Foundation for neural network verification and validation

Gerald E. Peterson

Gerald E. Peterson, "Foundation for neural network verification and validation," Proc. SPIE 1966, Science of Artificial Neural Networks II, (19 August 1993); doi: 10.1117/12.152651

SPIE.

Event: Optical Engineering and Photonics in Aerospace Sensing, 1993, Orlando, FL, United States

A foundation for neural network verification and validation*

Gerald E Peterson

McDonnell Douglas Corporation
address: mailcode 106 5165, P. O. Box 516, St. Louis, MO 63166
phone: (314) 232-1208 *email:* peterson@mdcgwy.mdc.com *fax:* (314) 232-7499

ABSTRACT

Although neural networks are gaining wide acceptance as a vehicle for intelligent software, their use will be limited unless good procedures for their evaluation, including verification and validation, can be developed. Since neural networks are created using a different methodology from conventional software, extensive changes in evaluation concepts and techniques are necessary. This paper proposes some clarifying concepts related to evaluation, and a process for developing neural networks in which the role of evaluation is emphasized. Some ideas about how the various evaluation activities may be performed are also described.

1. INTRODUCTION

Neural networks are currently gaining wide acceptance as a vehicle for intelligent software. Applications include nondestructive inspection for faults,[2,4,16] prediction of credit worthiness,[11] data fusion,[19] intelligent scheduling in manufacturing,[6] control systems for steering an automobile based on the recognition of features in a digitized television image,[5] and many others. Other critical applications of neural networks, such as their use in the control systems of missiles and airplanes are envisioned.

Neural networks will not actually be used in critical applications, however, unless people are convinced they can be trusted to perform reliably. Achieving this trust is more difficult for neural networks than for other software because the inner workings of a neural network are difficult to explain. The process followed by a programmed software system can be explained to a human and the understanding of what's going on inside goes a long way towards convincing that person that the software is functioning properly, and can be trusted in a critical application. Neural networks are not programmed, and the interaction of the internal functions which govern behavior are not well understood even by the developers, so they are difficult to explain to another human. This aspect of inner mystery makes people leery of using them in critical situations. For example, what would it take for you to be the first to ride in a car that was being driven at highway speeds by a neural network control system?

Current approaches to the verification and validation of conventional software or expert systems are inadequate for application to neural networks. This is true, in part, because neural networks are not programmed in the usual sense, nor are explicit rules written that govern their behavior. Rather, they are trained using a data sample chosen from the population of possible inputs and their corresponding correct outputs. Verifying this training process is a completely different activity from verifying programmed software, so previous verification methods are not applicable.

Even after a neural network is built, it is not possible to predict the output for most inputs. This is in contrast to conventional software where outputs are determined from the inputs by known rules. This nondeterminism means that requirements cannot be specified as exactly for neural networks as they are for other software. Also, it may be impossible to determine in advance exactly what the neural network will be able to achieve. Therefore, in contrast to conventional software, requirements cannot be relied upon as the primary criterion against which a neural network is evaluated. Neural networks must be evaluated against other criteria in addition to whatever requirements can be determined. These criteria include the goals established for the system, a standard development methodology, ideals such as accuracy or completeness, and an *a posteriori* specification of system behavior.

In this paper, a description is given of the activities involved in developing and deploying a neural network with emphasis on those pertaining to evaluation. The intent is to outline a foundation for the future development of a complete

* Work supported by the McDonnell Douglas Independent Research and Development Program
Export Authority: 22CFR 125.4(b)(13)

 0-8194-1202-3/93/$6.00

V&V methodology for neural networks. The scope includes the evaluation activities that occur both during and after the development of the network, up to the point of acceptance by the user. These activities include verifying that a neural network is the most appropriate type of software for the application, verification of the training sample, evaluating the network's generalization capability, evaluating the completed system, and final system validation.

In many respects, this work applies to other varieties of learning systems, such as those based on statistical pattern recognition or decision trees. However, our interest is primarily neural networks and some of the techniques introduced are specific to neural networks. Likewise, much of this work applies to every kind of neural network, but some of it is specific to the most common network: the feedforward multi-layer perceptron trained by backpropagation.

We believe the fundamental concepts related to software evaluation, including verification, validation, and testing, must be more carefully defined, if they are going to apply to neural networks. Because of this, we devote Section 2 of this paper to a clarification of fundamental concepts. We include definitions, a discussion of how the terms relate to one another, a consideration of the evaluation criteria, and a discussion of some misleading ideas which have occurred in the software evaluation literature.

In Section 3 we present an overview of the activities necessary to develop a neural network with an emphasis on those involving evaluation. Sections 4 through 6 consider these activities in more detail. Some conclusions are presented in Section 7.

There is not very much literature which is specific to neural network evaluation. The best previous article is by Tom Schwartz.[20] Schwartz discusses project plans, the proper size of the training data set, the steps that must be evaluated, how to divide the data sample for training and testing, and the accuracy level necessary to deploy the system. Frison[7] discusses the characteristics which make a problem a good candidate for neural network solution, and provides some information about how to test a neural network. The kinds of tasks for which neural networks are suited are discussed by Mortz.[14] There is a large body of literature[10] pertaining to verification and validation of expert systems. Finally, the statistical regression literature[3] contains some information that may relate to evaluation of neural networks.

