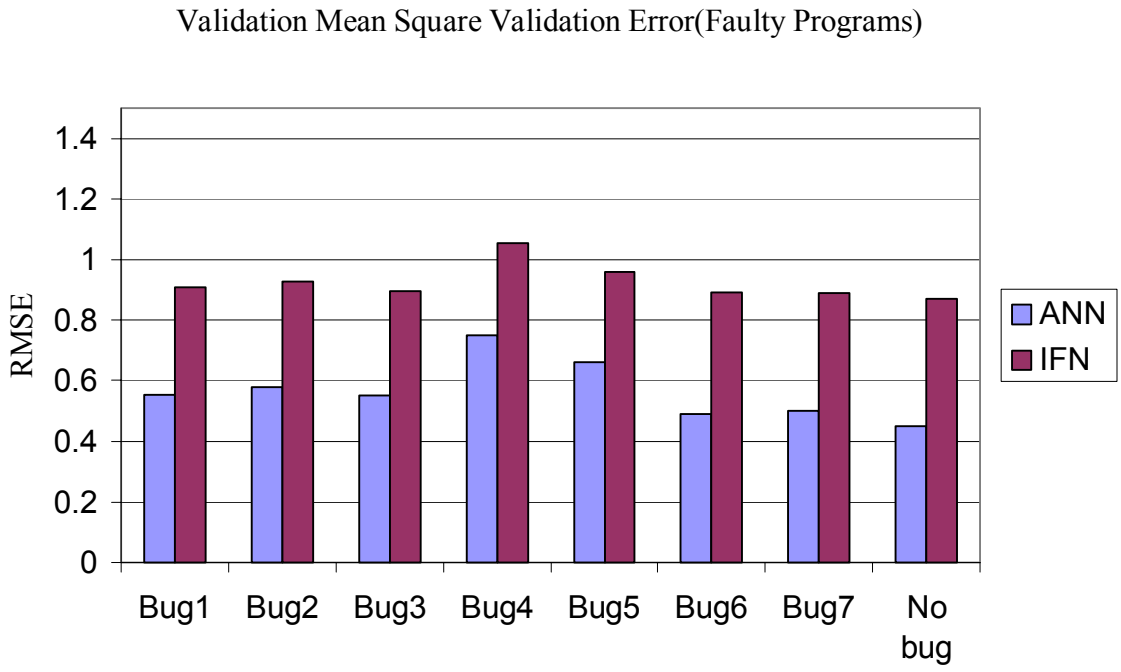

Figure 13 Root Mean Square Error For Faulty Version of Risk Calculator

All values are in generic units

Next ROC analysis has been performed to determine how well both the methods perform as an automated oracle. A ROC graph is a plot with the false positive rate on the X-axis and the true positive rate on the Y-axis. As pointed earlier positive here refers to identification of fault. For example the point (0,100) i.e. False positive (FP) rate being equal to 0% and True Positive (TP) rate equal to 100%) is the perfect classifier: it identifies all faults and doesn't identify any correct case as faulty. It is (0,100) because the false positive rate is 0%, and the true positive rate is 100%. The point (0,0) represents a classifier that predicts all cases to be fault free, while the point (100,100) corresponds to a classifier that predicts every case to be faulty. Point (100,0) is the classifier that predicts every case to be faulty. Hence for our results the more closer the points are to (0,100) the better the classifier is.

In many cases, a classifier has a threshold parameter that can be adjusted to increase TP at the cost of an increased FP or decrease FP at the cost of a decrease in TP. Each parameter setting provides a (FP, TP) pair and a series of such pairs can be used to plot an ROC curve. For our experiments we will be using different threshold values between 0 and 1 to calculate the FP and TP. The TP and FP are calculated as per table 2 and table 3 for neural networks and IFN respectively. A non-parametric classifier is represented by a single ROC point, corresponding to its (FP, TP) pair.

5.2 Results Of ROC Analysis

In the following subsections the ROC curves for all the three sample applications discussed before are presented. These ROC curves are plotted using the Rel_v and Rel_p

values for IFN and the distance value for the ANN. The Rel_v ,Rel_p and distance values are already explained in chapter 4.Each point on the ROC curve is plotted by varying the threshold values, as there is no predefined value.

5.2.1 Results For Pay Calculator

A neural network with two hidden layers consisting of 35 and 8 units respectively was used for pay calculator. The output was split into 10 equal frequency intervals. The network was trained to 88% accuracy .For the IFN also the output was split into 10 intervals. The ROC curves for pay calculator are shown below.

