A NEURAL NETWORK FOR EVALUATING DIAGNOSTIC EVIDENCE

by

John W. Sheppard
William R. Simpson

ARINC Research Corporation
2551 Riva Road
Annapolis, Maryland 21401

ABSTRACT

The basic problem of diagnosis, which is extremely complex, consists of choosing tests to perform, evaluating the tests, drawing inferences, and forming conclusions. This problem is complicated by difficulties associated with performing the tests, such as inaccurate measurements, unskilled technicians, or incomplete understanding of gathered evidence. Approaches to solving these problems are treated in the literature under the subject of "reasoning under uncertainty." Closely related to reasoning under uncertainty is the problem of determining when enough evidence has been gathered to draw a diagnostic conclusion. We call this the *termination problem*.

Several techniques exist for addressing the termination problem. Some of the approaches we considered include:

- Terminate when all of the tests are complete.
- Terminate when a set of heuristics indicates completion.
- Terminate when reasoning with certainty would terminate.
- Terminate when a pattern of certainty values indicates a conclusion can be drawn.

In our research, we decided to explore the last technique. Because neural networks have been demonstrated to recognize complex patterns, we chose to explore the application of neural networks to this problem.

This paper presents the results of developing a trained neural network that is embedded in a portable maintenance aid called POINTER™—the Portable Interactive Troubleshooter. This neural network is model-independent and may be used on a wide class of diagnostic problems. We review the process by which we selected the network paradigm, gathered the training and testing data set (with a discussion on our uncertainty formulation), trained and tested the network, and incorporated the resulting network into POINTER. We also discuss the overall architecture for reasoning under uncertainty as implemented in POINTER.

INTRODUCTION

The 1980s saw a revolution in the approach to field maintenance for complex systems. The testability and diagnosis considerations continue to be placed deeper and deeper into the design itself. Despite the shift in emphasis, there still exist shortcomings in the field maintenance process. ARINC Research Corporation developed two tools to assist in developing testable systems and in streamlining the maintenance process—STAMP® (System Testability and Maintenance Program) and POINTER. STAMP is a computer-based model that is used to conduct testability analyses and develop fault-isolation strategies to improve system maintenance. The dependency modeling approach incorporated by STAMP permits analysis of a wide range of systems, including digital, analog, digital/analog hybrid, electrohydraulic, and electromechanical. STAMP has been used to analyze systems in various stages of the acquisition process: preliminary design, prototype, redesign, and operational. Several of these applications have included analyses of built-in test (BIT) or have used built-in test equipment (BITE) and other forms of automatic and semiautomatic test equipment. In addition, the particular levels of analysis have varied from macro (full system) to micro (piece-part level), with several levels in between, for example, line and shop replaceable units (LRUs and SRUs) or weapon and shop replaceable assemblies (WRAs and SRAs).

For many systems that STAMP has analyzed, significant improvements have been achieved, and for some systems, order-of-magnitude improvements have been achieved [1, 2]. The software is mature and has been used on more than 50 systems. It is currently being used by ARINC in a number of applications.

POINTER, which was derived from STAMP, uses the system model generated in STAMP for its knowledge base. With the system model, POINTER interactively presents test material and guides the maintenance process through fault isolation. At each step, POINTER examines the current system state to determine the next best test to perform. The basic processes of modeling and knowledge-base development, as well as the testability and fault-isolation output, are described in depth in the literature [3-5] and are not discussed in this paper.

The process of diagnosing complex systems requires determining the tests available, how to perform these tests, the appropriate order for sequencing the tests, and the conclusions to be drawn from the test outcomes. In many diagnostic systems, test outcomes are assumed to be 100% certain, and diagnosis proceeds through a strict partitioning of the answer set into feasible and infeasible conclusions. Frequently, this approach will not adequately solve the problem because of uncertainty in the testing process.

THE PROBLEM

POINTER provides a flexible, interactive approach to fault isolation, taking advantage of the optimization and inference techniques used in STAMP. One of the shortcomings of the POINTER system arose from the assumptions implicit in the STAMP approach to testability—the assumptions that the model is correct and test outcomes are correct. No provision existed for allowing uncertainty in the fault-isolation process. As part of its 1990 in-house research and development program, the ARINC Advanced Research and Development Group incorporated an approach to "reasoning under uncertainty" into the POINTER system. The approach uses a modified form of Dempster-Shafer evidential reasoning described in the literature [6-9].