2. CLARIFICATION OF FUNDAMENTAL CONCEPTS

One of the foundations for effective software evaluation is the establishment of well-understood meanings for common terminology. In this section we give improved definitions for the terms *evaluation*, *test*, *verification* and *validation*. This is necessary because earlier definitions were for the purpose of providing a conceptual foundation for ideal conventional software, but this foundation is not broad enough to underpin the evaluation of neural networks. In addition to definitions of the fundamental terms, we provide several clarifying statements in order to solidify a common understanding. We then focus on the *basis* of an evaluation, i.e. that which we evaluate *against*. We broaden the possibilities for this basis in order to provide for neural network technology. Finally we state and clarify some possible misconceptions about evaluation which could interfere with its application to neural networks.

2.1. Concepts related to evaluation

A dictionary definition of the word evaluation is to *examine and judge*. We build on this to create the following definition.

Evaluation is the process of

1. developing an examination which can be the basis for correct judgments,
2. administering the examination, and
3. using the results to make a judgment about what to do next.

If there are weaknesses, then the purpose of an evaluation is to find them. Otherwise, the purpose of an evaluation is to confirm that there are no weaknesses.

The examination pits the product against everything the builders allege it can do and all properties the builders say it has. These properties do not need to be requirements, they may be the attainment of goals or other achievements. An evaluation should answer questions such as these: "Does this product satisfy its constraints?" "Does it meet the goals set for it?" "Does it meet customer needs?" "Is it the best possible product we can build?" "Are stated strengths actually present?"

Some think of test and evaluation as separate activities: the test is administered and then the software is evaluated based on the test. Our definition is somewhat different in that it includes testing as a part of evaluation. With this broadening, the concept of evaluation includes testing, walkthroughs, audits, verification, validation, and judgments as to whether or not weaknesses remain. We use one word, *evaluation*, to encompass all these related activities.

The most creative part of an evaluation is the development of an appropriate test. The primary problem is that to be effective, the test must be comprehensive. There must be a sufficient number of obvious, average, and boundary test cases.

Verification and validation have been singled out as important kinds of evalution. A clarification of the meanings of these terms is presented next.

Validation has taken place when the the final product passes a comprehensive and feasible test. If the test is not passed and weaknesses are found, then validation fails—the product has *not* been validated. The weaknesses need to be corrected and validation tried again. A validation should have sufficient force to compel acceptance of the product by the customer.

A *verification* is precisely the same as a validation except it is applied to intermediate products or processes. A verification should have sufficient force to compel acceptance by the developers who will use the intermediate product or the results of the process.

A *test* or *examination* is a collection of exercises. Four types of exercises that can be applied to a product or process are shown in Table 1 along with an indication of how they are administered, how they are assessed, and how defects are located. Theorems and proofs are shown for completeness, but proofs of correctness are not possible for neural networks given the present state of the art.

type	applied by	assessed by	errors resolved by
test case	running the program	comparing expected to actual output	discovering why test failed
theorem	proof attempt	whether theorem was proved	discovering why proof failed
question	asking	whether desired answer was obtained	discovering why answer unsatisfactory
scrutinization	scrutinizing	whether product was satisfactory	finding inadequate details

Table 1. Exercise Types

A *comprehensive test* is one that can distinguish a process or product with one or more weaknesses from one without weaknesses, and can pinpoint the location or cause of any weaknesses that are found. A *feasible test* is one that can be administered with a reasonable level of resources.

A major difference between validation and verification on the one hand, and evaluation on the other, is that when performing validation or verification, there is the presumption that the test will be passed. Note that in some cases, a product or process may pass the test even if some small number of weaknesses of an acceptable type are found. Obviously, evaluation for the purpose of finding errors and debugging should precede verification or validation.

The difference between evaluation and validation is illustrated in Figure 1. A product is built and evaluated. If weaknesses are found by the evaluation, then the product is modified. This build-evaluate loop is repeated until no weaknesses are found. Then the product is validated. The validation is not supposed to find additional weaknesses, but if it does, then it goes back to the build-evaluate cycle. If validation is successful, then presumably the customer is convinced of the product's worth, accepts it, and begins using it.

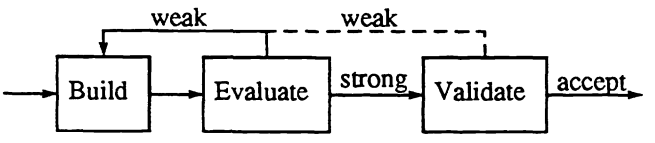

Figure 1. Comparison of Evaluation and Validation

Validation and verification are often done in public for the purpose of convincing potential customers to buy and use the product. Other kinds of evaluation may be done in private for the purpose of detecting and locating weaknesses. For validation or verification, the test results are made available for examination by the customers. Ideally a public certification is made that the test was passed.

Validating or verifying is like testing your best student—the one you expect to ace the exam. A good exam is one which covers all the bases so you feel confident to certify mastery with an "A."

2.2. The basis for comparison

In an evaluation, the examination is for the purpose of comparing the product or process to some desire. This desire is called the *basis* of the comparison. In conventional software development, the basis is the set of requirements. For neural networks, a complete set of requirements which could be used for this basis is difficult, if not impossible, to create. However, *the basis does not need to be a set of requirements*; it could be some goals, a specification of behavior, a standard methodology, or even an ideal such as correctness. The purpose of this section is to clarify the nature of the basis.