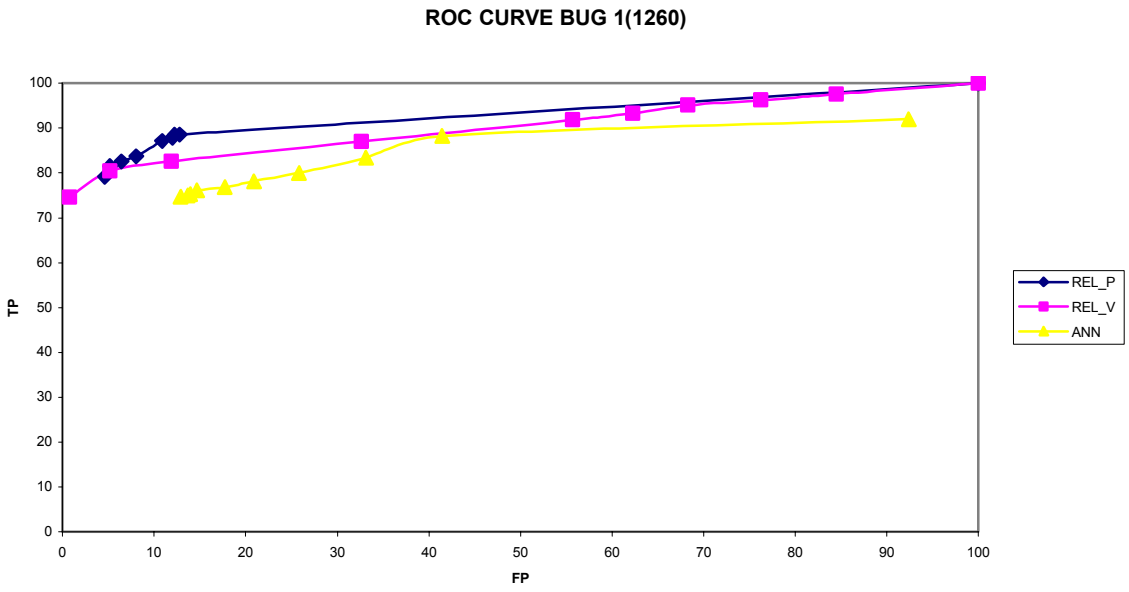

Figure 14 ROC Curve For Induced Fault No.1 For Pay Calculator

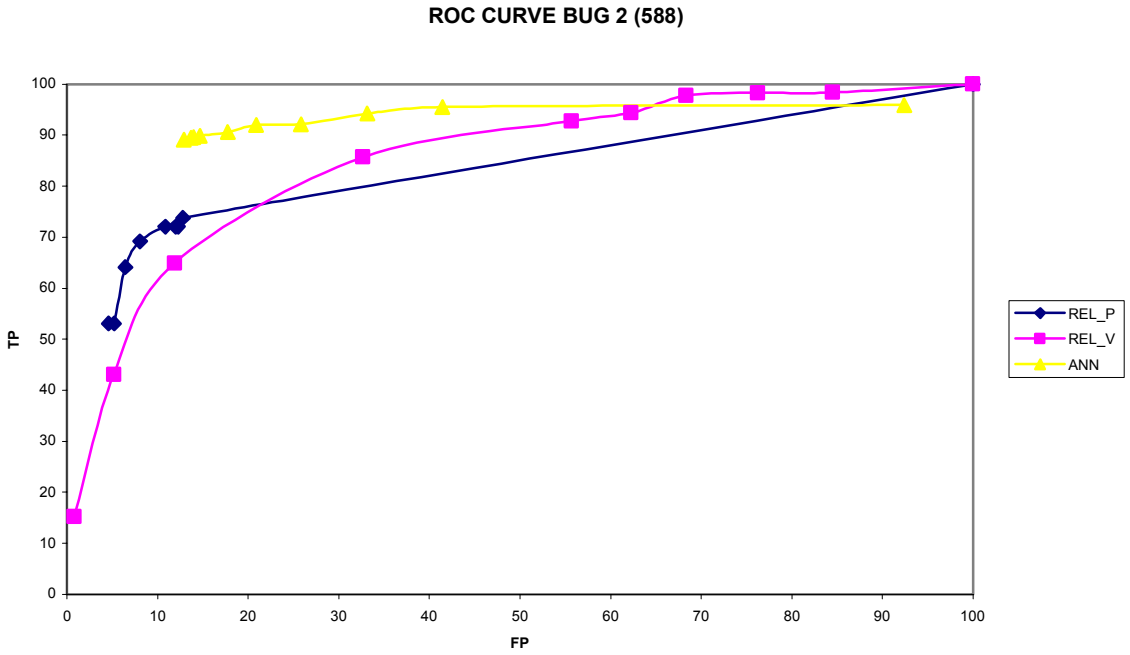

Figure 15 ROC Curve For Induced Fault No.2 For Pay Calculator

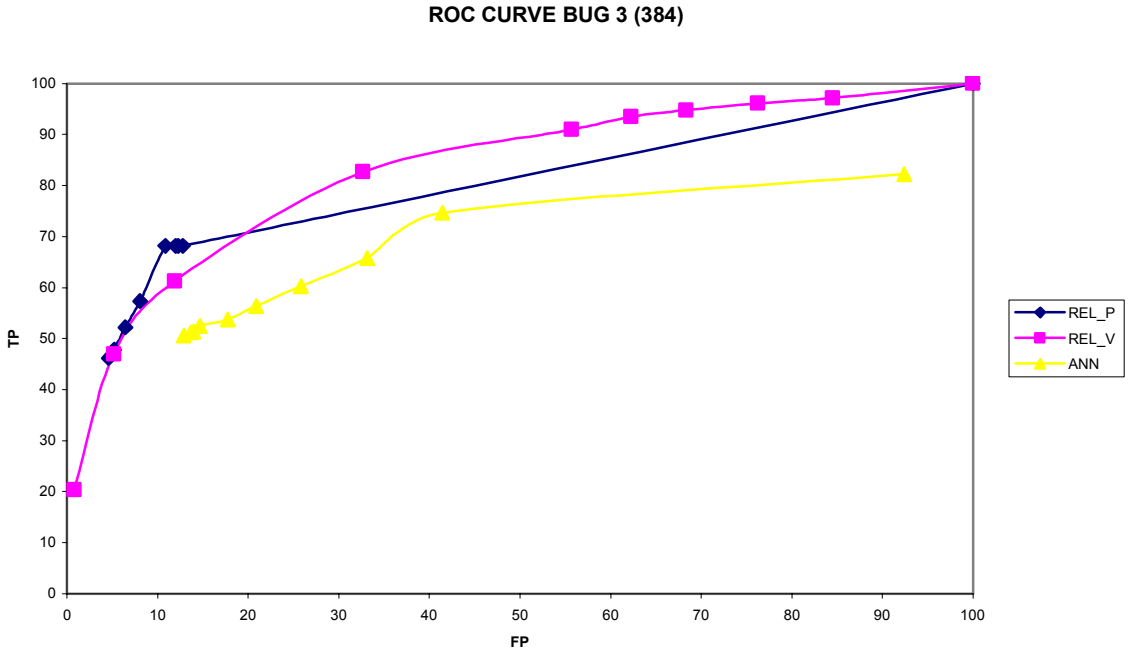

Figure 16 ROC Curve For Induced Fault No.3 For Pay Calculator

TP and FP are calculated in % units

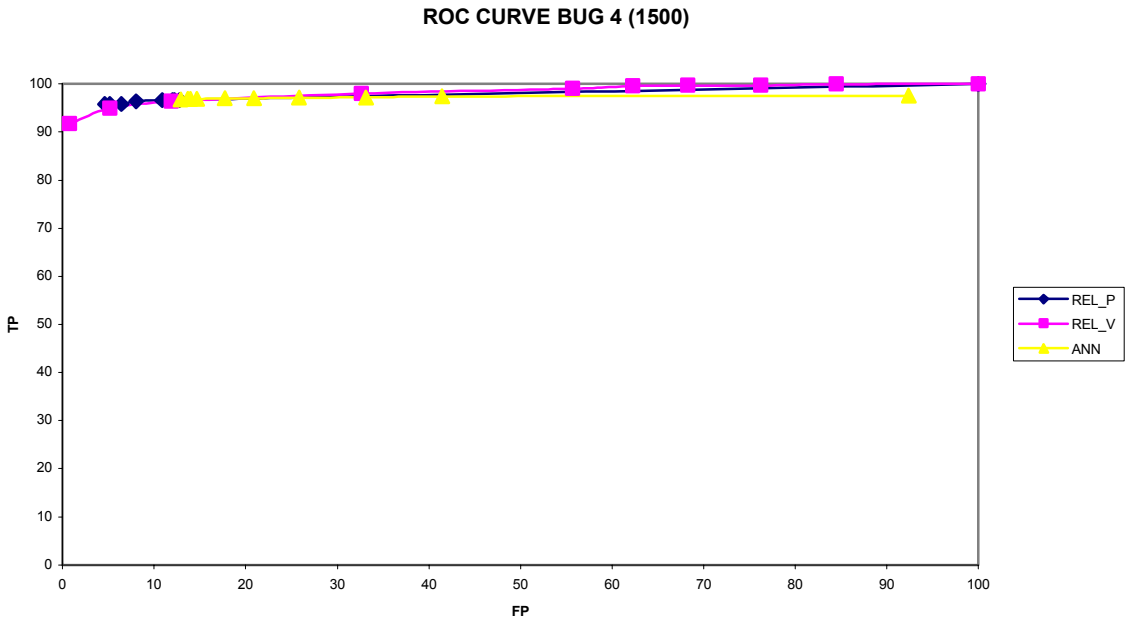

Figure 17 ROC Curve For Induced Fault No.4 For Pay Calculator

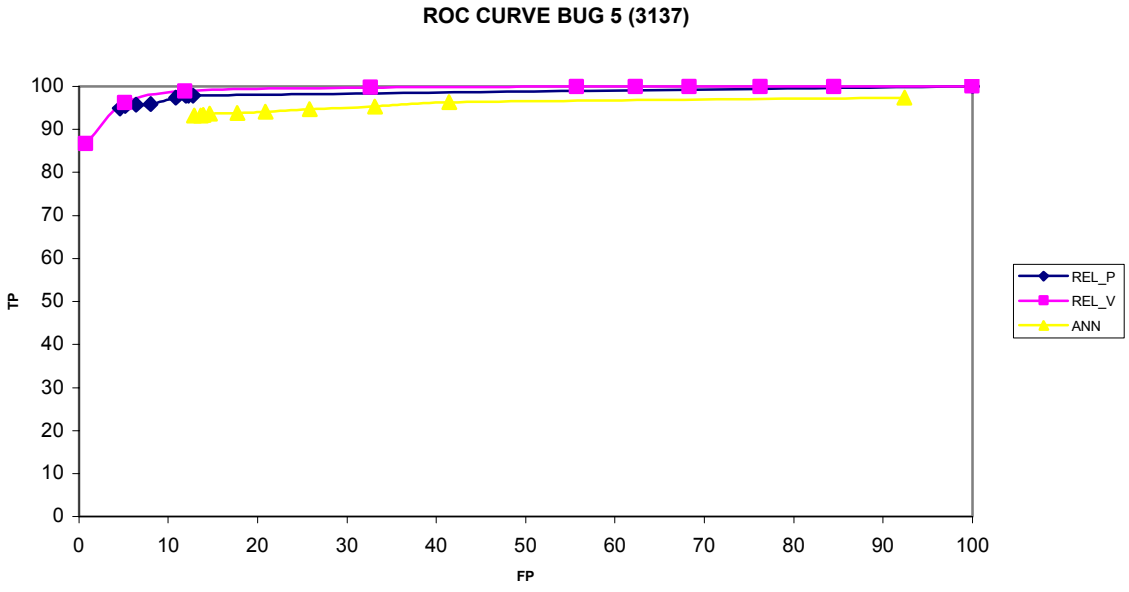

Figure 18 ROC Curve For Induced Fault No.5 For Pay Calculator

TP and FP are calculated in % units

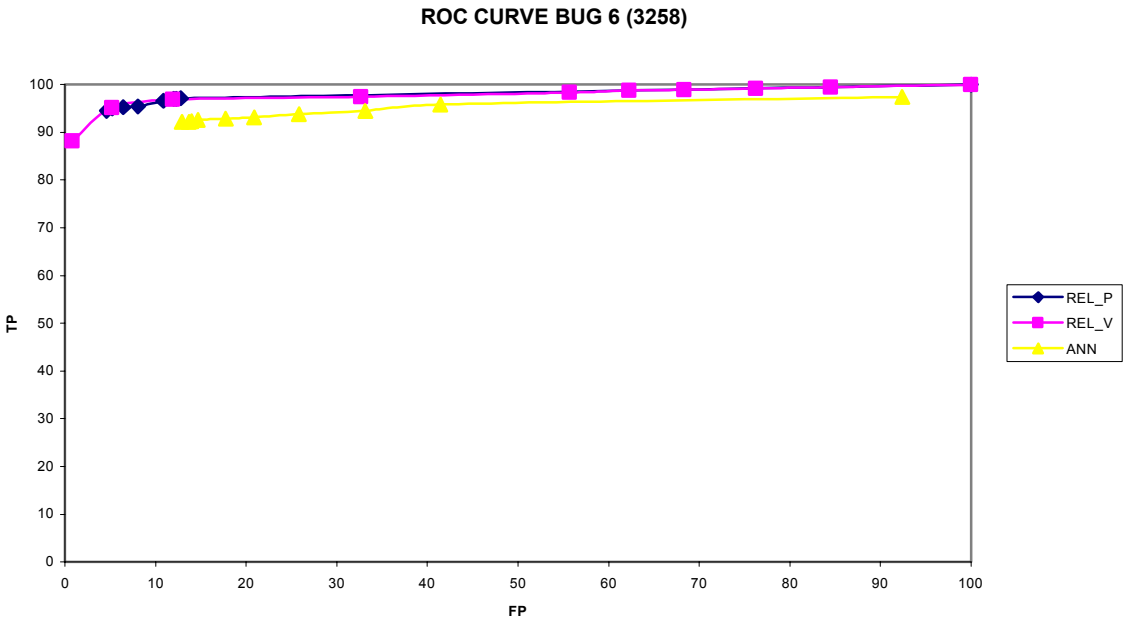

Figure 19 ROC Curve For Induced Fault No.6 For Pay Calculator

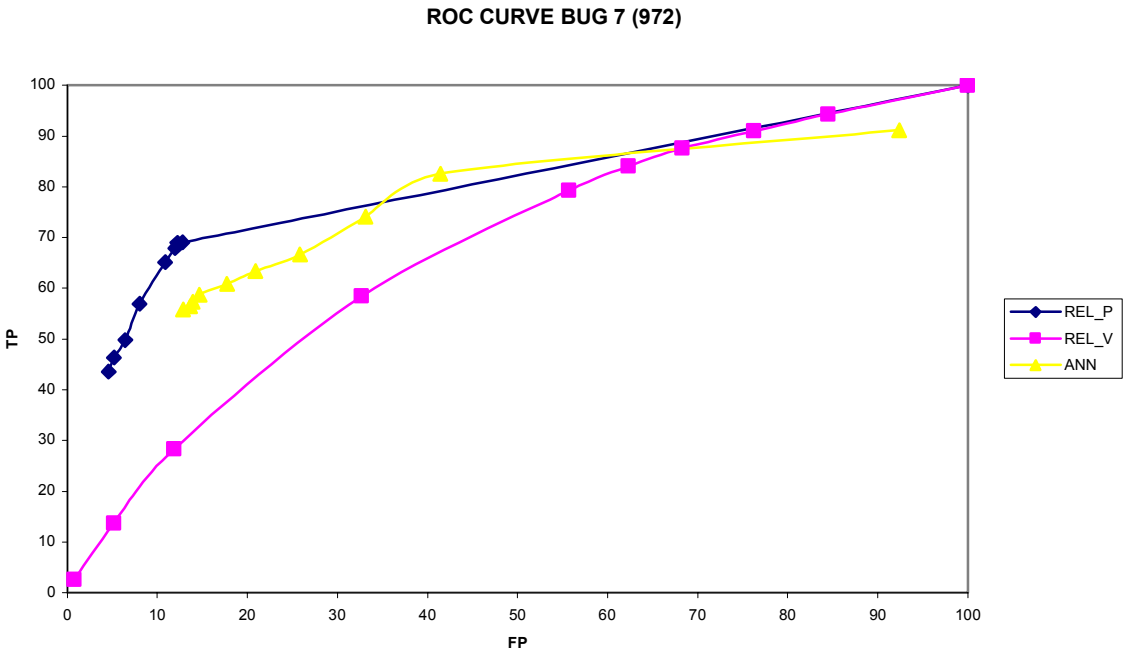

Figure 20 ROC Curve For Induced Fault No.7 For Pay Calculator

TP and FP are calculated in % units

The above figures show that for relatively low number of faults for example figures 15, 16 and 20 none of the methodologies work fine.There is hardly anything to choose between the two methods. But we can see that for relatively high number of faults IFN performs better than ANN. For example in figure 18 we are able to achieve (FP, TP) equal to (5.16, 96.2) with IFN as compared to (12.88, 93.14) for the ANN. In figure 19 also we can see that with IFN a (FP, TP) of (5.16, 95.18) is achieved and with ANN (12.88,92.17). Hence we conclude that for this application, for relatively high number of faults IFN is better according to the ROC . Nothing can be concluded when the number of faults is low.

5.2.2 Results For Loan Calculator

For the Loan calculator a neural network with two hidden layers consisting of 40 ,8 hidden units respectively was used for output 2 which is a continuous output. The output was again split into 10 equal frequency intervals. The network was trained to 87% accuracy .For the IFN also the output was also split into 10 intervals. For output 1 which is a binary output there are two output units. The ANN was trained to 99% accuracy and neural network with single hidden layer consisting of 32 hidden units was used.

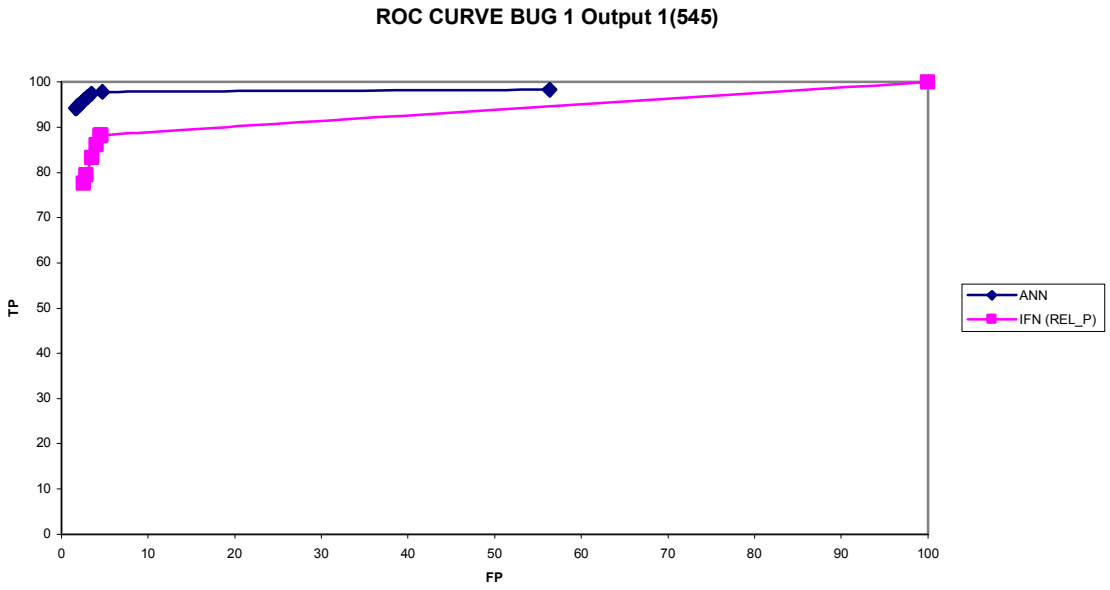

Figure 21 ROC Curve For Induced Fault No.1 For Loan Calculator (Output 1)

