An expert system for utilizing artificial neural networks

by

Matthew Lelon Reid

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Industrial and Manufacturing Systems Engineering
Major: Industrial Engineering

Signatures have been redacted for privacy

niversity
Ames, Iowa

1994

Copyright © Matthew L. Reid, 1994. All rights reserved.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to thank Commonwealth Edison for their funding of this research. Without their support, this thesis would not have been possible. I would also like to thank the faculty, staff, and students of both the Nuclear Engineering program and the IMSE department who have made me feel at home during my stay here at Iowa State University. I will miss them all. I would especially like to thank the members of the Adaptive Computing Laboratory for their support and input to this research. Special thanks to Dr. Eric Bartlett who has been my major professor, mentor, and guide over the past year during the conducting of this research and writing of this thesis. I would also like to thank Dr. Barta and Dr. Gemmill for being on my program committee. Last, but not least, I would like to thank my parents who have continually supported me in all my endeavors and who have always shown me by example the importance of higher education.

CHAPTER 1: INTRODUCTION

Artificial Neural Networks

The field of neural computation has been rapidly growing over the past few years. "In a recent survey, the trade paper *Electronic Engineering Times* found that 85% of the engineers it questioned in the U.S., Europe, and Japan ranked neural computation as the hottest emerging computer technology" [5](p. 96). This research in neural computation has led to the development of many different artificial neural networks (ANNs). ANNs are computer programs which came about as a result of scientists attempting to model the thought process of the human brain. Most ANNs are used to model the relationship between a vector of inputs and a corresponding output vector by adjusting a set of numerical weights after the repeated presentation of a series of examples. ANNs are currently being applied to projects in such diverse areas as medical diagnosis, credit risk assessment, process control, speech recognition, and optical character recognition [26]. This recent surge of research in ANNs is due in part to several advances in the technology which occurred in the 1980's, which included in part, the development of the backpropagation algorithm [37].

Many users currently use backpropagation neural networks, but as of late many new ANN architectures and algorithms have been developed. Unfortunately, choosing the correct ANN to model a particular problem can be a daunting task. Besides selecting the correct type of neural network, each user must correctly set network parameters such as learning rate,

momentum, number of nodes in the network and sometimes additional variables such as transfer function minimums and maximums, biases, and gain [32](p. 73). Currently the development of an ANN model is more of an art than a science because there are no guaranteed methods for setting network parameters and architecture. There are, however, problem specific heuristics that every expert uses.

Expert Systems

Rule-based expert systems are computer programs which mimic the decision making process of an expert by encoding knowledge into a set of if/then rules. Expert systems often try to capture the knowledge of a human expert in the form of rules which are heuristic in nature, rather than absolute. This coding of decision making rules can often be time consuming and difficult due to the nature of human decision making. The performance of an expert system is usually evaluated by comparing the accuracy of the output with the output from the domain specific expert that the system is attempting to mimic.

Expert systems consist of two major parts, the knowledge-base and the inference engine. The knowledge-base contains rules which are executed based on facts which are input by a user. These facts relate to the specific case of the problem that the user is trying to solve. The knowledge-base for an expert system should be subject to correction as the problem solving rules may change over time. The inference engine, on the other hand, controls how the facts entered by the user will be searched and matched with the rules in the knowledge-base.

Problem Statement

This work describes the development of a rule-based expert system, hereafter referred to as the ANNexpert, which incorporates the author's knowledge in neural computation. The rules for the ANNexpert were developed by comparing the performance of many variations of neural networks, from public domain and commercial software packages, on a series of benchmark problems. The expert system described herein is capable of guiding a novice user through the steps of selecting the proper network architecture and learning algorithm to develop a model for a general data set. In addition, the ANNexpert can instruct the user in the selection of an appropriate software package for model development. The ANNexpert will also act as a training tool for the user by providing educational comments about the use of ANNs. The incorporation of this knowledge into an expert system is an important tool which engineers and scientists can use to select the proper ANN model for their specific applications.

CHAPTER 2: BACKGROUND ON ANNS AND EXPERT SYSTEMS

Introduction

The study of neural networks began many years ago and was inspired by scientists attempting to model the behavior of neurons found in the human brain [18](p. 2). The first successful adaptive computing element was developed by Frank Rosenblatt in 1957 and is referred to as "the perceptron" [9](p. 22). Today, however, the science has shifted toward the construction of adaptive computing networks which are designed to solve engineering problems. With recent advances in computer hardware, neural networks have become more economical.

"A neural network is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and can carry out localized information processing operations) interconnected together with unidirectional signal channels called connections" [17](p. 593). Most neural networks have their processing units (neurodes or nodes) organized into layers, with each node containing a single output which is then channeled to the next layer of nodes via individual connections(or weights). Neural networks function by processing input vectors through this structure and producing a corresponding output vector. Before a neural network can be used, the series of connections must be adjusted, or trained so that the ANN gives the correct output. This is done by presenting the network with a series of training examples and adjusting the interconnecting

weights. This is usually very time consuming, but not always. Adjusting these weights gives the neural network the ability to learn the relationship between the desired variables.

One of the advantages of ANNs is that they are able to correctly model data sets that they have not been trained on. This is known as the ability to generalize. Often when training an ANN, a subset of the training data is not used during training but is set aside to test the network. If the ANN has been trained correctly, it should be able to correctly model both the training data and the test set. If the neural network learns the training set but does not perform well on the test set, then the network is said to have "memorized" the data rather than learning the relationship among the input and output variables.

There are two categories of neural networks, supervised learning networks and unsupervised learning networks. Supervised networks are those which require the presentation of both an input and an output vector during training, while unsupervised networks train with only an input vector. The scope of this research involves exploring the capabilities and use of supervised artificial neural networks.

Backpropagation

Delta Rule Backpropagation

"Without question, backpropagation is currently the most widely applied neural network architecture" [17](p. 593). It was developed by Werbos and published in his doctoral dissertation in 1974, but didn't become widely known until it was rediscovered by Rumelhart, Hinton, and Williams [37]. Backpropagation is a supervised ANN architecture in which the nodes are arranged in a series of layers. The first layer contains a number of nodes equal to the number of values in the input pattern. The first layer is often referred to as the input layer because each node receives an input directly from the input vector for each training

pattern. Each backpropagation network will usually contain from one to three hidden layers, so called because they have no direct contact with the input or output of the network. The top or outermost layer of a neural network has a number of nodes equal to the number of desired output variables. The number of nodes in the hidden layer is usually a function of the number of inputs and outputs and will vary depending on the problem at hand.

Each layer in a backpropagation network is fully connected to the previous layer through the use of numerical weights. For example, each node in the hidden layer is connected to each node in the input layer, and each node in the output layer is connected to each node in the previous hidden layer. Figure 2.1 shows an example of the connection scheme for a backpropagation network with four input nodes, three hidden nodes, and two output nodes. In addition, each node in the network will usually be connected to itself by a weight with a constant input of one, known as a threshold weight.