We first consider why a complete set of requirements is difficult to create for a neural network. We do this by comparing conventional software development in which the behavior can be determined in advance with software whose exact behavior is not known in advance, that is, *partially nondeterministic software*, such as neural networks.

The contrast between these two types of software is illustrated in Figure 2. When building conventional software, the exact behavior of the completed product can (at least in theory) be specified in advance in a requirements document. The requirements and the behavior are both certain: a given input will result in a precisely specified output. On the other hand, for some kinds of software, such as neural networks or expert systems, there may remain a degree of uncertainty about just what will be output for a given input. This uncertainty is present even after the product is in use, and there is normally even more uncertainty before the product is built.

Some requirements, or more precisely, constraints, are known, but the primary user desires are goals with no certainty that they can be achieved. For this type of software, specification certainty increases as the software is constructed, but never reaches 1. Our thesis is that this partially nondeterministic software can be evaluated, verified, and validated, but the basis will be something other than a complete set of requirements.

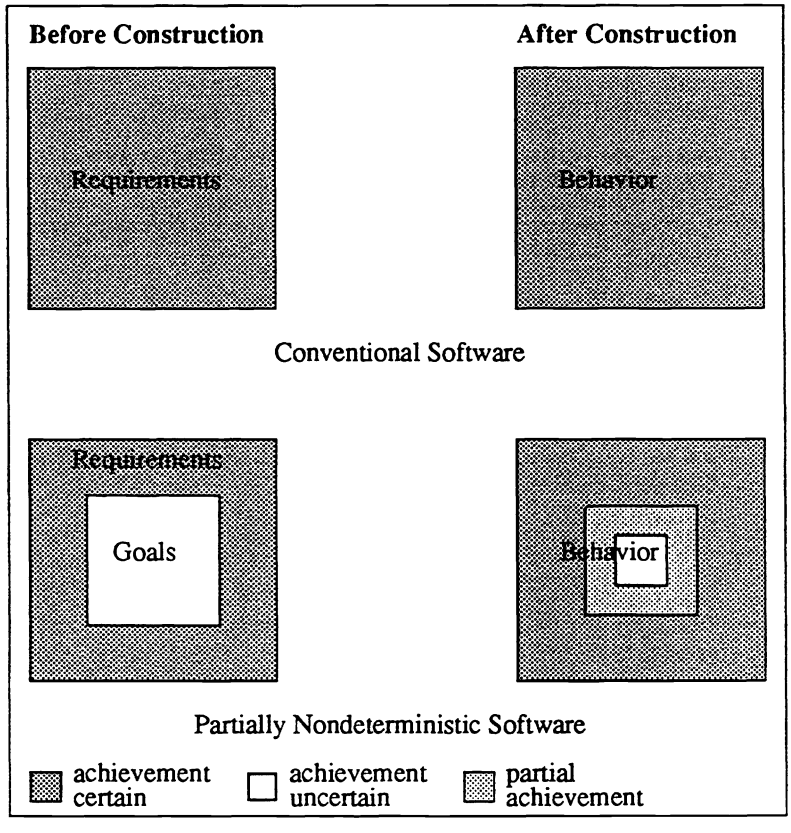

Figure 2. Uncertainty in Software Behavior

The greater the flexibility, the less the need for requirements. For example building a house is quite inflexible—plans must be adhered to closely. On the other hand, building neural networks is quite flexible—only a small amount of effort needs

be expended to change the structure of the network and rerun the training cases. Therefore, in neural network construction, it is quite feasible to proceed with a minimal set of requirements.

We now consider the nature and source of possible bases for evaluating neural networks.

A possible basis will consist of whatever requirements, or constraints, can be specified. For example, it may be possible to sharply specify undesired or "safety" properties. Or it may be possible to precisely specify the input source and format. To the extent they are available, requirements may, of course, be used as the basis.

Another possible basis is the set of goals of the users or developers. If these goals can be quantified, say in a statement containing probabilities, then the extent to which they are achieved can certainly be the basis for an evaluation. However, even unquantified goals which are clear and unambiguous could be used as the basis.

In evaluating the neural network development effort, a possible basis would be a standard development methodology adopted by the developers. The evaluation would pit the actual development methods against the standard. Similarly, when evaluating a data base of training cases, a possible basis would be an ideal such as accuracy or completeness.

Finally, the basis could be a specification of what the system does, written after the product is built. This specification should quantify the final uncertainty and delineate boundaries of acceptable and unacceptable behavior. Then a final validation could pit the product against its final specification.

Evaluation can be against user needs or wants, and validations normally have user needs as their ultimate basis (see the definition of *validation* in the *Software Productivity Consortium Glossary*[17]). However, many user needs are presumed rather than articulated or specified. The presumed user needs are conformance with requirements, meeting of goals, and/or satisfaction of ideals such as accuracy, speed and usability. So the immediate basis for a validation becomes one of those already considered.

2.3. Misconceptions about evaluation

Some of the literature about software evaluation is based on a narrower view than we have elucidated here. Because of this, misleading ideas may arise when comparing our concepts with those of others. In this section several examples of misleading statements are examined. We clarify each statement by reformulating it in light of the concepts of the previous sections. In the following paragraphs, the misleading statement is in italics.