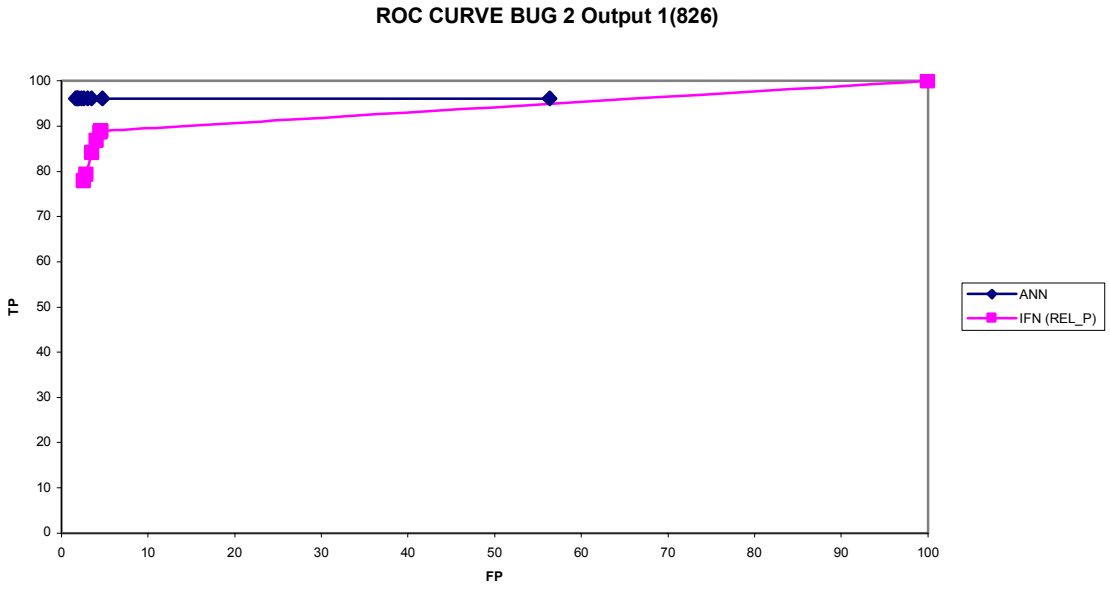

Figure 22 ROC Curve For Induced Fault No.2 For Loan Calculator (Output 1)

TP and FP are calculated in % units

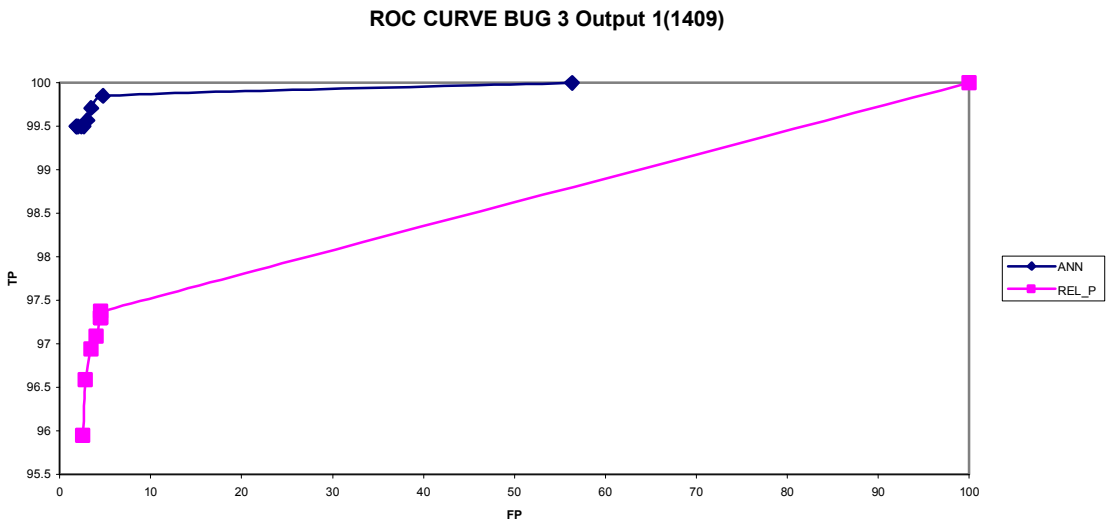

Figure 23 ROC Curve For Induced Fault No.3 For Loan Calculator (Output 1)

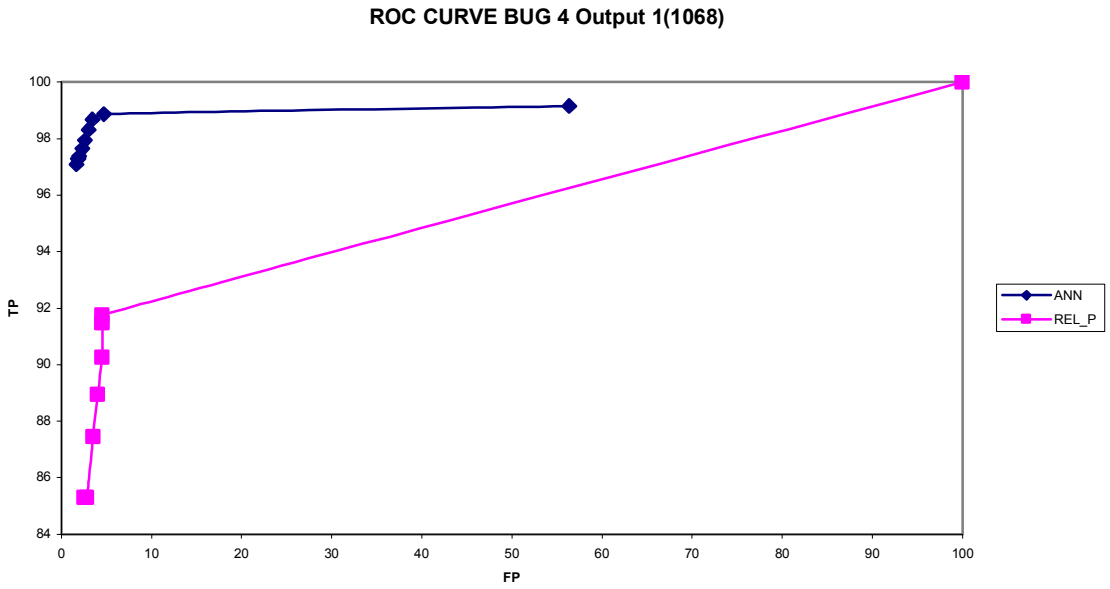

Figure 24 ROC Curve For Induced Fault No.4 For Loan Calculator (Output 1)

TP and FP are calculated in % units

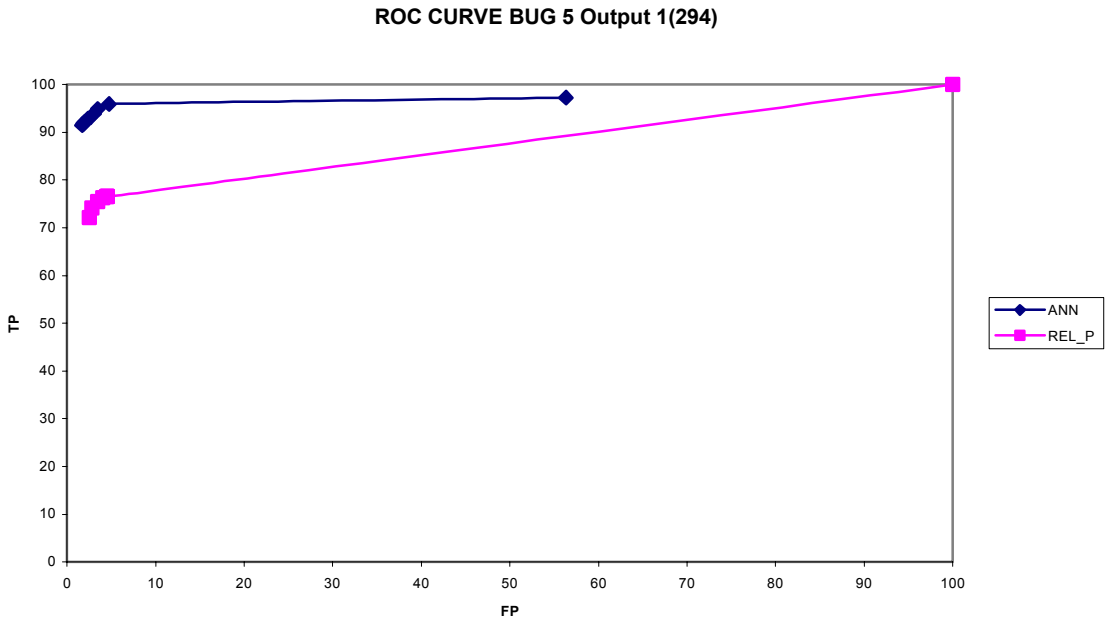

Figure 25 ROC Curve For Induced Fault No.5 For Loan Calculator (Output 1)

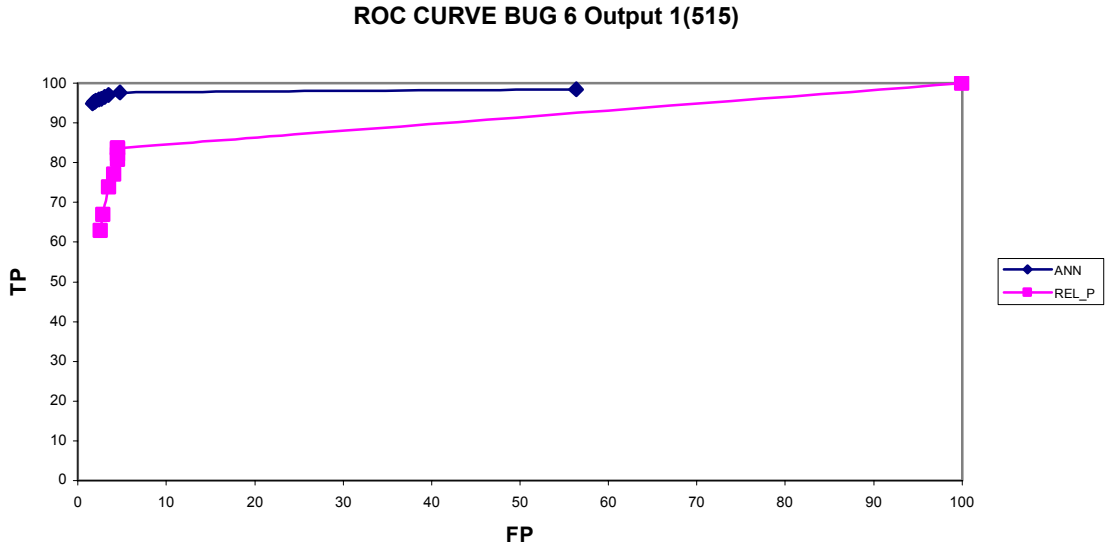

Figure 26 ROC Curve For Induced Fault No.6 For Loan Calculator (Output 1)

TP and FP are calculated in % units

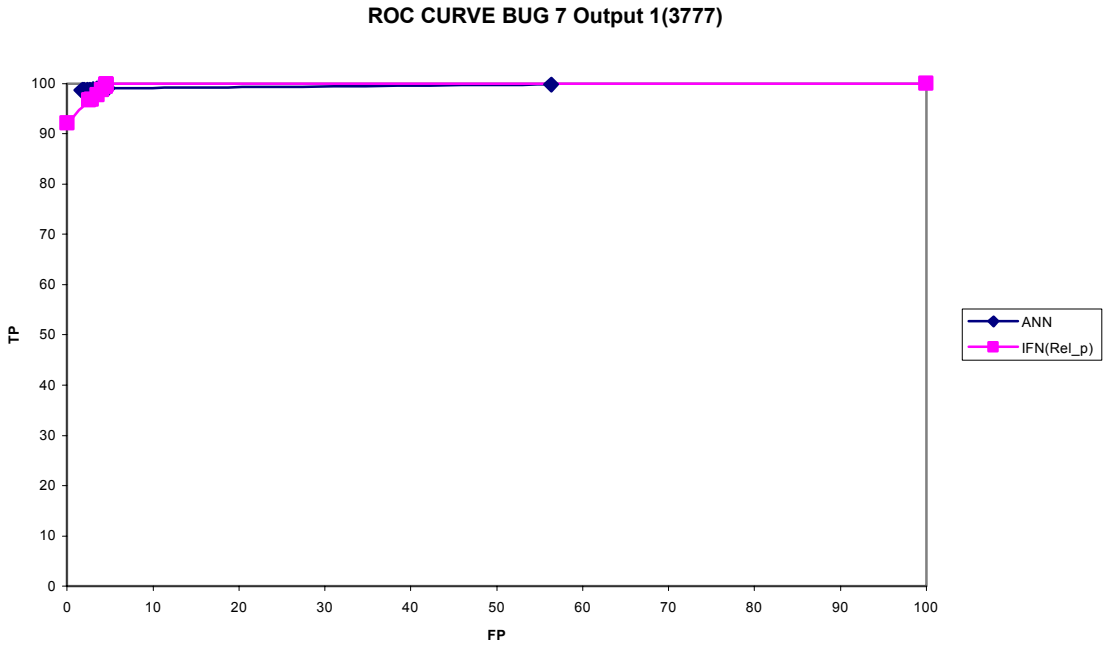

Figure 27 ROC Curve For Induced Fault No.7 For Loan Calculator (Output 1)

From the above figures it can be concluded that that for binary output ANN performs better than IFN. Similar to the case of previous testbed the performance of the methods improve with the number of faulty cases the same behavior can be seen here also. The ROC curves of the output no.2 are presented below.