Backpropagation requires a two step process to update the weight connections. During the first step, known as feed forward, an input pattern in presented to the network and the corresponding output is calculated. Random weights are used during this step to start the network off. The network processes the input from bottom to top as shown by the arrows in Figure 2.1. The input nodes do not perform any type of computation in the network, but simply output the value corresponding to the input for that node. The hidden nodes will sum up the value of the product of the input nodes and the corresponding weights and pass this value through a differentiable, continuous function such as the hyperbolic tangent, linear, or sigmoid function. This process continues through each hidden layer until the output layer is reached. The output layer nodes

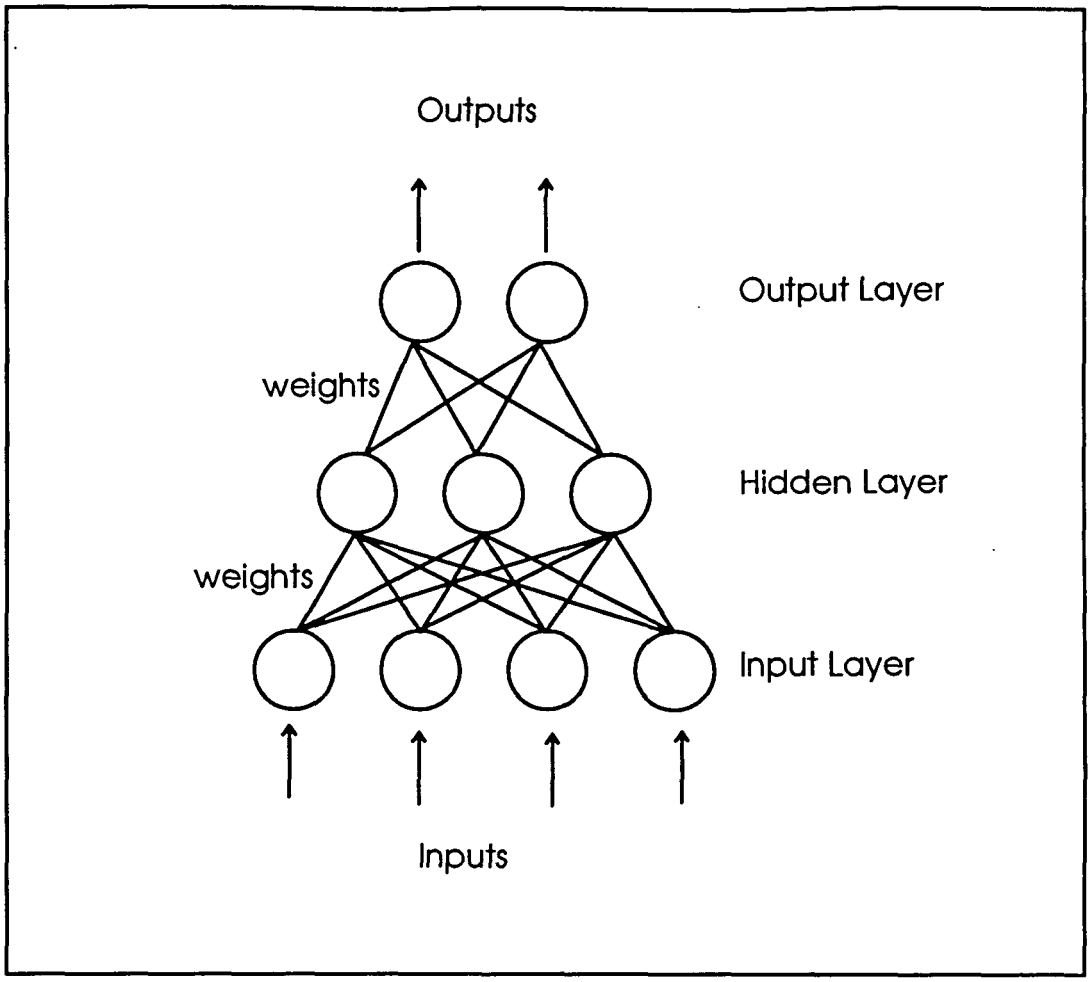

Figure 2.1: Connection Scheme for a Backpropagation ANN

sum up and process the output from the previous hidden layer to give an output value for the network.

The second step of operation for the backpropagation network is known as the backward error propagation stage. During this step the output from the feed forward stage of the network is compared with the desired value for that data pattern. The error for each data pattern is calculated and the corresponding weights for each node in the network are corrected by performing a gradient decent search on the error function of the network. The

network distributes the error evenly amongst all the nodes in the network and will update weights accordingly.

A clear concise derivation of the backpropagation algorithm is presented in Rumelhart, Hinton, and Williams [37]. In general, the process can be summed up in a few simple equations. Assuming a network uses sigmoidal activation functions, the output for the *J*th node, O_j is typically of the form

$$O_j = \frac{1}{1+e^{-(net_j+Q_j)}} \qquad (2.1)$$

where Q_j is the threshold and net_j is the input to node j which is calculated using the following formula.

$$net_j = \sum_i W_{ji} O_i \qquad (2.2)$$

Here, W_{ji} is the weight of the connection from node j to node i of the previous layer and O_i is the output of node i of the previous layer. The error term to be minimized for each pattern is represented by

$$E = \frac{1}{2}\sum_k (T_k - O_k)^2 \qquad (2.3)$$

where O_k is the kth element of the actual output for the pattern, T_k is the kth element of the target output value for the pattern, and E is the total error for the pattern. Equation 2.3 is differentiated using the chain rule to determine the error which is associated with each weight in the network. The weights are then updated by using the delta rule which performs a

gradient decent to minimize the error function. The changes for weights in the network may be expressed by

$$\Delta W_{kj} = \eta \delta_k O_j \qquad (2.4)$$

where η is called the step size or the learning rate which is specified by the user and δ_k is the error associated with node k. The error for an output node k may be calculated by

$$\delta_k = (T_k - O_k)O_k(1 - O_k) \qquad (2.5a)$$

and the error for all other nodes, j may be calculated by

$$\delta_j = O_j(1 - O_j)\sum_k \delta_k W_{kj} \qquad (2.5b)$$

Backpropagation networks are trained until a specified tolerance reached. One such measure is the root mean squared (RMS) error, which is summed up across all output nodes using the following formula

$$RMS = \sqrt{\frac{\sum_p \sum_k (t_{kp} - x_{kp})^2}{n_p \bullet n_k}} \qquad (2.6)$$

where t_{kp} represents the target output for the kth output node of pattern p, x_{kp} is the actual output for the kth output node of pattern p, n_p is the number of patterns and n_k is the number of output nodes. Observing the RMS error during training can help determine if training is occurring properly. If learning rates are set correctly, then a plot of the RMS should slowly

converge to zero [30](p. 22). An ANN is often considered to be a good model if it has an RMS below .05 on both the training set and the test set.

One problem with the backpropagation algorithm is that it doesn't always guarantee an optimal solution and can become trapped in local minimum on the error surface [7](p. 58). Like any gradient descent technique, backpropagation seeks to find a local minimum on the error surface and will head in that direction. However, if there is a local minimum, or a gully on the error surface, the neural network could get stuck in it and fail to converge to an appropriate minimum error. Adding small random "jogs" to weights, the use of a momentum term, or using different random seeds to generate the initial set of weights are some typical ways to reduce the probability of getting stuck in a local minimum.

One last concern when dealing with training a backpropagation network is the order of the data examples as they are presented to the network. Each pattern should be presented in a random order, rather than sequentially, to insure proper training. If a network is trained on data in a sequential fashion and the data is grouped in patterns than the network may train to learn one group of patterns only to lose what it has learned as it moves to another group of data [42](p. 151).

Batch Training

Batch or cumulative learning is often used to speed the convergence of backpropagation networks. In cumulative backpropagation the changes in weights are accumulated over a certain epoch, usually the number of patterns in the training set, and then applied to the individual weights. This approach may increase convergence speed depending on the size of the epoch because using individual updates may only decrease error for a particular pattern but may increase error for other patterns [29](p. 69).