Testing is the process of executing a program with the intent of finding errors.[15] This is too restrictive. Testing may be for the purpose of finding errors, or it may be for the purpose of convincing someone that there are no errors, or it may be for the purpose of defining precisely what the software does.

A successful test case is one that detects an as-yet undiscovered error.[15] Again this is too restrictive. A successful test (not test case) is one that locates errors if there are any, and convinces people that there are no errors in the software if none are found by the test. A test case can successfully contribute to this goal without detecting an error. For example, you may believe all your students will answer a certain question correctly. However, it may be of a fundamental nature and must be answered correctly if you are to certify mastery. This question is a good one even though it does not fulfill the above criterion.

Verification is the process of demonstrating that software possesses features specified by its documentation.[13] This states that the only basis for a verification is the documentation. However, the basis for verification does not need to be in written form, especially if it is an ideal such as correctness or consistency.

The primary goal of verification is to immediately detect and correct software errors.[18] In this statement the concepts of evaluation and verification, as we have defined them, are confused. Rather, in our view, the primary goal of verification is to *convince* that there are no errors. If errors are found during a verification, then the verification will not be very convincing. It may correctly be said that the primary goal of *evaluation* is to immediately locate errors if there are any. First time quality means that verification and validation do not find any weaknesses, i.e. evaluation was successfully used during development.

If there is no requirements specification, then the verification and validation effort must start with the definition of requirements.[8] This assumes that the only legitimate basis for verification or validation is a statement of requirements. This may have arisen from the definitions of validation or verification in a standard glossary[9] in which the basis is defined as the set of requirements. As explained in Section 2.2, we believe there are other possibilities for the basis.

3. THE NEURAL NETWORK DEVELOPMENT PROCESS

In this section we will explain where the various evaluation activities fit in the neural network development process. Figure 3 shows the primary phases in a neural network development with an emphasis on the evaluation activities. Two

guidelines for developing this model were that errors must be recognized, diagnosed, and dealt with as early in the development process as possible; and we should anticipate the types of errors that are likely to occur and incorporate practical controls to minimize them. These guidelines imply that evaluation will be a constantly ongoing activity as the development proceeds. The evaluation steps are shown in Boxes 2, 4, 7, 8 and 10. Note that an evaluation step is always a branch point (sometimes the branch is implicit) because a judgment about what to do next is being made.

Neural network development begins with a desire to achieve some goal. This goal should be explicitly stated in order to provide an initial basis for the evaluation activity. In addition, any constraints that are known, such as the format of the input data, or the programming environment, should also be specified. These activities are shown in Box 1.

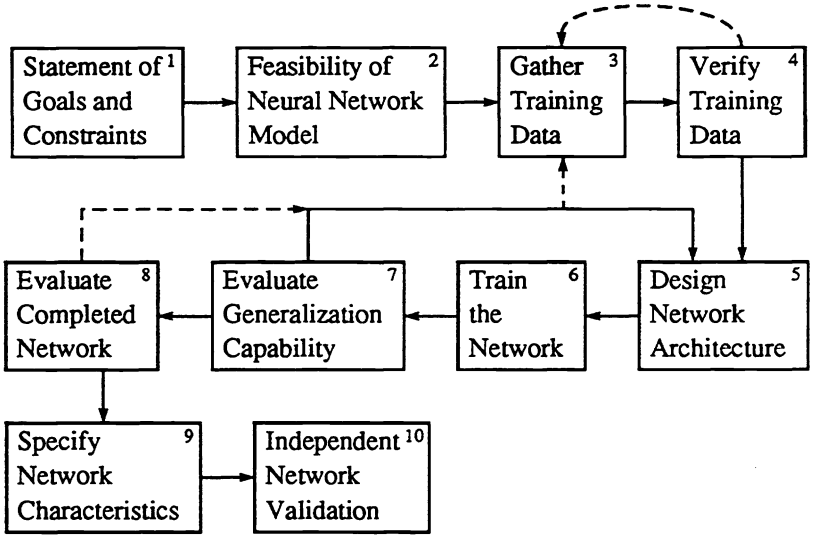

Figure 3. Phases in Neural Network Development

The next activity, shown in Box 2, is to examine the goals and constraints and make a judgment about whether a neural network is the right kind of technology for this task. In order to do this we need to understand the characteristics of the domain in which neural network or machine learning approaches are viable. For backpropagation networks, the characteristics which are thought to be important[7,12] include:

1. *Adequate data is available.* Training a useful neural network usually requires accurate training cases numbering in the hundreds and it is best if a similar amount of data is available for testing. The training data should be representative of the entire universe of experience that the network is expected to deal with in operation. With each training case there must be a known correct response.
2. *The problem is one requiring adaptation to changing situations.* Neural networks can be trained to respond well to many different combinations of inputs. Therefore, a properly trained network can give adequate real-time responses to changing inputs. An example of a network requiring adaptation is that of continuous monitoring of manufactured products for faults.[11]
3. *The problem is of the "pattern recognition" type.* These problems include simple recognition tasks such as recognizing hand-written letters, and more complex problems such as detecting and classifying enemy aircraft. A problem which requires both recognition and adaptation, such as steering an automobile based on the changing view of the road,[5] is a prime candidate for solution using neural networks.
4. *The problem is hard to model.* If the problem is hard to model, then a neural network may be able to build an adequate model, even though other techniques would have nowhere to start. Problems that are hard to model are generally nonlinear and may involve non-Gaussian statistics.
5. *Real-time learning is not required.* Normally it takes time measured in minutes or hours to train a neural network using backpropagation. This is likely to be incompatible with real-time operation.