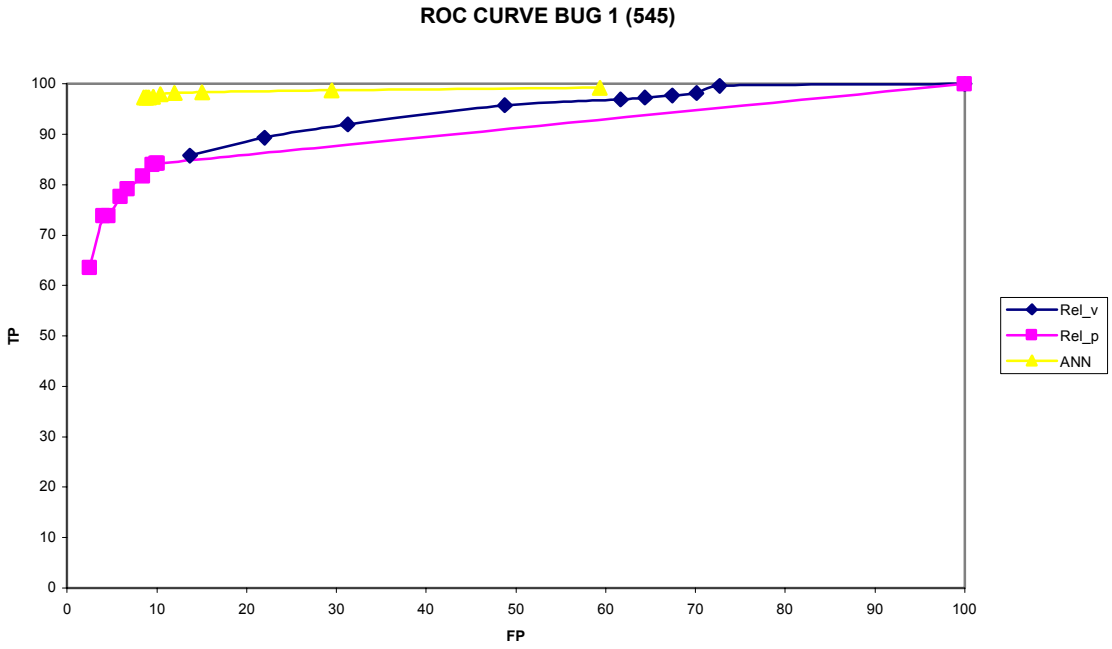

Figure 28 ROC Curve For Induced Fault No.1 For Loan Calculator (Output 2)

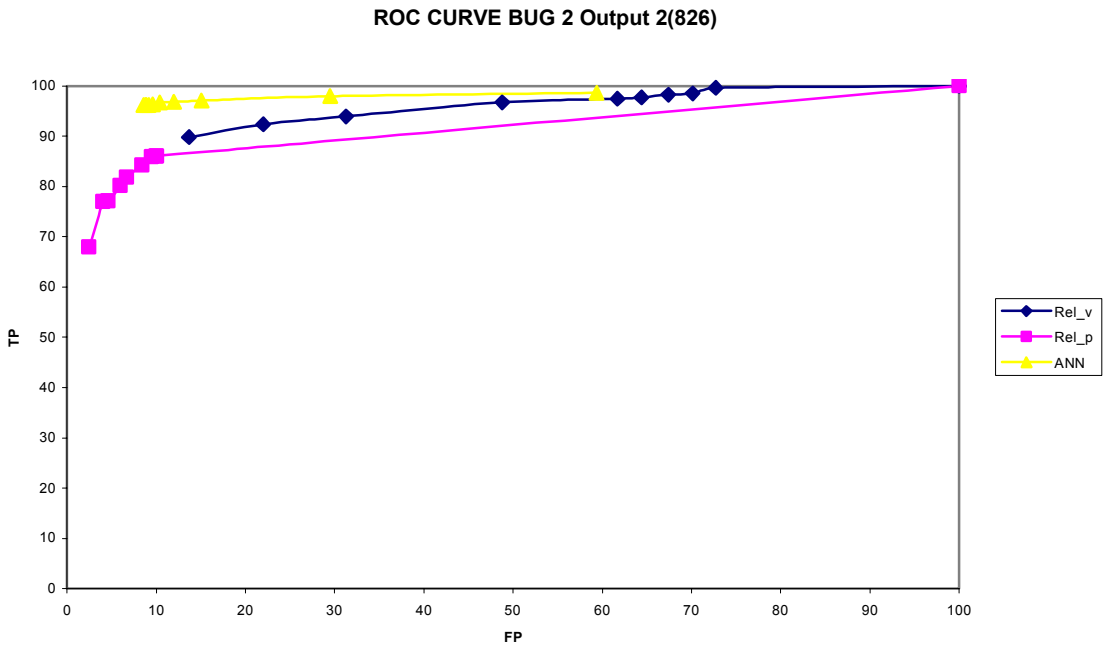

Figure 29 ROC Curve For Induced Fault No.2 For Loan Calculator (Output 2)

TP and FP are calculated in % units

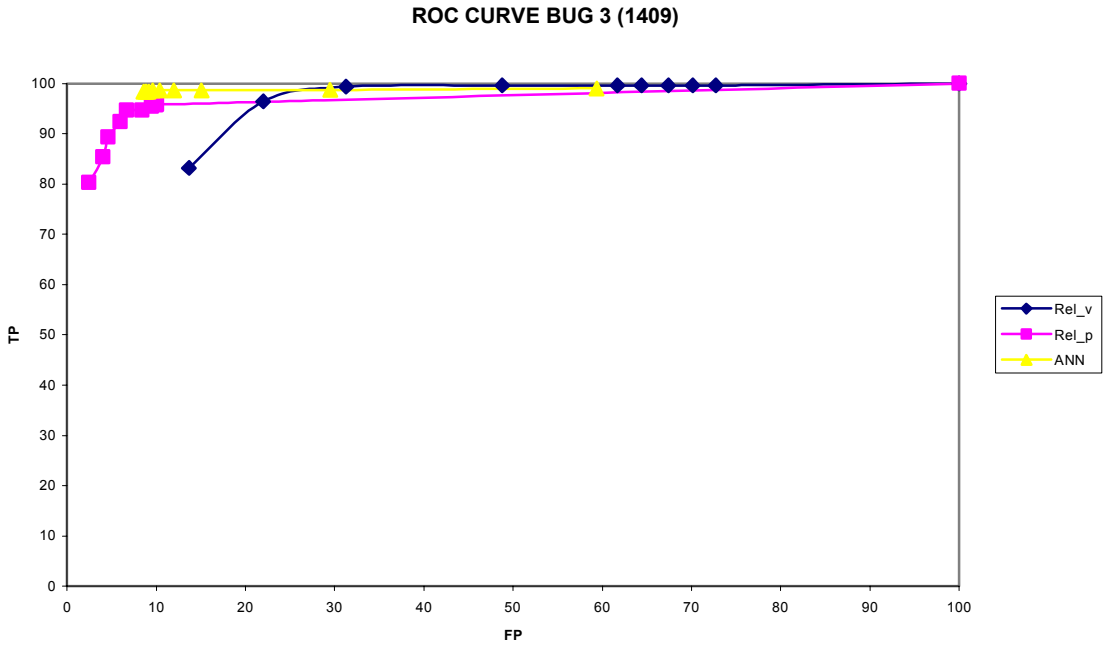

Figure 30 ROC Curve For Induced Fault No.3 For Loan Calculator (Output 2)

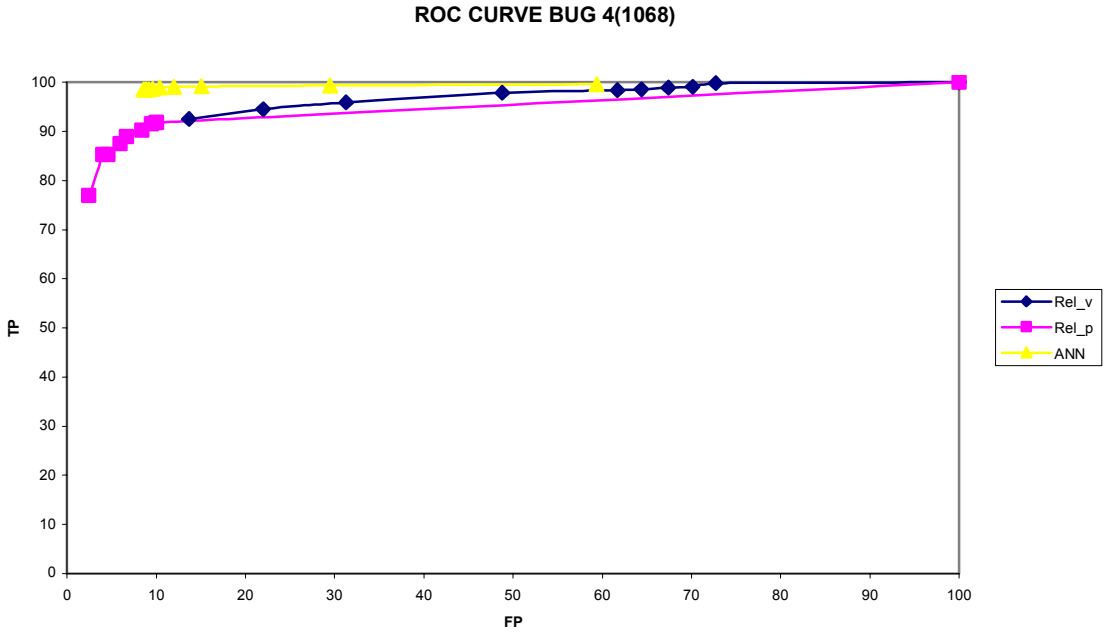

Figure 31 ROC Curve For Induced Fault No.4 For Loan Calculator (Output 2)

TP and FP are calculated in % units

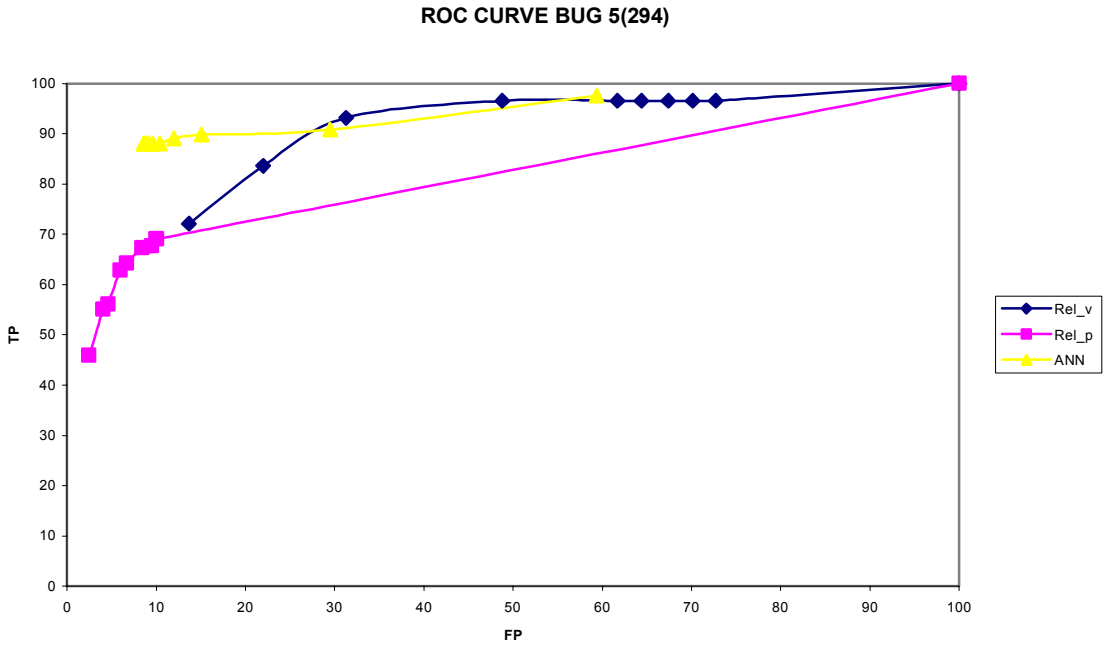

Figure 32 ROC Curve For Induced Fault No.5 For Loan Calculator (Output 2)