Learning Rate

One of the problems with backpropagation is that learning performance can vary drastically depending on the value of the learning rate η in Equation 2.4. The learning rate should always be a small positive constant, usually between zero and one. If the learning rate is too small, the network could take an extremely long time to converge. If the learning rate is too large, the network weights can grow too large and become saturated [29](p.152). Usually, the network is less likely to fall into a local minimum near the beginning of training, so it is often desirable to start off with a large learning rate, say .9 during the initial stages of training and reduce this as training continues. With smaller learning rates, a network is more likely to perform a true gradient descent on the error surface rather than jumping around sporadically. Some problems may be impossible to learn with large learning rates. In general, the leaning rate should be reduced if the network has a large number of weights and increased for networks with fewer weights [32](p. 79).

Using different learning rates in each layer of a multi-layer network can reduce training time. "In particular, having a larger learning coefficient at the hidden layer than for the output layer allows the hidden layers to form feature detectors during the early stages of training. These feature detectors can then be combined to form more complex detectors at the output layer" [30](p. 54).

Momentum

Another term, momentum, is often added to equation 2.4 to speed convergence of a backpropagation neural network. "It has been empirically shown that one way to increase the convergence rate without causing the error function to oscillate is to include a momentum term..." [4](p. 330). The momentum term increases the speed of descent on the error curve just as it adds speed to a skier gaining momentum down a slope. It is usually a small constant

between zero and one which carries over a small portion of the previous weight change and adds it to the current weight change. Adding a momentum term to equation 2.4 would change it to

$$\Delta W_{kj}(n+1) = \eta \delta_k O_j + \alpha \Delta W_{kj}(n) \qquad (2.7)$$

where n indicates the iteration number and α represents a multiplier to the momentum term.

Hidden Layer(s)

The performance of the backpropagation algorithm may also vary depending on the number of hidden layers and nodes used in the network architecture. There is no maximum number of hidden layers, but there will usually be only one or two in each network. Researchers have discovered that many pattern classification problems can be solved with only one hidden layer [26](p.242). Kudrycki has determined by experimentation that if two hidden layers are used, the optimum ratio of first layer nodes to second layer nodes is three to one [26](p. 242).

A theorem stated by the Soviet mathematician Andrei Kolmogorov, and applied to neural networks by Robert Hecht-Nielsen states that any vector of dimension m may be exactly mapped to another vector of dimension n through the use of a three-layer neural network, provided that the components of the input vector are scaled in the range [0..1]. "Furthermore, the network will have exactly m neurodes in its input layer, n neurodes in its output layer, and $2m+1$ neurodes in its hidden layer" [7](p. 57). If too few nodes are used in the hidden layer(s), then the network may be unable to classify the data and may not converge, or may take an extremely long time to converge. However, if too many hidden nodes are

used, the network may tend to memorize the data, rather than learn to recognize features in the data [7](p. 54).

NeuralWare, a neural network software company, proposes in their literature a heuristic for determining the number of hidden nodes in problems involving the modeling of behavioral data. The number of hidden nodes should be small compared to the number of training patterns since the data is likely to contain noise [30](p. 19-20). They consider behavioral data to be that which involves fluctuations based on the influence of humans such as credit risk assessment or stock market prediction. NeuralWare recommends the following heuristic when dealing with such data:

$$h = \frac{p}{5 \cdot (m+n)} \qquad (2.8)$$

where p is the number of training patterns, m is the number of inputs, and n is the number of outputs, and h is the number of nodes in the hidden layer.

Ward Systems Group, the makers of the Neuroshell2 software package, propose the following formula for the number of hidden nodes in a network [42](p. 35):

$$h = \frac{1}{2}(m+n) + \sqrt{p} \qquad (2.9)$$

This formula is intuitively appealing because many researchers feel that the number of nodes in an ANN should be somewhere between the number of inputs and the number of outputs. The formula satisfies this requirement for problems with few data patterns but will allow an increase in the number of hidden nodes for complicated problems where large training sets are needed to learn the relationship among the data.

Another heuristic used for diagnosis problems is to use the geometric mean of the number of inputs and outputs [26](p. 243). The geometric mean is the square root of the product of the number of inputs and outputs in this case. This technique can often be used to give a reasonable number of hidden nodes when the number of inputs is much larger than the number of outputs. Additional information on the selection of hidden nodes can be found in the Handbook of Neural Computing Applications [26].

Transfer Functions

As stated above, any differentiable continuous transfer function can be used in the nodes of a backpropagation network. Typical candidates are the sigmoid, hyperbolic tangent, and linear functions. A clear discussion of the different types of transfer functions and guidelines for when they should be used can be found in the literature [42](p. 129). One of the considerations when choosing a transfer function is the range of the transfer function. Training data for an ANN is often preprocessed to fit within a certain range, often 0 to 1 or -1 to +1. The transfer function used should be scaled slightly larger than the input data. For example, if the data is scaled between .1 and .9, and a sigmoid transfer function is used, then the transfer function should be scaled between 0.0 and 1.0. Certain neural network software packages such as DynaMind Developer ask the user to input this information.

Another consideration when dealing with S-shaped functions like the sigmoid function is the gain of the function. "The gain of the transfer function is a measure of how sensitive the neuron is to an input" [32](p. 77). A large gain will tend to produce a binary output from the network, while a small gain will tend to produce linear analog output. Using a gain of 1.0 can be practical for most problems. However, the gain should be increased (perhaps to around 10) if the problem requires a mapping from analog inputs to binary outputs and decreased (to around .1) if the problem requires a continuous analog output.

A fairly uncommon transfer function is a radial basis function, such as the Gaussian. It has been suggested that this function may be better suited for continuous function mappings because multiple parameters such as width and center may be adjusted, thus making radial basis functions more adaptive to complex mappings [26](p. 51). A problem which may require two hidden layers with sigmoid transfer functions might be able to be learned by a one layer network with radial basis transfer functions.

Preprocessing of Data

One important aspect of using ANNs that is often overlooked by the novice is the preprocessing of data. To function correctly the range of training data should be slightly smaller than the range of the transfer function used in the nodes of the network [42](p. 127). This is done as a safety measure in case the network encounters data outside of the maximum and minimum values found in the training set. Each input should be normalized to the same interval, otherwise the ANN will receive a bias from certain inputs and will tend to weight the larger inputs as more important. Preprocessing of data can be done using many mathematical techniques, from simple linear interpolation to logarithmic transformations.

Variants of Backpropagation

Fully Connected Networks

Fully connected backpropagation networks are those in which the hidden and output layers are fully connected to all previous layers in the network [42](p. 123). For example, in a three layer network, the output layer would be connected to the hidden layer and the input layer. These ANNs are sometimes referred to as "jump networks", since the weight connections appear to jump over layers. The learning takes place as in normal

backpropagation, but there are simply more weights. Jump networks tend to take longer to train than normal backpropagation networks due to the increase in the number weights.

Recurrent Networks

Recurrent networks are a special version of back propagation which are often used for analyzing time series data such as stock market prediction [42](p. 123). Recurrent networks are usually trained in a sequential order, rather than in a random order. The networks function by having an additional input vector to the network from a previous pattern, so that as the network goes to the next pattern, it will retain the influence from a previous pattern. One problem with recurrent networks is that the information from the first few patterns may be lost since there is nothing to input to the network from previous training patterns. Adding a few "dummy" data examples to the beginning of the set of training examples is recommended to alleviate this problem [42](p. 124). Recurrent networks should not be used on data that does not contain a time influence as this will just add random noise to each pattern as it is presented to the network.