If the problem is judged achievable by neural network methods, then the training data is gathered and verified (Boxes 3 and 4). The verification will consist of assessing the comprehensiveness and accuracy of the data and judging its appropriateness for use in training the network. It may also be necessary at this stage to decide on a preprocessing strategy for the training data. Neural network training data is commonly preprocessed in order for the desired features to become more "visible." Verification of the training data is considered in more detail in the next section.

The task of designing the network architecture (Box 5) is that of deciding on the type of network to use, the number of hidden nodes, the number of layers, and how to distribute the weights. After this, the network is trained by automatic weight adjustment until the actual output for each of the training cases is as close as possible to the correct response. Then the generalization capability of the network, that is its ability to get near the correct response for previously unseen data, is examined (Box 7), and a judgment made about whether the network should have its architecture altered, or whether a different training algorithm should be employed. If so, then the design and training steps are repeated with an improved architecture or training algorithm. It may be found that poor generalization results from inadequacies in the training data. In this case, the gathering and verification of the training sample is repeated. Enough documentation should be kept to allow someone else to duplicate creation of the net. In Section 5 we examine in more detail the process of evaluating the generalization capability of a network.

Once the generalization capability of the network has been optimized, the final network is evaluated for user acceptability (Box 8), and tests are run for the purpose of delineating the boundary of acceptable behavior and quantifying other characteristics of the network (Box 9). These characteristics are specified and become the basis for the final validation. Then the completed network is examined (Box 10) and a judgment made about its suitability for deployment in the field. Objectivity will be improved if the final validation is made by a team independent from the development team. In Section 6 we examine the purposes and activities of this final validation.

4. VERIFYING THE TRAINING DATA

In this section we describe the ideal characteristics of neural network training data, and some techniques for diagnosing problems with the data.

4.1. Ideal characteristics of the training data

In neural network construction, the training data is used to determine the network. It is very important that this data be representative of the actual data that will be used in the field. To be useful it must be *comprehensive* in the sense that training cases exist for every feature that is to be recognized. Furthermore, sampling should be done more frequently in areas of input space that are changing more rapidly. It must be *correct* because errors in the training data will map to corresponding errors in the software. Thus there can be no contradictions or inconsistencies. If the input data that will be used varies with time, then the training data must be *up to date* in order to be most useful. Normally there should not be any repeated training cases.

In addition, it has been found that networks train better if the data is scaled so that the numerical values of each feature are in a similar range, usually between 0 and 1 or -1 and 1. Additional preprocessing of the data may be desirable in order to make the patterns which are being sought more "visible."

There should be information accompanying the training data which describes its source, its format including scaling information, its scope, why it is considered accurate, and the details of any preprocessing.

4.2. Evaluation techniques

The evaluation of the training data will use as its basis the training data itself, the accompanying documentation, and the ideal characteristics listed above. Initial testing will be done by questions and scrutinizations. One idea is to plot the values of each feature and look for oddities such as outliers and gaps in the data.

Some additional potential problems in the training data may be diagnosed with statistical techniques.[3] Using these techniques it is possible to identify *outliers*, that is, data points that are far removed from the rest of the data; *influential subsets*, that is, data points or subsets which could provide more than their share of influence on the outcome of the training process; and *collinearities*, that is, situations in which the values of one feature are linear combinations of the values of other features.

If any of these potential problems are found, they should be communicated to the neural network developers. The developers, using the techniques which are described in the next section, may be able to assess the extent to which these potential problems actually affect the capability of the trained network. If the network is adversely affected, then remedial action such as deleting an outlier, or adding additional data in order to remove a collinearity, can be undertaken.

5. THE DESIGN-BUILD-EVALUATE LOOP

An important key to building neural networks with minimal risk of use, is the evaluation process that is undertaken as an integral part of the construction activities. This evaluation measures the error in using the network on unseen data, that

is the *true* error of the network, as opposed to the *apparent* error which is measured only over the training cases. If the true error can be estimated, then design decisions such as the number of hidden nodes, the number of inputs, the number of weights, or the amount of training time, can be made using a trial-and-error process. This theme is illustrated in Figure 4. For any design decision that must be made, choose some arbitrary initial value, measure the true error, then move in the direction of decreasing true error until minimum true error has been achieved. Methods for doing this are practical and are beginning to be applied to industrial problems.

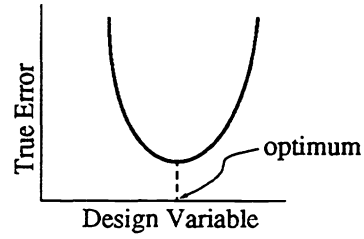

Figure 4. Optimizing a Design Variable

It is tempting to believe that the true error will be close to the apparent error, but experience[21] has shown that the true error is nearly always greater than the apparent error and in common situations the true error can be very significantly greater than the apparent error. The true error is a measure of the generalization capability of the system and the true risk in using the system. Therefore, the goal of design and construction of a neural network is to minimize risk by minimizing the true error.

One way to estimate the true error is to divide the sample cases into two parts, a training sample and a testing sample. The training cases are not used for testing and the testing cases are not used for training. The error on the testing cases will be an estimate of the true error of the system.