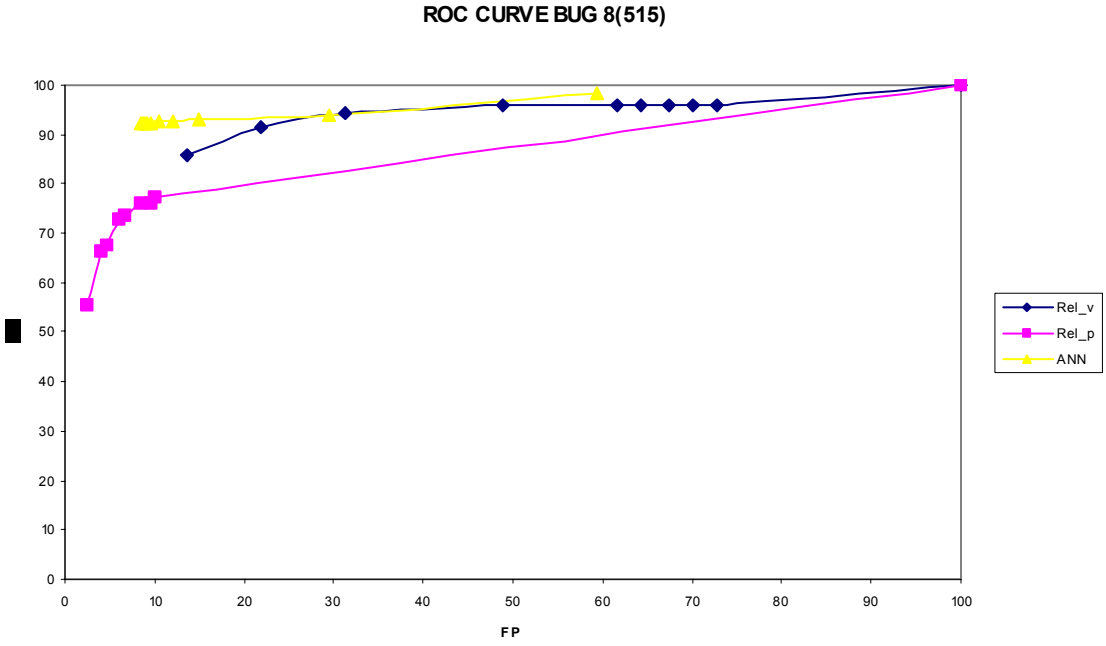

Figure 33 ROC Curve For Induced Fault No.6 For Loan Calculator (Output 2)

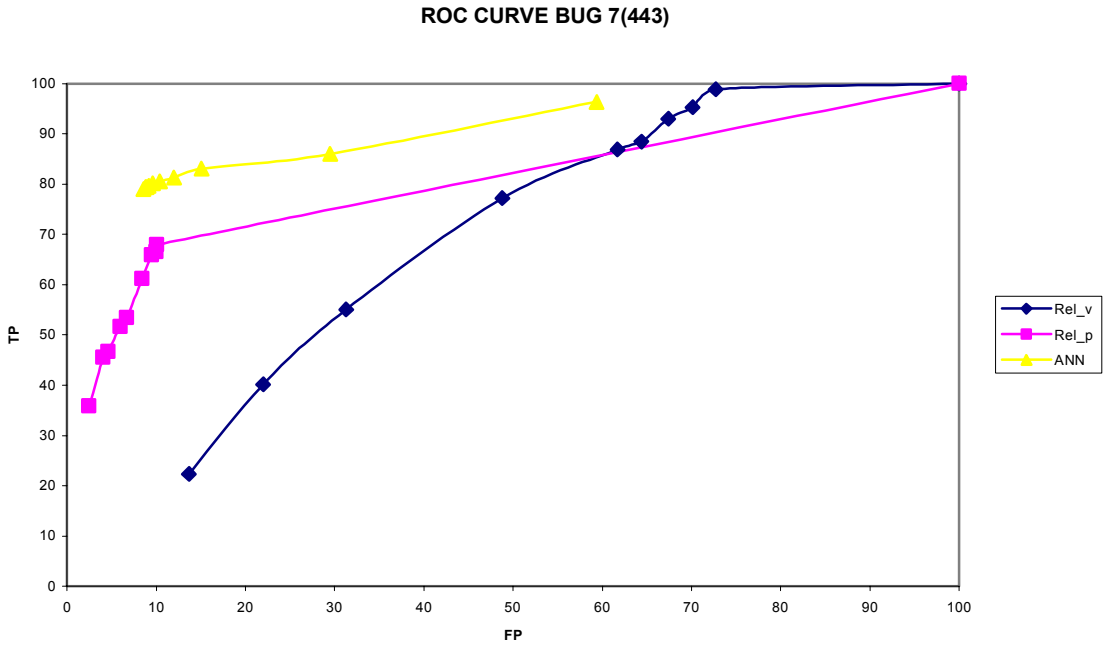

Figure 34 ROC Curve For Induced Fault No.7 For Loan Calculator (Output 2)

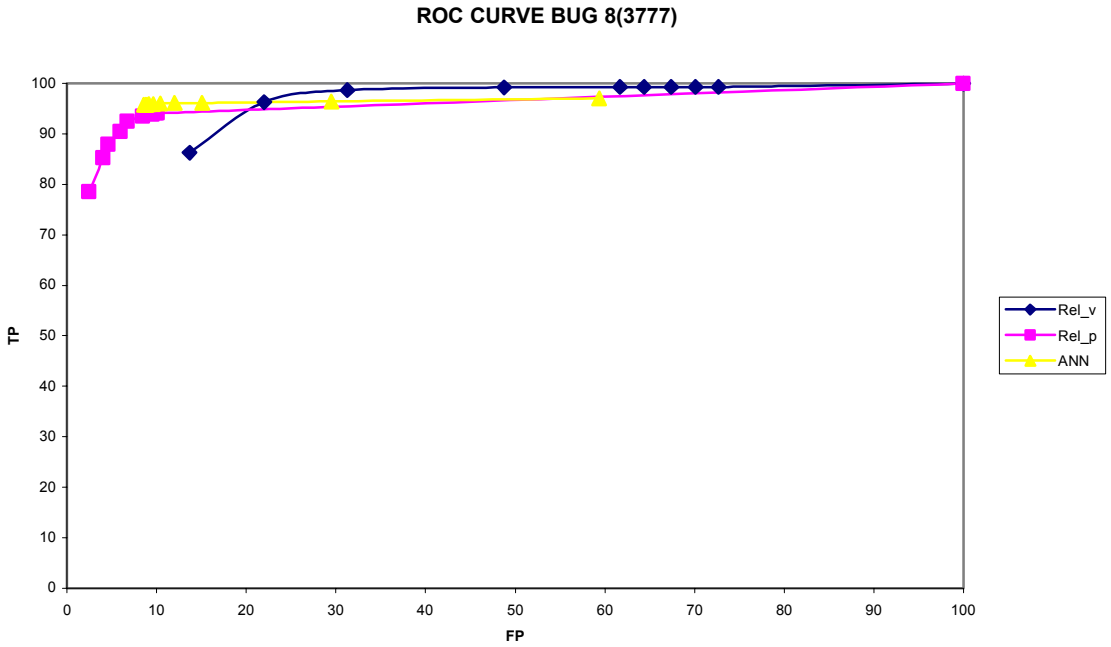

Figure 35 ROC Curve For Induced Fault No.8 For Loan Calculator (Output 2)

TP and FP are calculated in % units

From the above figures we observe the same kind of behavior as was seen for the Pay calculator. Comparatively for lower number of faults none of the methodology seems to perform good but for higher number (for e.g. figures 30,31,35) of faulty records we get better results i.e. high true positive and a low false positive for both methodologies. In this case, however ANN just outperforms IFN in case of higher number of faults as opposite to the previous case.

5.2.3 Results For Risk Calculator

For the Risk calculator a neural network with two hidden layers consisting of 47 and 9 hidden units respectively was used. The output was split into 5 equal frequency intervals. The network was trained to 92% accuracy .For the IFN also the output was split into 5 intervals. Figure 34 to 43 represents the ROC curves obtained for different faulty versions of risk calculator.