Multiple Activation Functions

Ward Systems Group Inc., incorporates an architecture in their Neuroshell2 software package which features a backpropagation architecture, named "Ward Networks", which has multiple groups of hidden layers with different transfer functions. For example a three layer network might have a hidden layer with 12 nodes in which 6 have a Gaussian transfer function and 6 have a sigmoid transfer function. Different types of transfer functions might be able to recognize different features from the input data and pass these along to the output layer, thus leading to a better prediction [42](p. 125). The total number of hidden nodes used should be chosen as for a backpropagation network.

Cascade Correlation

Cascade correlation is a variation of backpropagation which was developed by Scott Fahlman and Christian Lebere [11]. Cascade correlation works by building up a hidden layer by adding one hidden node at a time until a desired accuracy is obtained. It starts off without any hidden nodes and will incrementally add hidden units which are fully connected to the input and output layers [25](p. 74). The network will then train each new node until a fix number of iterations have elapsed or the network converges. If the network does not learn, another node is added and the previous node's connections remain fixed. Previously trained hidden nodes are referred to as 'tenured' nodes, and are no longer trained. The general idea is that each node is responsible for a certain portion of the desired output. As new nodes are added they are used to develop connections to explain the remaining error in the output and thus reduce the total error on the output.

Cascade correlation has advantages over backpropagation because it will select the fewest number of nodes needed to model a certain problem and then cease training. This would tend to lead to better generalization abilities of cascade correlation networks. Training times are typically longer than standard backpropagation however due to the number of additional steps involved in training, such as training, adding a node, retraining, etc..

Two variants of cascade correlation currently exist. The first allows the training of several 'candidate' nodes before adding a node to the network. This allows for a small group of nodes to be trained separately with different random seeds, and the node that produces the lowest error is added to the network. Another option of cascade correlation allows the network to retrain tenured hidden layer units, rather than leaving their connections fixed. Each of these variants of cascade correlation will increase training time, but might lead to better performance during recall.

Modular Neural Networks

A modular neural network (MNN) is a variation of the backpropagation architecture in which several separate groups of hidden layer units are used to solve the same problem. This can be thought of as using several different competing backpropagation networks to generate a solution to a problem. This is referred to as the use of "adaptive mixture of local experts" as proposed by Jacobs, Jordan, Nowlan, and Hinton [22]. Each of these local experts is a group of hidden layer nodes which are competing with each other to produce the correct output for a given input vector. An additional layer of nodes, referred to as a gating network, is used to decide which of the local experts outputs should be used for each pattern presented to the network [25](p. 235). Both the gating network and the local experts are fully connected to the input layer. The gating network is connected to a layer known as the gating layer which has as many outputs as there are local networks. Each local expert will give an output between 0 and 1 which is normalized in the gating layer and then summed to produce the network output.

MNNs function by separating the input space into several different regions, each of which one local expert is responsible for learning. Training in the MNN is achieved by the backpropagation of error, similar to a typical backpropagation network. "In general, any problem that can be solved with an error-based network, such as a backpropagation network, can be solved at least as well by a modular neural network" [29](p. 237). If a problem is simple and can be learned by a single backpropagation network then one of the local experts will take over and always be chosen as the winner by the gating network and the MNN will behave as a single backpropagation network. One problem with MNNs is that training times will be longer than standard backpropagation due to the fact that several layers of nodes are being trained at the same time.

Probabilistic Neural Networks

Probabilistic neural networks (or PNNs) are not actually ANNs in the classical definition as proposed by Hecht-Nielsen [17], but have an architecture similar to an ANN and incorporate the idea of feed-forward parallel processing of information through a set of simple computation elements. They are actually a relative of a statistical technique developed by Specht to develop decision boundaries to classify data, and a complete derivation can be found in his work [38]. Since they are applied to some of the same problems as neural networks, they are relevant to the discussion here.

PNNs are only applicable to applications which can be formulated as a pattern classification problem in which only one output node generates a positive value for each training example. The PNN clusters training data to develop statistical probability distributions to decide the probability of an input being in a certain class. This can be thought of as the network summing up all the input vectors for all the inputs that belong to the same output class and generating a probability density function (pdf) for each class. An input vector presented to the network during recall is evaluated by a distance metric to determine which class it belongs in.

The PNN does not use an iterative approach to train like other networks, but rather trains in one pass through the data. This training can be done in real time, allowing many advantages over backpropagation. In tests by Maloney, the PNN performed comparatively to backpropagation while offering a speed improvement of 200,000 to 1 [38].

Setting up a PNN is slightly different than other networks. Instead of a hidden layer, the PNN has what is known as a pattern layer, which should have at least as many nodes as there are training patterns for the network. Each input vector used for training in the PNN

should be normalized to a unit length of one [38](p. 112). One way to accomplish this is to normalize the input vectors between zero and one and then divide each field by the square root of the number of training patterns. This process can be avoided by adding the length of the training vector to the pattern layer as outlined by Tseng [40].

There is no learning rate or momentum which the user must specify in a PNN, but rather a constant called a smoothing factor. The smoothing factor determines how the data points will be clustered into the pdfs which are formed for each class. A small smoothing value will cause the pdf function for each class to have many individual humps corresponding to the locations of the training samples, whereas larger values will allow for a greater degree of interpolation between the training classes making the pdf appear to be smoother [38]. Usually, smoothing factors can take on small positive values, usually from .01 to 10, but this depends on the implementation used. Since the smoothing factor is used during recall which is not very time consuming, it is often easier to experiment with the smoothing factor to determine an appropriate value [42](p. 181).

General Regression Neural Networks

General regression neural networks (GRNNs) are a generalization of the PNN and were also developed by Donald Specht. A mathematical derivation of the learning method of GRNNs can be found in his work [39]. They also train in just one pass of the training data and have training times which are orders of magnitude smaller than those required for backpropagation. GRNNs generate an estimate of a continuous output vector by developing a nonparametric regression surface of the elements of the input vectors used in the training examples.

The GRNN is used for prediction and system modeling. While the GRNN may be also used for pattern classification problems, it is best to use a PNN for these types of problems, as the PNN is better suited to solve classification problems [29](p. 171). The inputs and outputs of a GRNN may take on continuous values, but the training data should be normalized before training.

GRNNs have both advantages and disadvantages when compared to backpropagation. Two main advantages are that they learn much faster than backpropagation, and they can be used effectively with sparse data [39](p. 572). The disadvantages are that the recall of a GRNN takes longer than backpropagation recall and the network is memory intensive when dealing with large training sets [29](p. 171).

Similar to the backpropagation architecture, the GRNN has multiple layers: an input layer, hidden layer, and an output layer. The input layer should contain a number of nodes equal to the number of fields in the input vector for each training example, while the output layer should contain one node for each output in the training patterns. Like the PNN, the middle layer of the GRNN should have at least one node for each training example. There is only one parameter for a GRNN that needs to be set by the user. This is a similar to the smoothing factor used for PNNs, and controls the interpolation of the fit of the output function. Since training times for the GRNN are so small, this can often be determined by experimentation.

The inputs to a GRNN should be scaled such that all inputs variables have approximately the same mean and range. Preprocessing becomes less important as the number of training examples gets large or with smaller smoothing factors [39](p. 570). A rough way to approximate this scaling would simply be to normalize all of the input variables to a certain range, say between 0 and 1, by using a linear interpolation.