The problem with using a separate sample for testing is that sample input-output pairs are usually hard to come by. With a limited number of cases, a better approximation function will be found if all cases are used for training. If we do this, then in order to measure the true error, we must proceed in a manner in which the training cases are *not* used for testing. The most common way to accomplish this goes by the name *cross validation*. The basic idea of cross validation is to leave out some of the sample cases when doing training and use the left-out ones for error estimation. Iterate this many times, each time leaving out different cases. The average of all the error estimates is an estimate of the true error when the system is trained using all the cases.

A final way to estimate the true error is to use an algebraic estimate. One algebraic estimate is the *final prediction error* (FPE) of Akaike.[1] It is defined as

$$\text{FPE} = \text{MSE}(1 + 2\frac{S}{N})$$

where MSE is the mean squared error on the training sample, S is the number of weights in the model, and N is the number of training cases. This is easy to calculate, since S and N are known numbers and MSE is a simple calculation using the training data and the trained network. Moody[12] references some additional algebraic estimates.

The trial-and-error technique may be used to optimize various features of the architecture of a neural network, and may be used to optimize the training time of the network. For a single-layer network, the number of hidden units may be optimized by starting with one hidden unit and increasing the number by one until the FPE, or other estimate of the true error, starts to increase. The number of inputs may be able to be reduced by ordering them according to how sensitive the output is to the various inputs, then deleting them one at a time so long as the estimate of the true error does not increase. Similarly, the weights may be ordered by magnitude or by the effect on the error by a change in the weight. One can then zero them one-by-one until the true error begins to increase. Finally, the training time can be optimized by continuing training only as long as an estimate of the true error decreases.

6. SYSTEM EVALUATION AND VALIDATION

System evaluations are performed on the completed system. After system evaluations have uncovered all the weaknesses and these have been corrected, a final system validation is performed for the purpose of convincing the potential users that

the system is ready for use in the field. In this section we describe the bases of system evaluation, the processes which can be used to accomplish system evaluation, the final system specification, and the final validation.

In Figure 5 we show the manner in which the final evaluation, specification, and validation are related. After the system is completely built, various evaluations of its performance are undertaken. These evaluations will probably uncover some weaknesses. These weaknesses are corrected and the evaluations performed again. This process continues until no additional weaknesses are found. At this point, the system can be frozen and a final specification of its capabilities written. The completed system, along with its final specification, is passed to the validation team. After being convinced of the system's adequacy for their needs by the final validation, the users accept it.

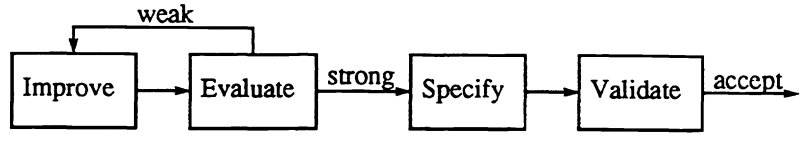

Figure 5. Relation of System Evaluation, Specification and Validation

6.1. The bases for system evaluation

The bases for system evaluation embody the criteria which will be used to judge the strengths and weaknesses of the system. These bases include the documents which were created during system construction, ideal characteristics of neural networks, and the human testers of the system. Specifically, these bases are:

1. the statement of goals and requirements,
2. the training sample and its accompanying documentation,
3. the description of how the neural network was built,
4. ideals which are generic goals for nearly all systems,
5. humans with expertise that the neural network may be trying to emulate, and
6. the potential users of the system.

Here we describe the ideal characteristics of neural networks since the other bases are self-explanatory or have been covered elsewhere in this report.

The following is a list of the ideal characteristics of a neural network in the approximate order of their importance. Those listed last will not be necessary in every case.

1. *Safe.* There is no circumstance under which the system will produce outputs which could cause serious harm to humans or equipment.
2. *Acceptable true error.* The true error has been carefully estimated and is acceptable to the users.
3. *The output is properly sensitive to the inputs.* Each input has a noticeable affect on the output, but a slight modification of an input normally should not cause an extreme change in the output.
4. *Boundary of acceptable performance is known.* There is a way to measure whether the input is within the boundary where acceptable performance has been verified.
5. *Graceful degradation at the boundary of expertise.* Near the boundary of acceptable performance, the fall off in accuracy is gradual.
6. *Robust.* The system continues to provide useful results even if some of the inputs are missing or estimated.
7. *Maintainable.* If the system's inputs vary with time, then retraining with up-to-date training cases and revalidation of performance is easily performed by the users of the system.
8. *Modifiable.* If the system is expected to be used on somewhat modified problems in the future, then the capabilities for performing the modifications are built into the system and instructions for their use are provided for the user.
9. *Network throughput meets the needs of the working environment.*
10. *Easy to use.* The system has been judged user-friendly by potential users.
11. *Cost effective.*
12. *Portable.* The system can be easily modified to run on every potential user platform.

6.2. Techniques for system evaluation

In this section we describe some techniques which can be used for evaluating a neural network. The primary technique is thorough testing, but other techniques such as inspections are also useful. The purpose of system evaluation is to locate

every weakness of the system. The range of available evaluation techniques is more limited for neural networks than for conventional software because neural networks are not programmed in the usual sense. For example, symbolic techniques such as symbolic execution and formal proofs that programs meet specifications are not applicable. Thus there is a greater reliance on testing and a very thorough test must be performed.