A typical-fault isolation session in POINTER involves performing tests, evaluating the results of the tests, assigning a value of *good*, *bad*, or *unknown* to the results, and determining the failure. In addition, confidence values for the test outcomes may be specified. For example, in addition to saying a test is bad, the technician is able to say that his or her confidence in that outcome is certain, somewhat certain, somewhat uncertain, or uncertain [8]. These qualifiers are then converted to confidence factors that are used to compute evidential statistics for each conclusion in the POINTER model [9].

The problem addressed in this paper is common to systems using uncertainty in their calculations and can be phrased as follows: At what point has enough information been gathered so that one may declare an answer? In terms of the fault-isolation problem, we are interested in determining the point at which additional testing is providing minimal new information. We call this the *termination problem*.

ALTERNATIVE APPROACHES

The planned approach for using evidential reasoning involves having POINTER collect evidential data during a traditional inferential session. When an inferential conclusion is drawn, the evidential data indicate the probabilities of failure for each conclusion in the model. Several alternative approaches to the termination problem were considered for the POINTER system.

First, fault isolation could stop when POINTER draws a conclusion under normal inference. Here, when POINTER acquires or infers an outcome for each test in the model, it would compute the expected failure with its corresponding probability. Then further testing could be requested by the operator. The problem with this approach is that if additional testing is required after the inferential process normally ends, this testing would be cumbersome.

Second, fault isolation could terminate after all tests are performed at least once. Here, test inference would have no effect on termination, so the result would be comprehensive testing of the system. Unfortunately, optimization of test strategies becomes irrelevant, and the testing could take too long. This approach may be used in conjunction with some other approach, but only as a final resort to end the test process.

Third, a heuristic could be applied. For example, several STAMP and POINTER users surveyed suggested that POINTER should terminate when the highest probability of failure is at least twice the value of the next highest probability [10]. The problem with this approach is that no two users fully agreed on when termination should occur when given actual data, so it is unlikely that any user would be satisfied with a chosen heuristic.

The final approach arose from the observation that the set of probability values for the conclusion describes a pattern of some sort. We thought that it might be possible to train a neural network to recognize the patterns and to determine when to terminate. In order to train the network, we decided to obtain data from STAMP and POINTER users and combine the recommendations of these "experts" into a training set. Obviously, this approach leads to a problem similar to the heuristic approach; however, the advantage to the approach is that the network can be constructed from actual data without having to extract a heuristic from the experts. The neural network approach was ultimately selected as the primary termination determinant.

THE NEURAL NETWORK APPROACH

In constructing the neural network for POINTER, we first examined the form of the available data to determine what information would be useful for the network. We determined that a probability of failure for each conclusion was sufficient. The probability values would be derived from the evidential statistics computed throughout the testing process, as described in the next section.

Second, we needed to determine what type of network was appropriate for the problem. Two general forms of networks exist—supervised learning networks and unsupervised learning networks. In supervised learning, inputs and expected outputs are given to the network during training. The network then learns the relationships between the inputs and outputs and develops an ability to generalize to unknown cases. Unsupervised learning, on the other hand, learns self-contained patterns in the data without the assistance of a teacher. In other words, expected outputs are not provided in training because, generally, expected outputs are not known. Because we surveyed STAMP and POINTER users, we selected supervised learning. In particular, the back-propagation network was chosen because of its proven reliability in situations such as these.

Next the training data set was constructed. This process consisted of determining obvious cases and generating not-so-obvious cases, surveying the experts on how to handle the not-so-obvious cases, and putting the data in a form suitable for the network. The network was then trained using the generated data.

Because it is not sufficient to examine the performance of the network on training data alone, a separate test data set was generated. The data set consisted of situations not covered by the training data, and the network was evaluated based on how it performed using the data. The test data were run through the network, and the results were used to evaluate the network.

EVIDENTIAL DATA

As described, the process for calculating uncertainty in POINTER is a modified form of the Dempster-Shafer evidential reasoning process. Statistics generated from testing using this process need to be transformed into data suitable for a neural network. We followed the approach discussed in this section.

Support and Plausibility

First, support and plausibility values are collected for each conclusion in a given system. As testing is conducted, evidence is gathered that either supports or denies conclusions in the conclusion set. The support measure quantifies the amount of supporting evidence for a given conclusion and is given by the confidence in the test outcome. Plausibility, on the other hand, provides a measure of the denial of conclusions and is simply one minus the support of all conclusions not supported by the test outcomes. These measures are combined as testing proceeds using a modified form of Dempster's Rule of Combinations. See Dempster [7] and Simpson and Graham [9] for more detailed discussions of this calculation.