Radial Basis Function Networks

"A radial basis function network (RBFN) in most general terms is any network which has an internal representation of hidden processing elements (pattern units) which are radially symmetric" [29](p. 265). This means that each hidden node (pattern unit) must have a vector in the input space which acts as a center, a distance measure to determine how far an input vector is from the center, and a transfer function which determines the output of the node by mapping the output of the distance measure. According to this definition PNNs and GRNNs would also belong to the class of RBFNs. Typically however, the name RBFN refers solely to an architecture developed by Moody and Darken [29](p. 266).

A RBFN consists of an input layer, a hidden layer, and an output layer. The number of nodes in the input and output layers is configured as for a backpropagation network, while the number of pattern units must be determined by experimentation. The input layer is fully connected to the hidden layer nodes by means of a series of weights. These weights are adjusted during training by using a clustering algorithm which determines the distance between the pattern unit and the input vector. This occurs during the initial phase of learning which is considered to be unsupervised because there is no feedback from the output layer. Each hidden layer node is connected to the output layer by a set of weights which are updated during the later phases of training by linear regression or a backpropagated error technique.

The RBFN has advantages and disadvantages when compared to standard backpropagation. One advantage is that it tends to train faster than a backpropagation network. It can also lead to better decision boundaries for the input space when used with classification problems. A disadvantage of RBFN is that sometimes important information is lost during the initial phases of unsupervised learning. RBFNs also tend to have problems

with regression tasks since regression may require the ANN to have asymptotic transfer functions [29](p. 268).

Learning Vector Quantization

Learning vector quantization (LVQ) networks were originally developed by Kohonen for the classification of data into groups of similar data [26](p. 142). LVQ networks can only be used for pattern classification tasks and work best on problems with multiple outputs in which only one output node has a positive value for each training example. The goal of an LVQ network is to memorize a set of example training patterns known as exemplars. When presented with a new input vector, the network should produce the output pattern for the exemplar which is closest to the input vector. The middle layer of an LVQ network is actually a Kohonen layer which has a group of nodes which compete with each other to learn the training vectors.

LVQ training starts by computing the Euclidean distance between a training example vector and each node's weight vector. This is done by using the formula [26](p.144)

$$dist_j = (x_i - w_{ji})^2 \qquad (2.10)$$

where $dist_j$ represents the Euclidean distance between the input vector x and each of the j hidden nodes' weight vectors w. The node with the weight vector closest to the input pattern has its weights adjusted so that its weight vector moves closer to the input vector, whereas the other nodes' weight vectors are moved farther away from the input vector.

LVQ networks consist of an input layer, a Kohonen layer, and an output layer. There is no clear method of determining the number of Kohonen layer units to use for an LVQ

network. One heuristic proposed by NeuralWare is that the Kohonen layer should have a number of nodes equal to ten percent of the number of training examples [30](p. 97). However the number of nodes in the Kohonen layer needs to be a multiple of the number of output nodes due to the dynamics of the LVQ network [30](p. 98). Therefore the heuristic should be modified such that the number of nodes is ten percent of the number of training examples rounded to the nearest multiple of the number of output nodes. Each hidden node in the Kohonen layer is fully connected to the input nodes, while each output node is connected to a group, or pool of Kohonen layer units.

Like most ANNs, the performance of an LVQ network relies heavily on the quality of the data used for training. Since the LVQ functions by grouping data vectors based on distance, it does not perform well on input vectors which are far removed from the training data. In these cases it is best to retrain the network with new data that is better representative of the problem the network is attempting to classify.

Directed Random Search Networks

Directed random search (DRS) networks are networks with architectures similar to backpropagation networks but with a completely different method of learning developed by Norio Baba [1]. While many ANNs use a gradient descent method to adjust the network weights to reduce error, DRS networks use random fluctuations of weights to search the weight space in order to determine the optimal weights for a particular problem [25](p. 145). These random fluctuations, or steps through the weight space are accompanied by a directional component which guides the steps in a direction to minimize the total network error. They have the same connectivity as a standard backpropagation network with input, hidden, and output layers.

DRS learning has some advantages over error backpropagation, especially with small to medium size networks. DRS networks tend to provide good results over a problem with a small weight space. DRS networks perform iterations faster than backpropagation networks since there is no error term to be calculated for the hidden nodes in the network. Only the total output error of the network is calculated. There are only two key parameters for the user of a DRS network to adjust. One is the upper bound on the size of the weights which defines the boundary of the weight space that the network will search. If this is set reasonably high there is a good chance that the optimum solution will be found in the weight space. The implementation of the DRS in the NeuralWorks Professional II Plus software package uses a default of 15 for the upper bound on the weight space for most problems [29](p. 154). The second parameter is the initial variance of the random distribution which controls the step size of fluctuations in the network weights. This parameter is not too critical and success has been found by giving an initial variance of 1.0 [29](p. 146).

One disadvantage of DRS networks is that training may take a long time for large problems. If the network contains more than about 200 weights, the weight space which must be searched will be large and will take a long time to converge to an appropriate error [29](p. 145). With these large networks it is probably more effective to use the backpropagation algorithm or some other ANN.

DRS networks minimize the total error in the network adding a random fluctuation to each weight and calculating the new total error. If the new error is lower than the previous error, then the new set of weights is saved. This process is repeated until the total error converges below an acceptable level. The directed component of the network records which previous step directions have provided success and uses these as a guide for the search. The size of each step varies in accordance with the success of the search. If the network encounters several successive improvements, the step size will be increased. If the network

encounters several failures, the step size will be decreased. Bartlett [3] modified and extended the original DRS algorithm to create the stochastic learning algorithm for neural networks.

Self-Organizing Maps

The self-organizing map (SOM) is an unsupervised learning network which was originally developed by Teuvo Kohonen between 1979 and 1982 [29](p. 294). It can often be linked with hidden layers of other ANNs to perform supervised learning. The SOM maps an *n*- dimensional input space into a 2-dimensional map [24]. This map clusters data into groups which can be used as an input to other layers in a network. The layer which performs this mapping is known as a Kohonen layer.

Each node is a Kohonen layer is fully connected to the input layer. The Kohonen layer weights are adjusted by computing the distance between each node's weights and the input vector. This can be accomplished by taking the dot product of the input vector and each weight vector [8](p. 62). The node with the closest distance to the input vector is declared the winner for that vector and has an output of one while all the other nodes will have an output of zero. The winner and all the nodes in a small neighborhood around it will have their weights adjusted so that they move closer to the input vector. This process is completed when all the input vectors have been presented to the network.

One of the advantages of the Kohonen layer is that it can bring out features in the inputs which might not be noticed by a gradient descent technique. After the unsupervised phase, the output of the nodes in the Kohonen layer can be used as input into other layers for additional processing. Training in these networks occurs in two phases. The first is the unsupervised learning phase in which the Kohonen layer organizes itself into a feature map. Next is the supervised learning phase where weights from the Kohonen layer to the rest of the

network layers are trained by a technique such as error backpropagation. Preprocessing the inputs in this way can lead to better decision boundaries and reduced training times.

Madaline III

The madaline III (MRIII) rule is an extension of the ADALINE (ADAptive LINear Element) network which was developed by Widrow [26](p. 90). A madaline is a network composed of multiple adalines, which are processing elements similar to the nodes in a backpropagation network. MRIII is one of several improvements on the original madaline networks. MRIII networks are similar in architecture to the backpropagation network and use the delta rule learning algorithm.

The main difference between backpropagation and MRIII is the method by which the weights in the network are updated. Backpropagation uses analytical equations based on the derivative of the output error, whereas the MRIII network uses a more experimental approach to update the weights. The MRIII network functions by changing, or perturbing, an input to a node by adding a small positive constant. The network then evaluates the effect that this perturbation has on network output error. If the overall error was reduced, the network will change the weights to increase the input of the node. If the network error was greater than before the perturbation, then the network will reduce the pre-node sum. This is repeated for each neuron in the network [32](p. 75).