System evaluations are broader in scope than the evaluations carried out during network construction. In the construction phase, the goal is optimizing system performance. In the system evaluation phase, the goal is evaluating whether stated requirements, goals, and user needs have actually been met. The techniques used will include:

Inspections. The various documents should be inspected for completeness and accuracy. In particular, an inspection should be made for a clear statement of the difference between valid and invalid output, since the system tests are critically dependent on this information. The training process is evaluated by examining the documents which were prepared during that phase and ensuring that the network was trained in an effective manner. After testing is completed, there is a final inspection whose purpose is the comparison of the testing results with the initial statement of goals and requirements.

Test Case Development. The main problem is that of developing a sufficient number of test cases. In order to adequately test the network, there must be a wide range of both valid and invalid cases, especially cases near the boundary of acceptable behavior. In order to obtain reliable statistical estimates of the true error, there must be a thorough sampling of the input space. If there is a knowledge of the distribution of the inputs that the system will confront, then the test cases should follow the same distribution. If there are known regions of the input space for which the system has difficulty, then these regions should be emphasized in the test set. Sampling should be done more frequently in areas of the input space that are changing more rapidly. If the purpose of the neural network is image recognition, then there must be tests both with and without the image in order to make certain that the image is recognized when it is present and is not recognized when it is not present. When real data can be obtained for testing, then it should be used. However, often the problem is the availability of real test data at reasonable cost. In this circumstance, consider improving the comprehensiveness of the test set by augmenting it with simulated test cases.

Use of simulations. The use of simulators to produce representative inputs for testing the completed network is an effective method for system evaluation. If a simulator is used, it should mirror as much as possible the environment in which the network will be used. The difficulty with simulations is the cost of creating faithful representations of the real world. If a simulation of the entire environment is not possible, then representative real-world scenarios can be considered for simulation.

Testing by experts. Often the task of a neural network is one at which at least some humans excel. If this is the case, then the network's capabilities can be compared with those of a human expert. For example, measures of performance, such as the percent of patterns recognized, can be used to compare the machine with the expert.

Testing by users. The ultimate judges will be the eventual users of the system. It is essential, therefore, that some users be part of the evaluation team. Find out if the system meets their standards of usability. Find out if they make improved decisions by using the system.

Testing for the ideal characteristics. One intelligent way to approach testing is to consider each of the ideal characteristics of a neural network and decide how to test for it. We mention here some ideas about how to approach testing for three of these characteristics.

Testing for safety Ask yourself: "Are there any possible network outputs that could result in harm to people or equipment?" If the answer is yes, then consider trapping such outputs and providing a safe response instead. If traps and alternate responses are not possible, and the system is operating in an inherently unsafe environment, then extremely thorough testing must take place in order to ensure that the network is as safe as possible.

Testing to find the boundary of acceptable behavior. For most neural networks it will be difficult to find the boundary between acceptable and unacceptable behavior. However, if this boundary can be found, even if only in a fuzzy sense, then confidence in using the network can be significantly increased. In general, the boundary is found by providing a wide range of cases in the test set and using testing to separate these cases into valid and invalid categories. Then try to find some simple way to describe the category of valid (or invalid) test data. For example, if the task is to recognize a tank, exhaustive testing may reveal that the tank is recognized 98% of the time in daylight if it is less than 10% obstructed.

Testing for robustness. In order to test for robustness, the data can be modified and the response checked. For example, we can find out what the network does if it is given an outlier; or we can try running the network when some of the data is missing, or if the user estimates the values of missing data.

Field testing. The best way to test any software system is to take it into the field where it will actually be used and let the real users test it in its real environment. If this is feasible, it should be done. If field testing is done, then dribble files or other methods for automatically collecting data about the use of the system should be considered.

Debriefing the testers. A carefully selected set of questions for the users and experts who test the system can aid in system evaluation.

6.3. The final specification of network characteristics

After the completed system has been thoroughly evaluated and all weaknesses corrected, a final specification of network characteristics should be prepared. The final specification will incorporate those goals that were achieved and the initial statement of requirements. The purpose of the final specification is to completely describe the scope and power of the network—to identify what the system can and cannot do. This specification will become a primary basis for the final system validation.

A clear understanding and statement of the boundaries of acceptable system performance is necessary in order to successfully validate the system and to successfully use it in the field. The boundary between acceptable and unacceptable behavior, and the boundaries on inputs in order to ensure a certain accuracy level may be specified in this document. The measures of system performance that are specified are probably statistical in nature. For example, the error rate under various circumstances, or the mean squared error over a certain region of the input space could be specified.

The system developers could consider whether they are willing to provide some kind of performance guarantee to the users. At present performance guarantees are unusual for software, but if the developers are themselves convinced of the accuracy of the completed network, a guarantee would go a long way toward convincing potential users of the adequacy of the system.

6.4. System validation

The final task in developing a neural network is that of demonstrating to potential customers that the system works well enough for them to benefit from using it. This is accomplished by conducting a final system validation whose purpose is to show that the system meets its specification and is adequate for the user's needs. The system validation is similar to the system evaluations, but the purpose of validation is to show that there are no weaknesses rather than to uncover weaknesses. The credibility of the network will depend on the accuracy and appropriateness of its outputs.