Bayesian Probability

A traditional approach to reasoning under uncertainty involves computing probabilities using estimated prior probabilities, *Pr*, and known evidence gathered from testing. This evidence is then combined using Bayes' Rule:

$$Pr[concl_i \mid smpt] = \frac{Pr[concl_i]Pr[smpt \mid concl_i]}{\sum_i Pr[concl_i]Pr[smpt \mid concl_i]}$$

where $concl_i$ is the ith possible conclusion in the set of conclusions and *smpt* is the combination of all evidence so far (i.e., the set of symptoms known).

The major problem with the Bayesian approach is determining the prior probabilities. These are usually determined either by tracking real data or by estimating the probabilities. The advantage of the Dempster-Shafer technique is that prior probabilities do not need to be known. Support and plausibility measures are determined solely by testing, the confidences in the test outcomes, and the known relationships between tests and conclusions.

Nevertheless, it is still desirable to use probability values, because these values are familiar to most people. In fact, a relationship is defined between probability and the evidential statistics [7]:

$$Support[concl_i] \leq Pr[concl_i] \leq Plausibility[concl_i].$$

In order to simplify both the neural network and the process of surveying the experts, an estimate of the probability was computed for each conclusion as a function of support and plausibility. The estimated probabilities were then used in polling the experts and as data for the neural network.

The Unanticipated Result

The Dempster-Shafer technique is not without flaws. One of the greatest flaws in the technique lies in the way it records total uncertainty. If any test is performed that provides any evidence in support of some conclusion, then uncertainty is reduced—even in the event of a conflict with known information. Ultimately, uncertainty disappears altogether. This was not a satisfying feature, so Simpson and Graham [9] introduced a new conclusion into the problem: the unanticipated result. The unanticipated result is never denied (i.e., it always has plausibility of 1.0), and it is supported only when a test outcome is inconsistent (i.e., conflicts) with previous test outcomes. Thus, uncertainty becomes a combination of Dempster-Shafer uncertainty and the support for the unanticipated result.

BACK-PROPAGATION

The back-propagation network is called a mapping network because it solves the following type of problem. Assume we have a function, F, that maps a set of inputs, I, into a set of targets, T (i.e., $F: I \rightarrow T$). Assume also that it is impossible to present all instances of I for training (perhaps because I is an infinite set). We want to learn F using a subset of I, which can then be used to generalize to all instances of I.

As a practical approach to the problem, we can choose I_s which is a subset of I, and T_s which is the corresponding subset of T. We will use these two sets to learn the map $F_s: I_s \rightarrow T_s$. We then want to present some i not in I_s to F_s as follows:

$$F_s: i \rightarrow T_s = F(i)$$

This corresponds to the generalization described above.

The back-propagation network solves this problem in the following way. A network is constructed (called a feed-forward network) in which the first layer represents the form of the input data (i.e., input data are transformed so that they can be entered into the network at the first layer), the last layer represents the desired output of the network, (i.e., the targets), and zero or more "hidden" layers are positioned in between the input and output layers.

All of the nodes at a given layer are connected to every node at the next higher layer. Associated with each of these connections is a weight that represents how strongly a signal will pass between nodes. Initially, these weights are set to random values. The process of training the network involves modifying the weights so that the inputs are transformed to the outputs as the signal is propagated.

The term "back-propagation" comes from the learning algorithm. The first time an input is passed through a network, the output has a very low probability of being correct. Back-propagation involves computing an error value corresponding to how much the actual output differs from the expected output. A complete derivation of the back-propagation learning rule is described by Rumelhart, Hinton, and Williams [11].

The actual process of training the network involves multiple presentations of the examples to the network. The network weights are modified gradually, following the error surface, defined by the network's weight space, down the gradient to a local minimum. This is referred to as "gradient descent." When the network settles, we hope it contains a solution to the mapping problem.

THE POINTER NEURAL NETWORK

After examining the data available from the evidential process, we were concerned that a separate network would need to be constructed for each model developed. However, after considering the task and the assumptions implicit in the process, a means by which one network could be used for all models was developed. This has the greatest impact on the input layer.

First, the unanticipated result must be present for all problems. Second, since terminating fault isolation rather than identifying the failure is the network objective, one node should correspond to the highest expected probability value. Third, in order to determine if the highest expected probability is "high enough," it needs to be compared with at least one other value as well as the uncertainty. Because we are still assuming that only a single failure will be isolated at a time, the comparison is made with only the next highest expected probability. Thus, only three nodes are required for the input layer corresponding to highest expected probability, second highest expected probability, and probability of an unanticipated result, respectively. Because the network only has a simple "yes/no" decision to make, the network only needs a single output. Finally, to handle nonlinearities inherent in this problem, we included a hidden layer. After several iterations, the hidden layer was developed with three nodes.