The MRIII uses an additional parameter, the perturbation parameter, which must be defined by the user of the network. This reflects the magnitude of the change applied to each node during the training. A value of .1 is recommended for activation ranges between -1.0 and +1.0. This value should be increased for larger ranges and decreased for transfer functions with smaller ranges [32](p. 79).

The MRIII rule offers both advantages and disadvantages over the standard backpropagation network. A typical backpropagation network requires fewer operations than MRIII and thus will have shorter training times [44](p. 1437). When dealing with computer simulations of ANNs it is probably best to use backpropagation rather than MRIII. However, MRIII offers a better algorithm for analog hardware design due to variances in hardware [44](p. 1437). "These variances are easily accommodated using Madaline III because they are reflected automatically in the output values of the system for a given input, and hence incorporated into the training" [32](p. 75-76).

Expert Systems

"An expert system is a program that supports high level tasks such as decision-making, classification, and forecasting, by using specific domain knowledge" [4](p. 324). Usually expert systems are built to provide solutions for specific applications in a certain domain, for example medical diagnosis, credit-risk assessment, etc. An expert system will usually recommend a decision or action to answer some problem based on inputs from a user. Rule-based expert systems function by executing, or firing, a set of rules which have been programmed to represent the problem solving logic of an expert, usually human, in a particular field.

Expert systems have three main characteristics [25](p. 309). They are "open to inspection" meaning that the user of the expert system has a method of determining how the expert system has reached its decision. Expert systems should also be easily modified since the rules which are programmed into them may need to be changed over time. Finally, the rules in an expert system are often heuristic in nature, similar to many human decision making processes.

Rule-based expert systems have two main parts, the knowledge-base and inference engine. The knowledge-base is programmed by the developer of an expert system and contains the rules which are used to solve a domain specific problem. The rules for the knowledge-base are developed by a knowledge engineer by consulting with domain experts to try to identify how they make decisions to solve problems. This information is encoded into a set of rules which have the following format:

if *conditions* then *fire*

Each rule will only *fire*, if all the *conditions* for that rule are satisfied. A rule firing will usually result in some kind of action, such as triggering another rule or generating an output for the user. *Conditions* are usually determined using data which have been provided by the user or added to the knowledge-base by another rule. The inference engine controls how the rules in the expert system will be fired by determining how facts and rules in the knowledge-base will be matched and executed.

Two basic strategies of performing the search through the knowledge-base are forward-chaining and backward chaining. With forward chaining, also known as data-driven, expert systems, facts are input to the system as evidence which are used to come up with some hypothesis. With backward-chaining, or goal-driven expert systems, a hypothesis is entered into the system and the system tries to come up with the most likely evidence leading to the hypothesis.

Today, many software packages are available to develop expert systems. These development environments, known as expert system shells, usually contain an inference engine and a knowledge-base editor which the programmer can use to set up rules for the knowledge-base. Many different types of applications may be programmed in these shells by simply modifying the knowledge-base, which can greatly reduce the time required to develop an expert system.

CHAPTER 3: EXPERT SYSTEM RULE DEVELOPMENT

Overview

To develop the rules for the ANNexpert, it was decided that a survey of the capabilities of several different types of neural networks would be needed in order to evaluate which networks were best suited for various applications. To facilitate this, five ANN simulators were chosen to allow for a large variety of neural network architectures and algorithms. A series of nine benchmark problems were chosen to evaluate the different ANNs. The problems were chosen to represent various applications which might be found in industry such as pattern classification, continuous function mapping and time series analysis. The degree of difficulty of these problems ranged from trivial such as the exclusive-or problem, to challenging problems such as complex visual pattern recognition. Over 150 neural networks were trained and tested on the benchmark data sets and the results were recorded for the expert system rule development.

An expert system shell package was chosen to implement the expert system. A decision tree was constructed using the results of the benchmark tests and the input of experts in the field of neural computation. This decision tree was then encoded into a set of rules which were programmed into the expert system shell. The resulting expert system, called ANNexpert, was developed and implemented on a DOS based computer. The ANNexpert was then evaluated using a series of additional problems to insure its validity.

Neural Network Simulators

A survey of ANN simulators and similar products was done in order to evaluate as many different neural network architectures as possible. A potential list of current products on the market was developed and a list compiled. These companies were contacted by phone to obtain information and promotional literature for their products and this information was reviewed in order to choose a set of simulators appropriate for this study. Five were chosen as outlined below.

Nets version 2.01

Nets version 2.01 is a public domain neural network simulator created by COSMIC, a software development group at the University of Georgia which is funded by NASA [2]. It was included in this study because of its widespread availability and negligible cost. Nets is designed to run under DOS on an IBM compatible personal computer. It provides a standard backpropagation algorithm with user defined learning rate and momentum, as well as options for the scaling of transfer functions. The user interface is archaic, with only text menus. Before setting up a network, the user must first write a short file, which tells the software how many nodes and layers are in the network. Training and test files for Nets must be written in ASCII text with parentheses around each data pattern, which can be very time consuming for a user to set up. Statistics for each network trained can be collected by using a "dribble" feature included in the package. The documentation for Nets is simple and easy to understand.

Neuroshell2

Neuroshell2 is a neural network simulator which is available from Ward Systems Group Inc., for approximately $450.00 [42]. It incorporates several basic network architectures including backpropagation, recurrent backpropagation, fully connected backpropagation, Ward nets, PNN, GRNN, and a Kohonen network. Of these, all were evaluated except the Kohonen network, which is an unsupervised network and was not compatible with the benchmark data sets or the objective of this work. Neuroshell2 is easy to use with a mouse driven user interface featuring icons and pull down menus. The particular version evaluated ran on an IBM PC with Microsoft Windows operating system.

Neuroshell2 has many nice features including a variety of tools to incorporate data from training files into networks. It offers an import feature that brings in data from ASCII, Lotus, or Excel formats and incorporates data into a spreadsheet format. Neuroshell2 also offers a beginner's system for ANN novices and an advanced system for experienced users. The beginner's system allows for a backpropagation model with program defined defaults for learning rate, momentum, and number of hidden nodes. With the advanced system, the user may alter many learning parameters or accept defaults from the program. Neuroshell2 also allows the user to select from several different transfer functions such as sigmoid, linear, hyperbolic tangent, Gaussian, and sine. Another special feature of Neuroshell2 is Netperfect, which is used to prevent over training of networks. Netperfect works by removing a subset of data and using it to test during training so that the network does not lose its ability to generalize. This function in Neuroshell2 performs what is known as cross validation. Neuroshell2 also offers a Run-time facility for generating output code to imbed an ANN in another application. The documentation for Neuroshell2 is well organized but a bit remedial for an experienced user.

NeuralWorks Professional II Plus

NeuralWorks Professional II Plus is available from NeuralWare, Inc. for approximately $1,895.00 for a personal computer [29] and contains many popular ANN architectures. Most of the architectures which involve error backpropagation can use any of several learning algorithms including standard delta rule, normalized-cumulative-delta, extended-delta-bar-delta, quickprop, and maxprop. The NeuralWorks architectures included in this study were the following:

- standard backpropagation with sigmoid transfer functions
- norm-cum-delta backpropagation with hyperbolic transfer functions
- cascade correlation
- general regression neural network
- learning vector quantization
- modular neural network
- probabilistic neural network
- radial basis function network
- directed random search network
- self-organizing map

NeuralWorks also offers several different transfer function options including sigmoid, hyperbolic tangent, linear and sine functions. Learning parameters such as learning rate and momentum may be set by the user to change over time, allowing for better convergence for some problems. NeuralWorks utilizes a mouse driven graphical user interface to make network development quick and easy. It allows incorporation of standard ASCII files for training and testing. It also has a feature called Flashcode to generate recall code for trained networks in C for either MS-DOS or UNIX operating systems. NeuralWorks provides

several books which outline the algorithms used in detail while providing simple tutorial manuals for users unfamiliar with ANNs [30].