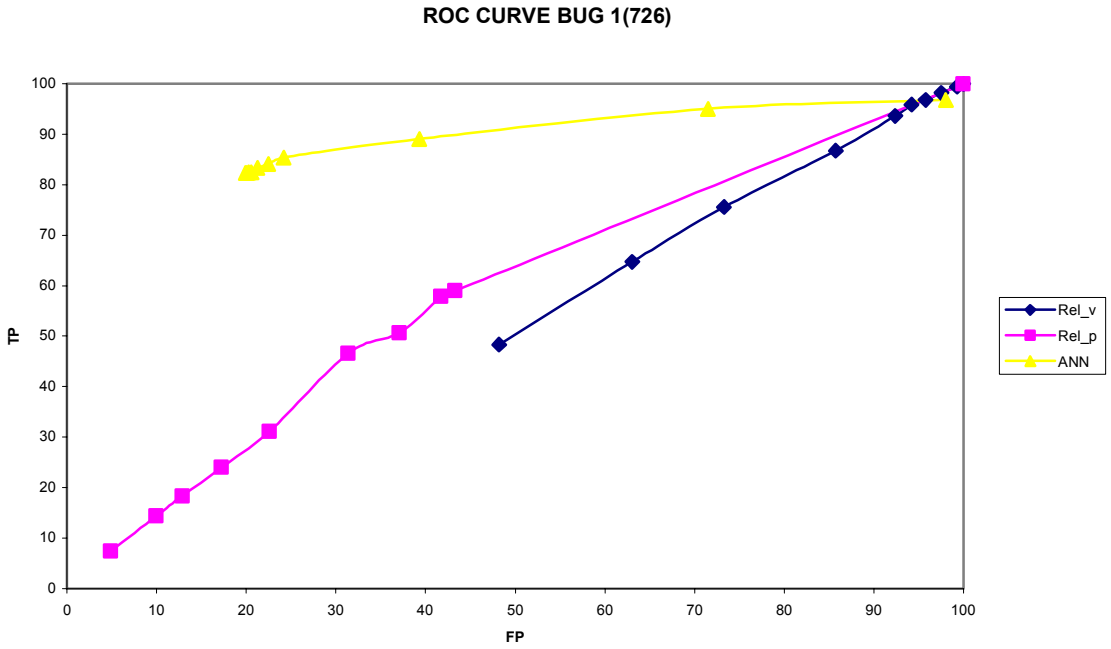

Figure 36 ROC Curve For Induced Fault No.1 For Risk Calculator

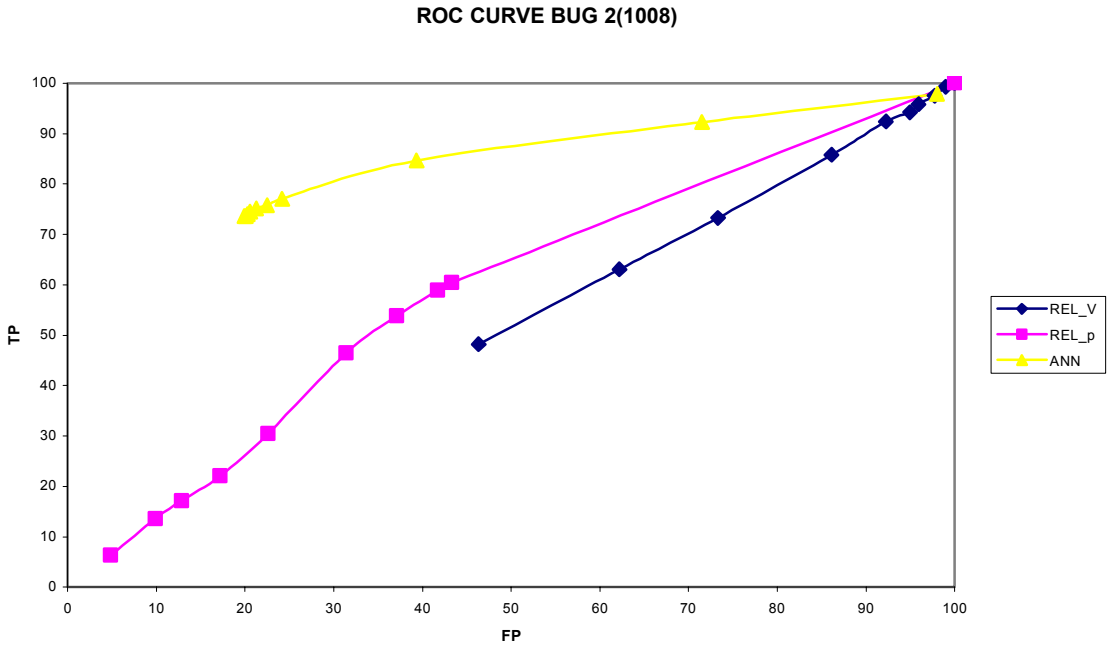

Figure 37 ROC Curve For Induced Fault No.2 For Risk Calculator

TP and FP are calculated in % units

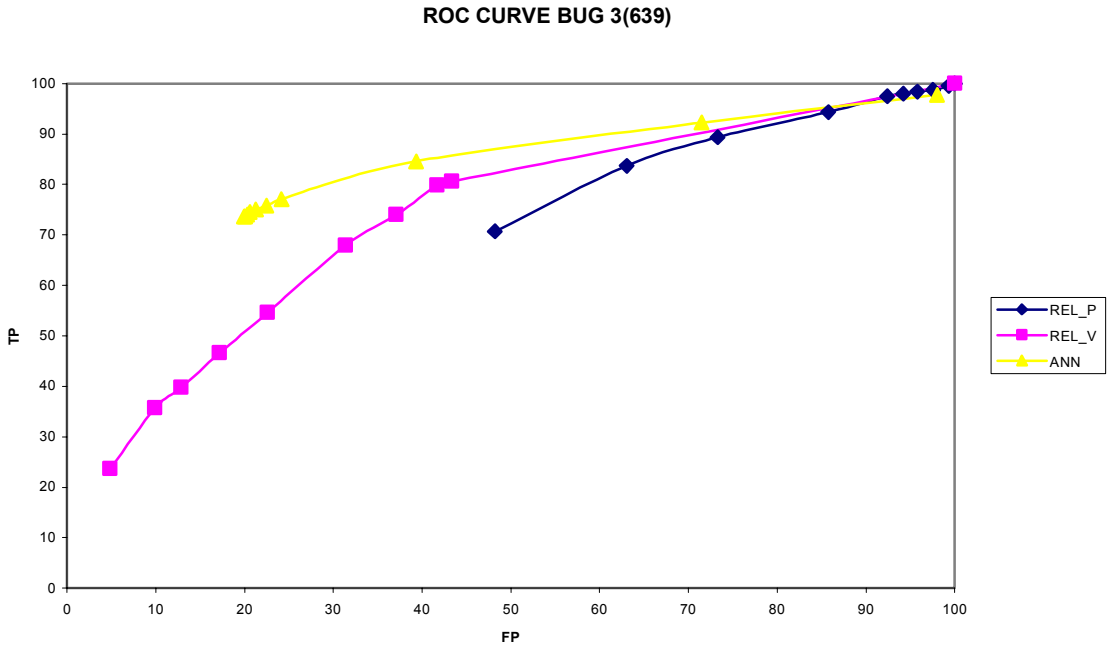

Figure 38 ROC Curve For Induced Fault No.3 For Risk Calculator

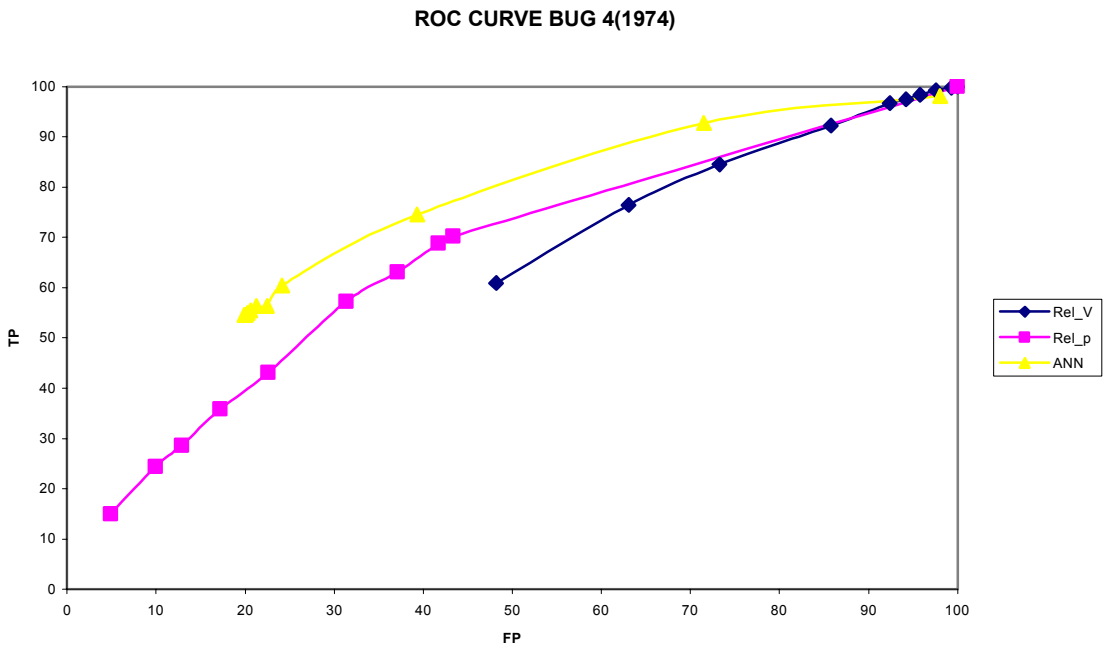

Figure 39 ROC Curve For Induced Fault No.4 For Risk Calculator

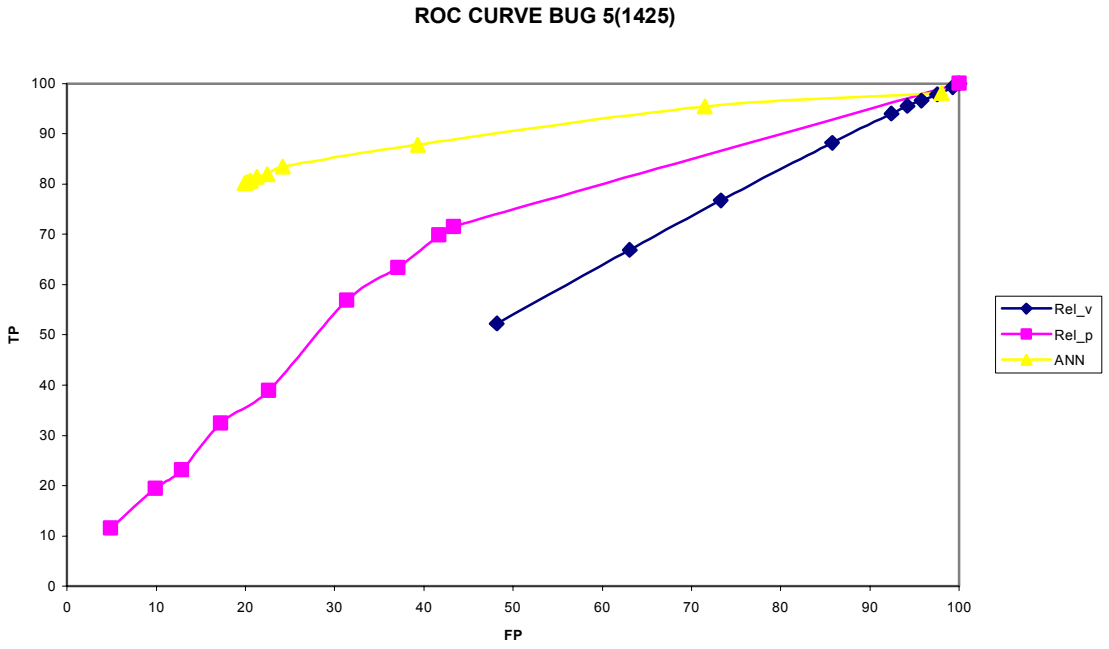

Figure 40 ROC Curve For Induced Fault No.5 For Risk Calculator

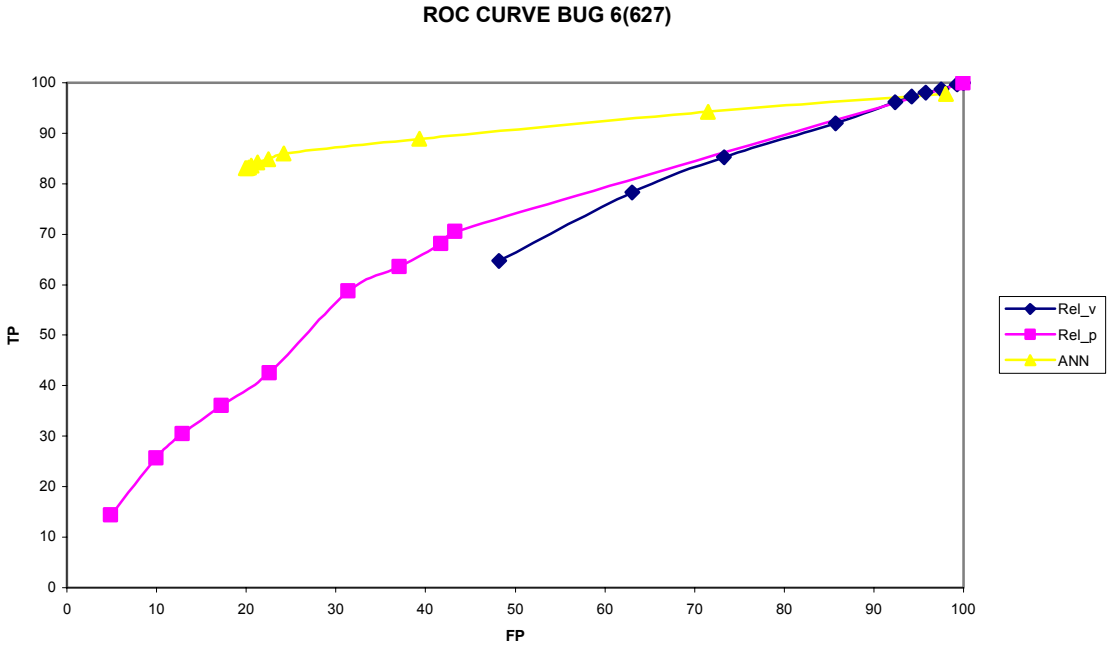

Figure 41 ROC Curve For Induced Fault No.6 For Risk Calculator

TP and FP are calculated in % units

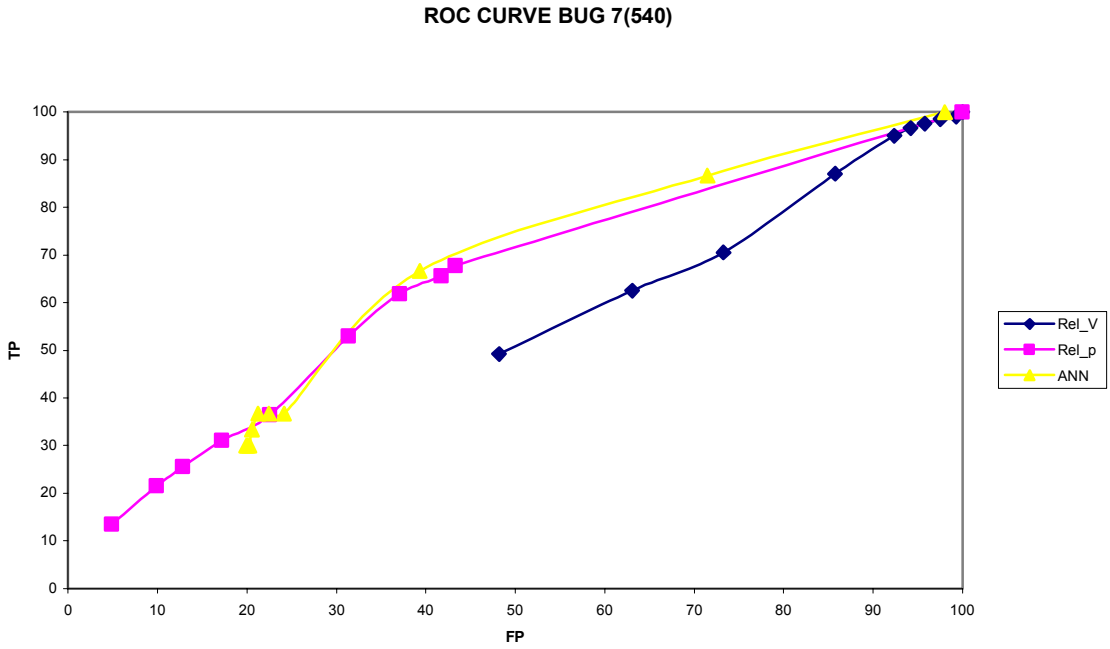

Figure 42 ROC Curve For Induced Fault No.7 For Risk Calculator

For risk calculator ANN seems to be a better approach than IFN although performance wise none of the methodologies look effective.

From the above results we conclude that according to the ROC Analysis there is no exact advantage of one method over the other. In the next section we present a comparison of the running times of both the methods.

5.3 Execution Times

To compare the execution times of both the methods we calculated the time it takes for training the networks and the evaluating a testbed. For all the three testbeds the time it takes to train and evaluate a testbed for both IFN and ANN was calculated. Twenty different runs were taken for each testbed. The times shown in table are

calculated on the same machine in the same environments. The networks were trained on a set of 5000 training cases.