An expert system is validated, in part, by explaining to the user the reasoning process on which it is based. Such understanding can go a long way in convincing the user that the machine is effectively handling the problems which it is presented. Unfortunately, given the present state of the art, it is difficult to explain the reasoning process which is used by a neural network. *This fact makes it more difficult to validate neural networks than to validate expert systems or conventional software.* The only way that can be used to convince potential customers of the value of a neural network is extensive and comprehensive testing.

The most convincing test from the user's point of view will be a comprehensive demonstration of the neural network's capabilities on fresh data, perhaps supplied by the user himself. Cross-validation, although theoretically justified, may not convince a user because the same data is being used for testing as for training, and the networks used for cross-validation differ slightly from the one the user will obtain. A calculation of expected test-set error using an algebraic formula may not be convincing because it is based on a difficult theory in which some potentially significant approximations were made.

It is very important that the customer not be misled by developer's claims about the performance of a neural network. It is tempting to show customers the performance results on training cases which will almost certainly be much better than could be achieved on independent test cases. For this reason, it is best if an independent validation of a completed network be conducted by a team other than the developers.

7. CONCLUSION

The primary issue related to evaluation of neural networks is that of user acceptance, now and in the future, for critical applications. Acceptance now requires validation demonstrations that convincingly illustrate the benefits of neural networks. Acceptance in the future requires extensive use in the field of neural networks with satisfied users and no serious problems.

The primary emphasis of this work is that incorporation of evaluation activities, including verification and validation, in the neural network construction process is necessary in order to gain user acceptance now. We have laid a foundation for neural network evaluation by revising and clarifying the fundamental concepts in a way that makes them applicable to

neural networks. We have also described the evaluation activities that should occur as the network is developed. We have defined the *scope* of neural network evaluation activities. However, much work remains to be done to further understand the best possible techniques in each of the evaluation activities. One aspect, in particular, which was not considered here but is of primary importance, is the automation of evaluation activities which will almost certainly be essential in order to test the network with enough data to thoroughly understand its strengths and limitations.

8. REFERENCES

1. H. Akaike, "Statistical Predictor Identification," *Ann. Inst. Statist. Math.* **22** (1970) 203-217.
2. M. Amirfathi, S. Morris, P. O. O'Rorke, W. E. Bond, and D. C. St. Clair, "Pattern Recognition for Nondestructive Evaluation," IEEE Aerospace Applications Conference, Crested Butte, CO, February 1991.
3. David A. Belsley, Edwin Kuh, and Roy E. Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, John Wiley & Sons, New York, 1980.
4. W. E. Bond, D. C. St. Clair, M. M. Amirfathi, C. J. Merz, and S. Aylward, "Neural Network Analysis of Nondestructive Evaluation Patterns," to appear in the 1992 Symposium on Applied Computing, Kansas City, MO, March 1992.
5. Maureen Caudill, "Driving Solo," *AI Expert*, September 1991, 26-30.
6. Cihan H. Dagli, Scott Lammers, and Mahesh Vellanki, "Intelligent Scheduling in Manufacturing Using Neural Networks," *Journal of Neural Network Computing*, Spring 1991, 4-10.
7. Ted Frison, "A General Discussion on Matching Problems to Paradigms," *Journal of Neural Network Computing*, Summer 1990, 45-55.
8. Christopher J. R. Green and Marlene M. Keyes, "Verification and Validation of Expert Systems," *WESTEX-87—Western Conference on Expert Systems*, 1987, 38- 43. Reprinted in [Gu91], 20-25.
9. *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers, Inc, 1983.
10. Uma G. Gupta, ed. *Validating and Verifying Knowledge-Based Systems*, IEEE Computer Society Press, 1991.
11. Jessica Keyes, "Getting caught in a Neural Network," *AI Expert*, July 1991, 44-49.
12. John E. Moody, "The *Effective* Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," in J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.
13. Larry J. Morrell, "Use of Metaknowledge in the Verification of Knowledge-based Systems," *Proceedings of the IEA-AIE*, 1988, 847-857. Reprinted in [Gu91], 176-187.
14. Margaret Mortz, "Fitting the Tool to the Task," *Journal of Neural Network Computing*, Winter 1990, 64-68.
15. Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, 1979.
16. P. O. O'Rorke, S. Morris, M. Amirfathi, W. E. Bond, and D. C. St. Clair, "Machine Learning for Nondestructive Evaluation," Eighth International Workshop on Machine Learning," Evanston, IL, June 1991.
17. Samuel T. Redwine, Jr., editor, *Software Productivity Consortium Glossary*, Software Productivity Consortium, Inc., 1987.
18. A. Essam Radwan, Michael Goul, Timothy J. O'Leary, and Kathleen E. Moffitt, "A Verification Approach for Knowledge-Based Systems," *Transportation Research-A*, Vol. 23A, No. 4, 1989, 287-300. Reprinted in [Gu91], 136-149.
19. George Rogers, Jeffrey Solka, and David Steffen, "A Neural Network-Based F-14 Battle Manager," *Journal of Neural Network Computing*, Spring 1991, 18-28.
20. Tom J. Schwartz, "AI's Future: Neural Networks," *AI Review of Products, Services, and Research* July-August 1991, 51-55.
21. Sholom Weiss and Casimir Kulikowski, *Computer Systems that Learn*, Morgan Kaufmann, 1991.