BrainMaker Professional Development Edition version 3.1

BrainMaker Professional is available from California Scientific Software for $795 [23] and contains one basic algorithm, backpropagation, with many variations. BrainMaker runs under Windows on IBM personal computers and uses a mouse driven user interface. The backpropagation algorithm offered in the software has many features which can be set by the user. These include a dynamic node scheme by which nodes are added to the hidden layer during training if a network fails to converge. BrainMaker also offers a pruning feature to remove the least significant node after training to improve the generalization ability of a network. Another interesting feature of BrainMaker Professional is sensitivity analysis, by which each input is analyzed to determine its effect on the output of the network. BrainMaker comes with ample documentation to aid both the beginner and an expert user.

For an additional $275.00, the Genetic Training Option (GTO) is available to provide a means of optimizing network performance. The GTO helps to create an optimal network configuration by training several networks simultaneously with different learning parameters. Unfortunately, this is computationally intensive and takes large amounts of CPU time due to the large numbers of networks created by this feature.

DynaMind Developer version 4.0

DynaMind Developer 4.0 may be obtained for $1,295.00 from NeuroDynamX, Inc.[32]. It features three main algorithms including backpropagation, madaline III, and True Time. True Time is a version of a recurrent network which is used for time series analysis. True Time was not evaluated during testing. DynaMind allows the user to control a few

parameters such as transfer function gain, transfer function limits, learning rate, momentum, and perturbation (for madaline III). DynaMind only allows two types of transfer functions, sigmoid, and linear. The user interface is menu driven but fairly primitive. One problem with DynaMind is that the recall of a test set after training is time consuming because it is linked to a graphics output which is printed to the screen during recall. Another problem with DynaMind is that the user must preprocess training and test files by using a separate program before entering DynaMind. The documentation available with the software is relatively simple but needs elaboration in some areas.

Benchmark Data Information

Nine data sets were used to evaluate the capabilities of the various architectures from each ANN simulator software package in order to come up with rules for the expert system. They were chosen to represent problems in pattern classification, continuous function mapping, visual pattern recognition, and time series analysis. These problems were chosen to represent actual problems which ANNs might be used to solve in real world applications.

The first benchmark test is a simple binary problem involving the logical exclusive-or (exor) problem, which has a simple non-linearly separable solution space. This problem can be thought of as a simple pattern classification problem. The training set consists of 4 patterns, each with 2 inputs and 1 output. The network is trained to output a 0 if both inputs are the same and a 1 if they are different. The data used as a validation set consisted of 400 patterns with five percent uniform noise added to each input value.

The second benchmark test is a more complex pattern classification problem which contains 8 training patterns each with 3 inputs and 8 outputs. The problem represents a binary decoder in which the inputs are 3 bit patterns representing the numbers zero through seven.

Each of the eight output nodes is trained to fire when the corresponding input bit pattern is presented. For example, if the network is presented with the pattern {0 0 0} the network should respond with {0 0 0 0 0 0 0 1} , where a one is output in the column representing zero. The network was tested on 160 sets of points with five percent uniform noise added to the input values.

The third benchmark is a continuous function mapping of a cosine curve. Each ANN was trained on 50 data patterns, with one input and one output which were generated randomly from a cosine function $y=cos(x)$. The input is a value of x between zero and one and the output is the $cos(x)$. The test set for this problem consisted of 500 test patterns which were generated from a cosine curve and had five percent uniform noise added to each input.

The fourth problem was a more complex continuous function mapping. Each ANN was trained to learn a shifted two input cosine hump given by the following formula

$$z = (0.5 + 0.5 * \cos(\pi + 2\pi x)) * (0.5 + 0.5 * \cos(\pi + 2\pi y)) \qquad (3.1)$$

Each ANN was trained with 50 randomly selected patterns with two inputs and one output. This problem is slightly more complex than the one input cosine function due to the increased number of variables involved. Each network was tested using 900 test patterns with 3 percent uniform noise added to the inputs.

The fifth benchmark problem represents a time-series analysis problem using real world data. The training set consists of 400 hourly air temperature observations from the Omaha Nebraska area for the month of January 1990 which were obtained from the High Plains Climatic Center. The network is given five previous observations and is trained to output the next (sixth) air temperature reading, thus each data vector has five inputs and one

output. Each network was tested on 339 additional observations from the month of January of 1990.

The sixth benchmark problem represents a challenging pattern classification problem where the objective is to calculate from which of two random distributions a set of data points was generated from. This is an interesting problem as it can be used as a method to solve process control problems. For example, when a process is running correctly a measured output will correspond to a certain statistical distribution. When the process has gone out of calibration then the generated outputs will follow a different distribution. Thus if an ANN can recognize which distribution a set of outputs came from, then it can determine when a process is running correctly and when it is producing an incorrect output. The training data set consisted of 200 patterns with ten inputs and two outputs, in which the inputs were ten random points which come from one of two distributions. The first noise distribution is Gaussian with a mean of twenty and a standard deviation of five, while the second is Gaussian with a mean of twenty and a standard deviation of two. Each data pattern output contained a flag to determine which of the two distributions the noise came from. The network is trained to output the corresponding distribution from which the ten inputs were generated. Each network was tested with a separate set of 200 randomly generated patterns.

The seventh problem represented a complex visual pattern recognition problem in which an ANN must decide which of two intertwined spirals a point lies upon [29](p. 221). Each network was trained on 194 patterns with two inputs and two outputs. The inputs corresponded to the coordinates of points in the xy plane and the outputs were binary values which act as flags to identify which spiral each point lies on. The network was tested on 194 patterns with two percent uniform noise added to the inputs. A graph of the normalized training data is shown in Figure 3.1, while a graph of the testing data is shown in Figure 3.2.