Traditionally, back-propagation uses a sigmoid function for its activation function. In other words, the function is S-shaped and is bound on the bottom and the top by zero and one, respectively. The actual function we used was:

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

where x is the raw output of a given node. The reason for the function is that the back-propagation algorithm requires a differentiable threshold function. The logistic function meets this requirement and behaves extremely well.

DEVELOPING THE TRAINING DATA SET

Given the neural network to solve the problem of terminating fault isolation, we needed to develop a set of training data for training the network. Three steps were followed to generate the data: generating the training cases, surveying the experts, and adapting the data.

Generating the Training Cases

In generating the training data set, we began with a model with 10 conclusions plus an unanticipated result, using known cases. These consisted of the following:

1. One conclusion with 100% probability and all others with 0% probability.
2. One conclusion with 100% probability, the unanticipated result with 100% probability, and all other conclusions with 0% probability. (Note: This is not a realistic case, but it sets the bounds.)
3. All conclusions with uniform probability.

The first two cases resulted in termination, and the third case resulted in no termination.

Because no real data currently existed for this problem, we developed a data generator to generate examples that would not necessarily have an obvious answer. This program went through the following steps:

1. Probability values for the 10 conclusions were generated at random according to a lognormal distribution.
2. Probability values for unanticipated results of half of the cases were generated at random according to the lognormal distribution.
3. The random values for the 10 conclusions were normalized to sum to 1 - Pr[unanticipated result].
4. All values less than or equal to 0.1 were set to zero. This corresponds to less than a uniform value for the 10 conclusions, and these were deemed not to be "reasonable" conclusions (thus set to zero).
5. The 11 probabilities were normalized again.

Surveying the Experts

Forty training cases were generated using the above algorithm. The training cases were presented to a set of 15 experts, and the experts determined how to train the neural network. The survey was conducted in three steps.

First, a miniature version of the survey was constructed in which 11 training cases were given to the experts to evaluate. The purpose of the shorter survey was to educate the experts and get them thinking about the problem. Also, the 11 cases would be repeated in the longer survey and would be used as an indicator of consistency in evaluating the cases.

Second, a longer survey was constructed with 44 training samples. The four extra samples were internal duplicates used to verify consistency. The long survey was given to the experts over one week after the short survey was given.

As an additional consistency check, the experts were asked to circle the candidate conclusions for each case. This request was made because, in preliminary versions of the survey when given to trial subjects, the incorrect conclusion was indicated by two subjects when a failure was indicated. Following the formal survey, one more subject improperly identified the candidate failures.

The third step took place immediately following the long survey and was intended to clarify the opinions and approaches used by the experts. In this step, the experts were asked to respond to eight opinion questions about the problem studied and the survey. Following an examination of this portion of the survey, two-thirds of the expert opinions had to be rejected. A detailed discussion of the survey results is given by Sheppard [10].

Adapting the Data

The final training data set was derived from five expert responses. The five responses were selected following the consistency analysis of the survey results. Once the training set was completed (i.e., the answers were associated with each training case), the data were transformed into data compatible with the network architecture as follows:

1. The maximum probability was assigned to node one.
2. The second highest probability was assigned to node two.
3. The probability of an unanticipated result was assigned to node three.
4. The three probability values were renormalized to sum to one. This ensured that the data were independent of the size of the model.

TRAINING

The neural network was trained using the data generated according to the above process. Training consisted of presenting data to the network 5,000 times. Learning and momentum terms were set to 0.2 and 0.5, respectively. Following training, the network performed with 100% accuracy and a total error (based on the error function used in back-propagation) of 0.007484.

DEVELOPING THE TEST DATA SET

In order to evaluate how well the network generalizes to unknown cases, a set of data representing the unknown cases was generated. This data set contains all permutations of three numbers to two decimal places under the following constraints:

1. The numbers are all in the closed interval [0,1].

2. The numbers all sum to 1.0.

3. The first number is always greater than or equal to the second number.

Exactly 2,601 cases were generated.

TESTING THE NETWORK

All 2,601 test cases generated were run through the trained network. No additional training was done. After running the network, the results were scanned for reasonableness. We found that, as unanticipated result increased, the network became more likely to terminate—indicating that, under uncertainty, the network was conservative with its testing. This was how we wanted the network to perform.

In addition to scanning the test results, we generated a test survey for the experts to complete. The survey consisted of 20 cases selected at random from the test data (Table 1). This survey was given to the five experts used to generate the training set. On average, four of the experts agreed with the neural network on each test case. In one case, the neural network indicated termination was not appropriate where four experts said it was appropriate. This case was at the neural network decision boundary and could have been called either way. Thus, overall, network performance was in agreement with the experts.