Table 11 Execution Times Of ANN And IFN Applied As An Automated Oracle On Pay Calculator (Seconds)

Pay Calculator		
	ANN	IFN
1	89.12	1.95
2	58.2	1.67
3	84.1	1.11
4	72.98	2.56
5	46.92	1.44
6	45.2	1.55
7	93.1	1.44
8	51.78	2.86
9	38.4	1.37
10	40.45	1.2
11	18.65	2.11
12	94.26	1.13
13	25.67	1
14	60	1.03
15	47.2	1.02
16	34.1	1.86
17	41	0.98
18	64.2	1.05
19	102.9	1.1
20	109	1.32
Mean Value	60.86	1.4875
Standard Deviation	26.66	.5368

Table 12 Execution Times Of ANN And IFN Applied As An Automated Oracle On Loan Calculator (Seconds)

Loan Calculator		
	ANN	IFN
1	105.8	8.42
2	106.7	9.72
3	96.98	7.08
4	107.45	9.64
5	88.48	9.94
6	105.4	10.11
7	105.22	10.11
8	103.8	23.53
9	109.68	7.48
10	148.3	10.25
11	139.4	9.06
12	206	10.22
13	208	7.92
14	180.1	9.97
15	206.7	9.67
16	200.5	9.89
17	183.1	10.34
18	76.9	14.24
19	103	8.45
20	105.1	9.38
Mean Value	134.33	10.271
Standard Deviation	45.32	3.44

Table 13 Execution Times Of ANN And IFN Applied As An Automated Oracle On Risk Calculator (Seconds)

Risk Calculator		
	ANN	IFN
1	32	4.91
2	126.1	2.78
3	124	4.22
4	125.3	4.23
5	125.8	4.05
6	119.7	4.22
7	123	4.36
8	122.1	6.78
9	119.98	5.19
10	117.2	3.63
11	118.2	16.78
12	118.45	32.94
13	117.2	3.63
14	117.2	3.58
15	117.2	3.94
16	117.2	4.2
17	117.2	18.73
18	117.2	4.78
19	117.2	16.53
20	117.2	5.34
Mean Value	115.4715	7.741
Standard Deviation	19.19	7.64

We can see that although the standard deviation for execution times of the neural network vary but still execution time for IFN is much lesser than ANN. From the above values it is clear that IFN is much faster than ANN.

5.4 Summary And Conclusions

In this thesis we have compared IFN and ANN approaches. The study will help to choose the appropriate method for purpose of an automated oracle under the given parameters like training records, time constraints, required accuracy etc. The results presented indicate that the neural networks perform as a better classifier than the IFN for continuously valued outputs in terms of the Root Mean Square Error (RMSE). Especially when the number of training records is lower the induced model for neural network performs much better than IFN. The main advantage of IFN over ANN is its speed. The results show that the IFN is much faster than ANN. According to the results indicated by the ROC analysis there is no absolute advantage of one technique over the other. Both techniques become efficient only when the percentage of faulty cases is relatively large.

CHAPTER 6

FUTURE WORK

One of the interesting things to do in the future is to determine what characteristics of the software or the test-bed make it suitable to be tested by Neural network or IFN. Specifically, one can compare the performance of both methods for applications with discretely valued outputs, where the error is measured by misclassification rate rather than RMSE. It will also be interesting to apply other neural network training algorithms like Rprop, Quasi Newton etc for the neural network training instead of the traditional backpropogation.

REFERENCES

[1] A.von Mayrhausser, C. Anderson, R. Mraz, "Using a neural network to predict test case effectiveness","Proceedings 1995 IEEE, Aerospace Applications" Conference, Vol 2, Feb 1995.

[2] C. Anderson, A. von Mayrhauser, R.Mraz, "On the use of Neural Networks to guide software testing activities," Proceedings of the 1995 IEEE International Test Conference, UK: Cambridge, 1999.

[3] C. Anderson, A. von Mayrhauser, T.Chen, "Assessing neural networks as guides for testing activities". Proceedings of the 3rd International Software Metrics Symposium, March 1996.

[4] C. Jorgensen, Software Testing: A Craftsman's Approach: Paul CRC Press.

[5] J. M. Hanna and J M Bishop, "A comparison of fast training algorithms over two real time problems".

[6] J. Whittaker, "What Is Software Testing? And Why Is It So Hard?" *IEEE Software*, Vol. 17, No. 1, January/February 2000.

[7] L.V. Kirkland, R.G. Wright, "Using a neural network to solve testing problems," IEEE aerospace and Electronics systems Magazine,vol. 12,no 8, August 1997.

[8] M Anthony, P L Barlett, "Neural Network Learning" Theoretical Foundations Cambridge, UK: Cambridge 1999.

[9] M. F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning Neural Networks", 6(3): ,1993.

[10] M. Last, A. Kandel, "Automated Quality Assurance of Continuous Data" Systematic Organization of Information in Fuzzy Systems, P. Melo-Pinto, H.-N. Teodorescu, and T. Fukuda (Editors), IOS Press, NATO ASI Series, pp. 89-104, 2003.

[11] M. Last and A. Kandel, "Automated Test Reduction Using an Info-Fuzzy Network", T.M. Khoshgoftaar (Ed.), Software Engineering with Computational Intelligence, Kluwer Academic Publishers, pp. 235 – 258, 2003.

[12] M. Last, A. Kandel, O.Maimon, "Information Theoretic algorithm for feature selection", Pattern Recognition Letters, 22(6-7), 2001.

[13] M. Last, Y. Klein, A. Kandel, "Knowledge Discovery in time series databases", IEEE transactions on Systems, Man, and Cybernetics, Vol. 31: Part B,No.1, Feb. 2001.

[14] M. Last, M. Friedman, and A. Kandel, "The Data Mining Approach to Automated Software Testing", Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003), pp. 388 - 396, Washington, DC, USA August 24 - 27, 2003.

[15] M. Riedmiller H. Braun, "A direct adaptive method for faster backpropogation learning The RPROP algorithm" Proc. of the IEEE Intl. Conference on Neural Networks", 1993.

[16] M. Vanmali, M. Last and A. Kandel, "Using a Neural Network in Software Testing Process International Journal of Intelligent Systems", Vol. 17, No. 1, pp. 45-62, January 2002.

[17] M Vanmali., "Using a neural network in Software Testing Process", Master Thesis. University of South Florida, 2002.

[18] O.Maimon, M.Last, "Knowledge Discovery and Data Mining-The Info Fuzzy Network (IFN) Methodology", Kluwer Academic Publishers, Massive Computing, Boston, December 2000.

[19] P. Saraph, M. Last , A. Kandel, "Test Case Generation and Reduction by Automated Input-Output Analysis", Proceedings of 2003 IEEE International Conference on Systems, Man & Cybernetics (SMC 2003), pp. 768 – 773, Washington, D.C., USA, October 5–8, 2003.

[20] P Saraph, M. Last, A. Kandel, "Test Set Generation and Reduction with Artificial Neural Networks", to appear in M. Last, A. Kandel, and H. Bunke (Editors), "Artificial Intelligence Methods in Software Testing", World Scientific, 2004.

[21] R.Demillo, A.Offutt, "A Constraint based Automatic Test Data Generation",IEEE transactions on Software Engineering SE-17,9(Sept 1991).

[22] R Kohavi.,and Li,C-H. "Oblivious decision Trees, graphs and Top-Down pruning" Proc of international joint Conference on Artificial Intelligence (IJCAI).

[23] R. Salomon, "Improved Convergence Rate of Backpropogation with Dynamic adaptation of the learning rate". In hp schwefel and R Manner editors, Lecture Notes in Computer Science, PPSN 1, and 1990.Springer –Verlag.

[24] R. Setiono, "Feed Forward Neural Networks Using Cross Validation" Neural computation 13, 2001.

[25] S. A. Sherer "Software Fault Prediction," Journal of Systems and Software, vol.29, no.2, May 1995.

[26] S Hayken, “Neural Networks: A Comprehensive Foundation”, and Ed. Upper Saddle River, NJ: Prentice Hall, 1999.

[27] T.M. Mitchell, “Machine Learning”, McGraw-Hill Press, 1997.

[28] Z. Q. Luo, “On the Convergence of the LMS Algorithm with Adaptive Learning Rate for Linear Feedforward Networks,” Neural Computation, Vol. 3, 1991.

[29] www.planetsourcecode.com.

APPENDICES

APPENDIX A Algorithms

A. Backpropogation Algorithm For Neural Network Training

η ,n, n_{hid1}, n_{hid2} and n_{out} learning rate, number of input units number of hidden units in first layer,number of hidden units in the second layer and the number of output units respectively.

Step 1 Create a feedforward network with n inputs n_{hid1} hidden units in first layer n_{hid2} in the second layer and n_{out} units for the target layer.Here nout represent the numer of intervals the output has to be discretized into.

Step 2 Initialize all network weights to small random numbers (.05 < random numbers .05), Accuracy = 0

Step 3 Given are K training pairs arranged in the training set $\{(x_1, t_1), \ldots, (x_K, t_K)\}$

where $x_k = (x_{k1}, \ldots, x_{kn})$ and $t_k = (t_{k1}, \ldots, t_{km})$, k = 1……. K

here x_k refers to the input vector where as t_k refers to the output vector. n is the number of inputs and m is the number of output units.

Step 4 For each training example k Do

Step 4.1 Input x_k is presented, and output $o = (o_1, \ldots, o_M)$ is computed

Step 4.2 For each network output unit m where m= 1… M calculate error term δ_m Do

Step 4.2.1 $\delta_m = o_m (1 - o_m) * (t_m - o_m)$

Where t_m is the actual output, which comes from the training data.

Step 4.3 For each hidden unit h2 in second hidden layer Do

Step 4.3.1 $\delta_{h2} = o_{h2} * (1 - o_{h2}) * \Sigma w_{mh2} * \delta_m$

APPENDIX A (Continued)

Where w_{mh2} = Value of the link between node m in the output layer and node number h2 in the hidden layer 2.

Step 4.4 For each hidden unit in n_{h1} in the first hidden layer Do

Step 4.4.1 $\delta_{h1} = o_{h1} (1 - o_{h1}) * \Sigma w_{h2h1} * \delta_{h2}$

Where w_{h2h1} = Value of the link between node hid2 in the hidden layer and node number hid1 in the hidden layer 1.

Step 5 Update each network weight w_{ji} as $w_{ji = Wji} + \Delta w_{ji}$

where $\Delta w_{ji =} \eta * \delta j * xi$

Step 6 If k < K then k = k+1 and we continue the training by going back to Step 3, else go to Step 7

Step 7 Calculate the accuracy. Here accuracy is calculated by seeing the predicted output and the target output. Accuracy is simply the no of times predicted output by the neural network = actual output (training data)/Total no of training records

Step 8 If Accuracy < Acc_{max} Accuracy is set to 0 and a new training cycle is set by going back to Step 3 else terminate the training session.

Here Acc_{max} is the maximum desired accuracy.

Step 9 Obtain the trained network.

APPENDIX A (Continued)

B. Tree construction Algorithm for Info Fuzzy Network

Step 1 - Read tuples (records) of a relation. The algorithm ignores tuples with non-valid or missing target values. Missing values of candidate input attributes can be treated as additional values or ignored in the information-theoretic calculations.

Step 2 - Estimate unconditional (apriori) probability of each value of the target attribute by: $P(Vij) = Oij/n$, where

Oij - number of occurrences of the value No. j of a target attribute No. i in the relation

n - number of complete tuples(records) in the relation

Step 3 - Calculate the estimated unconditional entropy of the target attribute

$$H(Ai) = - \sum_{j=1}^{M_i} P(V_{ij}) * \log P(V_{ij})$$

Where *Mi* - domain size of an attribute No. *i*

Step 4 - Initialize the information-theoretic network (single root node associated with all tuples, no input attributes, and a target layer for values of the target attribute).

Step 5 - If the maximum number of layers (equal to the number of candidate input attributes) is exceeded, stop and return the list of selected (input) attributes Else, go to the next step.

Step 6- Repeat for every candidate input attribute Ai', which is not in the network:

Step 6.1 – Initialize to zero the degrees of freedom and the estimated conditional mutual information of the candidate input attribute and the target attribute, given the final layer of hidden nodes.

Step 6.2 – If *Ai'* is a continuous attribute, then Do:

APPENDIX A (Continued)

The information-theoretic algorithm uses a recursive partitioning procedure (6.2.1 - 6.2.5 below) that generates multiple splits of a continuous attribute at all the split nodes of a given layer.

Step 6.2.1 - Define the boundaries of the interval S to be partitioned as the first and the last distinct values of Ai'.

Step 6.2.2 – Repeat for every distinct value included in the interval S (except for the last distinct value):

Step 6.2.2.1 – Define the distinct value as a partitioning threshold (*T*). All distinct values below or equal to *T* belong to the first sub-interval *S1* (sub-interval No. *1*). Distinct values above *T* belong to the second sub-interval *S2* (sub-interval No. *2*).

Step 6.2.2.2 – Repeat for every node z of the final hidden layer:

Step 6.2.2.2.1 – Calculate the estimated conditional mutual information between the partition of the interval *S* at the threshold *T* and the target attribute *Ai,* given the node *z*, by the following formula

$$MI(T; Ai / S, z) = \sum_{j=0}^{M_{i-1}} \sum_{y=1}^{2} P(Sy;Vij;z) * \log P(S_y;v_{ij}/S,z)$$

$$P(S_y/S, z) * P(V_{ij}/S, z)$$

Where

P (S_y/ *S, z*) - an estimated conditional (*a posteriori*) probability of a sub-interval S y, given the interval *S* and the node *z*.

P (V_{ij}/ *S, z*) - an estimated conditional (*a posteriori*) probability of a value j of the target attribute the node z, and I, given the interval S.

APPENDIX A (Continued)

P (S_y ;V_{ij} / *S, z*) - an estimated joint probability of a value j of the target attribute i and a sub-interval Sy, given the interval *S* and the node *z*.

P (S_y; V_{ij}; z) - an estimated joint probability of a value j of the target attribute i, a subinterval Sy, and the node *z*.

Step 6.2.2.2.2 - Calculate the likelihood-ratio test for the partition of the interval *S* at the threshold *T* and the target attribute *Ai*, given the node *z*, by the following formula Where

$$G^2(T;Ai/S,Z) = 2 * \sum_{j=0}^{M_{i-1}} \sum_{y=1}^{2} Nij\,(Sy, z) * \ln \frac{Nij\,(Sy, z)}{P\,(Vij/\,S, z) * E\,(Sy, z)}$$

Nij *(Sy, z)* - number of occurrences of a value no. j of the target attribute No. i in subinterval *Sy* and the node *z*.

E (*Sy, z)* - number of tuples in sub-interval *Sy* and the node *z*.

P (Vij/ S, z) - an estimated conditional (*a posteriori*) probability of a value j of the

target attribute i, given the interval S and the node *z*.

P (Vij/ S, z)˜ E (Sy, z) - an estimated number of occurrences of a value No. j of the target attribute No. i in sub-interval *Sy* and the node *z,* under the assumption that the conditional probabilities of the target attribute values are identically distributed, given each sub-interval.

Step 6.2.2.2.3- Calculate the degrees of freedom of the likelihood-ratio statistic by:

DF (T; Ai / S, z) = (NI i' (S, z) - 1)˜ (NT i (S, z) - 1)= (2-1)˜ (NT i (S, z) - 1)

= NT i (S, z) – 1

Where

APPENDIX A (Continued)

NI i' (S, z) - number of sub-intervals of a candidate input attribute i' at node z (2)

NT i (S, z) - number of values of a target attribute i in the interval S at node z.

Step 6.2.2.2.4 - If the likelihood-ratio statistic is significant, mark the node as "split" by the threshold *T* and increment the estimated conditional mutual information of the candidate input attribute and the target attribute, given the threshold *T*; else mark the node as "unsplit" by the threshold *T*.

Step 6.2.2.2.5 - Go to next node.

Step 6.2.2.3 – Go to next distinct value.

Step 6.2.3 – Find the threshold T_{max} maximizing the estimated conditional mutual information between a partition of the candidate input attribute *Ai'* and the target attribute *Ai,* given the interval *S* and the set of input attributes *I* by:

T_{max} = arg max MI (T;Ai/Ii,S)
T

and increment the estimated conditional mutual information between the candidate input attribute *Ai'* and the target attribute *Ai.*

Step 6.2.4 – If the maximum estimated conditional mutual information is greater than zero, then do:

Step 6.2.4.1 - Repeat for every node *z* of the final hidden layer:

Step 6.2.4.1.1 – If the node z is split by the threshold T_{max}, mark the node as split by the candidate input attribute Ai'

Step 6.2.4.2 - Partition each sub-interval of S (go to step 6.2.2). If the threshold T_{max} is the first distinct value in the interval S, T_{max} is marked as a new encoding interval and only the second sub-interval is partitioned.

APPENDIX A (Continued)

Step 6.2.4.3 - EndDo.

Else:

Step 6.2.5 - Create a new encoding interval S and increment the domain size of Ai' (number of encoding intervals).

Step 6.2.6 – EndDo.

Step 6.3 – Else (if the attribute Ai' is discrete), Do

Step 6.3.1 - Repeat for every node z of the final hidden layer:

Step 6.3.1.1 - Calculate the estimated conditional mutual information of the candidate input attribute I and the target attribute i, given the node z (*MI (Ai' ; Ai / z*)), by

$$MI(Ai' ; Ai / z) = \sum_{j=0}^{M_i - 1} \sum_{j'=0}^{M_{i'} - 1} P(v_{ij}; v_{i'j'}; z) * \log\left(P(v^{ij}_{i'j'} / z) / (P(v_{i'j'}/z) * P(v_{ij}/z)) \right)$$

P ($V_{i'j'}$/ z) - an estimated conditional (*a posteriori*) probability of a value j' of the candidate input attribute i', given the node z.

P (V_{ij}/ z) - an estimated conditional (*a posteriori*) probability of a value j of the target attribute i, given the node z.

P ($V^{i'j'}_{ij}$/ z) - an estimated conditional (*a posteriori*) probability of a value j' of the candidate input attribute i' and a value j of the target attribute i, given the node z.

P (V_{ij}; $V_{i'j'}$; z) - an estimated joint probability of a value j of the target attribute i , a value j' of the candidate input attribute i' and the node z.

Step 6.3.1.2 - Calculate the statistical significance of the estimated conditional mutual information, by using the likelihood-ratio statistic Where

APPENDIX A (Continued)

$$G^2(Ai'; Ai / z) = 2 * \sum_{j=0}^{M_i - 1} \sum_{j'=0}^{M_{i'} - 1} C^{ij}_{i'j'}(z) * \ln C^{ij}_{i'j'}(z) / (P(v_{ij}/z) * E_{i'j'}(z))$$

$C^{i'j}_{ij}(z)$ - number of joint occurrences of value No. j of the target attribute No. i and value No. j' of the candidate input attribute i' in the node z.

$E_{i'j'}(z)$ - number of occurrences of value No. j' of the candidate input attribute No. i' at the node z. $P(V_{ij}/z) \tilde{} E_{i'j'}(z)$ - an estimated number of joint occurrences of value No. j of

the target attribute No. i and value No. j' of the candidate input attribute i' under the assumption that the attributes i' and i are conditionally independent, given the node z.

Step 6.3.1.3 - Calculate the degrees of freedom of the likelihood-ratio statistic by:

DF (Ai' ; Ai / z) = (NI i' (z) - 1)˜(NT i (z) - 1) degrees of freedom, where

NI i' (z) - number of values of a candidate input attribute i' at node z.

NT i (z) - number of values of a target attribute i at node z.

Step 6.3.1.4 - If the likelihood-ratio statistic is significant, mark the node as "split" and increment the conditional mutual information of the candidate input attribute and the target attribute, given the final hidden layer of nodes (MI (Ai'; Ai / Ii)); else mark the node as "terminal".

Step 6.3.1.5 - Go to next node.

Step 6.3.2 - Go to next candidate input attribute.

Step 6.3.3 - EndDo

Step 7 - Find a candidate input attribute maximizing the estimated conditional mutual information ("the best candidate attribute").

Step 8 - If the maximum conditional mutual information is zero, **stop the search** and go to next step. Otherwise, go to step 10.

APPENDIX A (Continued)

Step 9-Repeat for every unsplitted node z (including the nodes of the final layer):

Step 9.1 Calculate the connection weights linking the unsplitted nodes and the nodes of the final layer to the target layer nodes by :

$$w_z^{ij} = P(Vij;z) * \frac{\log P(Vij/z)}{P(V_{ij})}$$

Where

P(Vij;z) = an estimated joint probability of the value Vij and the node z.

P (Vij/z) = an estimated conditional 9 a posteriori) probability the value V_{ij}, given the node z.

$P(V_{ij})$ = an estimated unconditional (a priori) probability of the value V_{ij}

Step 9.2 Select a value j maximizing the estimated conditional (a posteriori) probability of the target attribute I at the node z (P (V_i / z)) and make it the predicted value of the target attribute I at the node z (j*).

Step 9.3 Go to the next unsplit node.

Step 10 - Add a new hidden layer to the network: make the best candidate attribute a new input (selected) attribute and define new nodes for a Cartesian product of split hidden nodes in the previous layer and the values of the best candidate attribute. Go to Step 5.

APPENDIX B Application Logic For The Testbeds Used For Experiments

A. Application logic for pay calculator

Input : hours worked, Age, Rate_of_pay here hours can take any value from

Output : Gross pay = 0

```
If (hours_worked > 40)
Overtime_hours_worked = hours_worked – 40
Regular_hours_worked = 40
reg_pay = rate_of_pay * reg_hours
ot_pay = rate_of_pay * 1.5  * ot_hours
gross_pay = reg_pay + ot_pay;
/*calculate taxes now
if(age > 18 && age < 70)
tax1 = gross * .023
else
tax1 = 0;
if (gross_pay > 272 || hours_worked>= 30)
uic = gross * .0225
else
uic= 0;
if (gross < 200)
      tax = gross * .18;
```

```
if (gross >= 200 && gross < 300)
      tax = gross * .21
```

APPENDIX B (Continued)

```
if (gross >= 300 && gross < 400)
      tax = gross * .25
if (gross >= 400 && gross < 500)
      tax = gross * .29
if (gross >= 500 && gross < 600)
      tax = gross * .33;
if(gross >=600)
   tax = gross * .35;
netpay = gross_pay – tax1 – uic - tax
```

B. Application logic for heart risk calculator

```
a = 11.1122 - 0.9119 * log(sbp) - 0.2767 * smoking - 0.7181 * log(tc /hdl) - 0.5865 * lvh;
b = 11.0938 - 0.867 * log(dbp) - 0.2789 * smoking - 0.7142 * log (tc / hdl) - 0.7195 *
lvh;
if (sex!= 0)
c = (a - 1.4792 * log(age)) - 0.1759 * tc;
```

```
c = (a - 5.8549) + 1.8515 * log(age/74) *  log(calcArray[1]/74)  - 0.3758 * diabetes;
if (sex!= 0)
d = (b - 1.6343 * log(age)) - 0.2082 * diabetes;
else d = (b - 6.5306) + 2.1059 * log(age / 74) * log(age/74)  - 0.4055 * diabetes ;
e = 4.4181 + c;
```

APPENDIX B (Continued)

```
g = 4.4284 + d;
f = exp(-0.3155 - 0.2784 * c);
calcArray[20] = exp(-0.3171 - 0.2825 * d);
h = (log(no*of years) - e) / f;
k = (log(no..of years) - g) / l;
i = 1 - exp(-exp(h));
j = 1 - exp(-exp(k));
if (j > i)
score = j * 100;
else
score = i * 100;
```

C. Application logic for Loan calculator

```
if (pref_customer == 0)
{
yearly_installment = loan/num_years // calculate the amount borrowed per year.
if(age > 40)
monthly_installment = (3/100) * yearly_installment //month. Install. 3% of YI
else
monthly_installment = (4/100)*yearly_installment
yearly_installment = (yearly_installment + monthly_installment * 12)
```

APPENDIX B (Continued)

```
bimonthly_income = (1.0/3.0) * annual_income
monthly_income = annual_income/12
minamt = monthly_income * .20
if(yearly_installment <= bimonthly_income && minamt >= monthly_installment)
approved = 1
else
approved  = 0
else
yearly_installment = loan / num_years;
if (age > 45)
monthly_installment = (1.5/100)*yearly_installment
```

```
else
monthly_installment = (2.5/100)*yearly_installment
yearly_installment = (yearly_installment + (monthly_installment * 12))
bimonthly_income = (1.0/2.0) * annual_income
monthly_income = annual_income/12;
minamt = monthly_income * .15;
if ((yearly_installment <= bimonthly_income) &&  minamt >= monthly_installment)
approved =  1
else
approved  =  0
if(approved == 1){
```

APPENDIX B (Continued)

```
if(pref_customer == 1){
if(annual_income >= 10000 && annual_income < 20000)
loan_approved = 60/100 * loan
else if(annual_income >= 20000 && annual_income < 30000)
loan_approved = 70/100 * loan
else if(annual_income >= 30000 && annual_income < 40000)
loan_approved = 80/100 * loan;
else if(annual_income >= 40000 && annual_income < 50000)
```

```
loan_approved = 90/100 * loan;
else if(annual_income >= 50000)
loan_approved = loan;}
else{
if(annual_income >= 10000 && annual_income < 20000)
loan_approved = 50/100 * loan;
else if(annual_income >= 20000 && annual_income < 30000)
loan_approved = 60/100 * loan;
else if(annual_income >= 30000 && annual_income < 40000)
loan_approved = 70/100 * loan;
else if(annual_income >= 40000 && annual_income < 50000)
loan_approved = 80/100 * loan
else if(annual_income >= 50000 && annual_income < 55000)
loan_approved = 90/100 * loan
```

APPENDIX B (Continued)

```
else if (annual_income >= 55000)
loan_approved = loan;}
```