INTEGRATING THE NETWORK

The POINTER neural network is one element in a much larger system for handling uncertainty. This system has five components:

1. An inference engine using first-order logic to guide initial evidence gathering.

2. A "qualifier" menu to be (optionally) attached to each test. This menu describes the confidences in the test outcomes and is based on the fuzzy logic work by L. Zadeh [12].

3. A statistical inference engine using Dempster-Shafer evidential reasoning that tracks the support and plausibility measures associated with each conclusion.

Table 1. Results of Validated Survey

Case No.	Expert Responses Yes	Expert Responses No	Network Response
1	4	1	Yes
2	0	5	No
3	5	0	Yes
4	4	1	No*
5	1	4	No
6	4	1	Yes
7	1	4	No
8	5	0	Yes
9	4	1	Yes
10	5	0	Yes
11	5	0	Yes
12	3	2	Yes
13	5	0	Yes
14	5	0	Yes
15	5	0	Yes
16	1	4	No
17	4	1	Yes
18	1	4	No
19	5	0	Yes
20	4	1	Yes

* This is the only disagreement. This case was also borderline for the network.

4. The neural network described in this paper to determine when to terminate fault isolation.

5. A learning procedure to expand the unanticipated result into new conclusions and to modify dependency relationships based on direct conflicts in testing.

The overall POINTER reasoning process was designed to be "user-driven." In each instance of a fault isolation, the computer can suggest testing to be done, but the user can dictate what is to be done and control both the isolation and evidential processes. For example, a user may stop and get an answer whenever he or she deems it appropriate. Also, when the computer suggests an answer, the user may request additional testing. The neural network in this case acts as an expert advisor on when sufficient evidence has been gathered. These and a number of other features are described in detail by Simpson, Sheppard, and Unkle [13]. A more complete discussion of the portable maintenance aid and its approach to reasoning under uncertainty is given by Simpson and Sheppard [8].

CONCLUSION

A unique approach to handling a difficult problem in any knowledge-based system incorporating reasoning under uncertainty has been developed in answering the question: "When has sufficient information been gathered to declare an answer?" By using a neural network to solve this problem, we were able to encode expertise gathered from maintenance experts into the process without explicitly describing the heuristics used. We were able to accomplish this by training the network to mimic known cases and then to generalize to unknown cases. So far, all tests indicate that the network should perform well in a fielded environment in handling fault isolation in an uncertain test situation.

REFERENCES

1. Simpson, W. R., "STAMP Testability and Fault Isolation Applications 1981-1984," *AUTOTESTCON 1985 Symposium Proceedings*, Uniondale, Long Island, New York, October 1985.

2. Esker, E. A., "Testability Analysis: Applications in the Real World," *Proceedings of the IEEE Integrated Diagnostics Symposium*, Dayton, Ohio, December 1985.

3. Simpson, W. R., "Testability and Fault Diagnosis of Airline Avionics," special edition of *PLANE TALK*, AMC Open Forum, Houston, Texas, March 1983.

4. Simpson, W. R., and B. A. Kelley, "Multidimensional Context Representation of Knowledge-Base Information," 1987 Data Fusion Symposium, Laurel, Maryland, June 1987.

5. Kelley, B. A., and W. R. Simpson, "The Use of Information Theory in Propositional Calculus," 1987 Data Fusion Symposium, Laurel, Maryland, June 1987.

6. Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, New Jersey, 1976.

7. Dempster, A. P., "A Generalization of Bayesian Inference," *Journal of the Royal Statistical Society*, Series B, 1968, pp. 205-247.

8. Simpson, W. R., and J. W. Sheppard, "The Application of Evidential Reasoning in a Portable Maintenance Aid," *AUTOTESTCON 1990 Conference Record*, September 1990.

9. Simpson, W. R., and Jerry L. Graham, "Notes on Evidential Reasoning," STAMP Technical Note 0343, ARINC Research Corporation, August 1989.

10. Sheppard, John W., "POINTER Neural Network Survey Results," STAMP Technical Note 0860, ARINC Research Corporation, May 1990.

11. Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, The MIT Press, Cambridge, Massachusetts, 1986.

12. Zadeh, L. A., "Possibility Theory and Soft Data Analysis," in *Mathematical Frontiers of the Social and Policy Sciences*, L. Cobb and R. M. Thrall, eds., Westview Press, Boulder, Colorado, 1981.

13. Simpson, W. R., J. W. Sheppard, and C. R. Unkle, "POINTER—An Intelligent Portable Maintenance Aid," *AUTOTESTCON 1989 Conference Proceedings*, Philadelphia, Pennsylvania, September 1989.