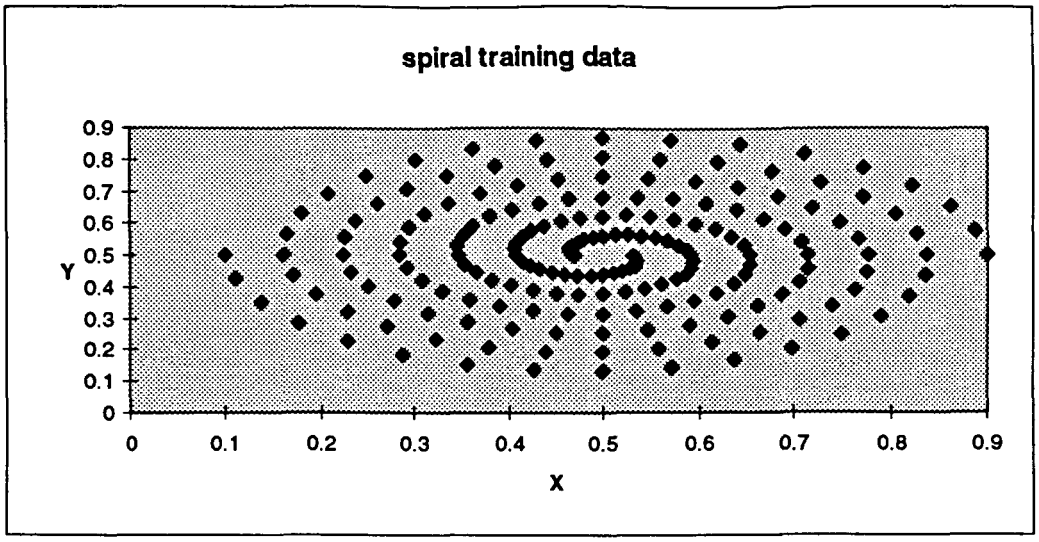

Figure 3.1: Spiral Training Set

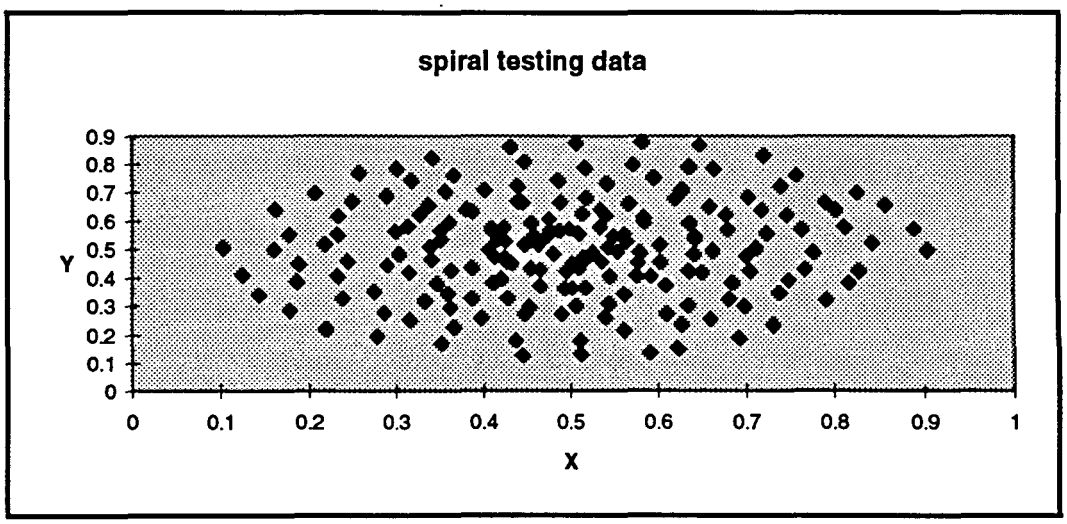

Figure 3.2: Spiral Test Set

The eighth benchmark is a problem in which a network must classify iris flowers into three categories based on four indicator variables: sepal length, sepal width, petal length and petal width. This taxonomy problem was originally proposed by Fisher in 1936 [13], and is used by many researchers to test ANNs. The training set consisted of 100 patterns with 22 inputs and three outputs. The inputs are all binary and consist off a code which records an approximation of the four indicator variables. Each output is an indicator to tell which class of flower (Setosa, Versicolor or Virginica) the inputs belong to. Each network was tested on a test set consisting of 150 real world data patterns.

The final benchmark problem is a visual pattern recognition in which a network must learn to recognize and classify the capital letters of the English alphabet. Each of the 26 data patterns consists of 49 binary inputs which correspond to a 7x7 grid in which one capital letter is embedded. The network outputs a five digit binary code, which gives a number from one to twenty-six, to identify the corresponding letter. The test set contained 260 patterns with five percent uniform noise added to each input.

Benchmark Testing

Hardware

All benchmark tests were done on an APEX IBM compatible 486 PC. Each neural network simulator was loaded onto the hard drive. The system had a speed of 66 MHz and 8 megabytes of RAM. The operating system was MS-DOS version 6.0 and MS-Windows version 3.1. A SAMPO Alphascan 15 high-resolution monitor was used to interface with the system.

Methodology

Experiments were run with the benchmark data sets to evaluate the ANN algorithms and architectures in the 5 neural network simulators. Each data set was normalized between .1 and .9 to preprocess them before running the experiments. A few of the pattern classification problems such as the iris classification and the noise classification problem, had the output vector of each training pattern normalized between 0 and 1 so that they could be used with ANNs which require binary outputs.

Each ANN model was run using the default settings available in those packages. The number of hidden nodes used for gradient descent networks was $2m+1$, where m is the number of inputs. The exceptions to this were the networks developed using the Neuroshell2 package, in which the default number of nodes specified by the software was used. On the iris classification problem only 10 hidden nodes were used since the 22 inputs only correspond to 4 actual variables. Most networks tested had a single hidden layer except those used for the two-spirals problem. For this problem, all the backpropagation networks where given two hidden layers with 10 nodes each due to the complexity of the problem. Other algorithms such as the PNN and GRNN had a number of hidden nodes equal to the number of training examples. Each neural network which used a gradient descent on the network error was trained down to an RMS between .03 and .055, and the training times were recorded. Networks which couldn't reach an error of .055 in a reasonable time were given the label "does not converge", and were no longer tested. Networks which had an RMS between .03 and .055 had their training RMS recorded and then were tested using the test set for each benchmark test. For those networks that train in only one pass, such as the PNN and GRNN, only one pass was made through each training file and the RMS was recorded if it was below .055. If the network failed to reach an RMS below .055 it was given the label "does not

converge", and was not tested. Those with an error below .055 were then tested using the testing data.

Evaluation Criteria

Since some networks had a lower training RMS than others, a method of normalizing the results was needed. This normalization was intended to remove any unfair bias which networks might have due to being trained down to a smaller RMS error than other networks. This term is known as the generalization score and is computed by dividing the testing RMS by the training RMS. The generalization score shows how well the network can generalized what it was trained on. A low score would indicate that the network generalizes well, while a high score would indicate that the network has only memorized the training set. This score was only used to compare ANNs which used the RMS error as a stopping criteria for training. The score was not used for the PNN and GRNN since they do not use RMS error as a stopping criteria but rather train in one pass of the training set. The generalization score was also not used for the LVQ network since it often had a RMS error of zero on the training set, making the calculation invalid.

Other criteria, in addition to the generalization score, for evaluating how well each network performed were training time, training RMS error, and testing RMS error. When comparing software packages two additional criteria were used to determine which software package to recommend. These were ease of use and consistency of performance.

As an example of how the network architectures were evaluated, consider the two spirals problem. The PNN from Neuroshell2 was considered to be the best network since it was the only network to learn the training set to an RMS error below .055. On the 2 input cosine problem different criteria were used. The Ward Net from Neuroshell2 was selected as the best architecture for this type of problem even though two other architectures (Nets

version 2.01 backpropagation and DynaMind Developer backpropagation) had lower generalization scores. For this problem, the Ward Net was chosen as the best network because the generalization score was good and it also had the fastest training time. In addition, the Neuroshell2 software package is much easier to use than either of the other two packages.

Results

The results for all benchmark tests are shown in Table A.1 in Appendix A. The table shows the number of hidden nodes, training RMS error, testing RMS error, generalization score and training time for those networks which converged to an RMS below .055. By looking at the results of the experiments and by comparing the performance of the networks many interesting results were discovered. The results of these tests were used along with knowledge of ANNs to develop a set of heuristics for using neural networks for different types of applications. As can be seen from the tests, certain neural networks do indeed perform better and solve problems faster than other ANNs.

For the continuous function mapping problems such as the cosine and two input cosine problems, backpropagation networks performed the best. Since there is usually only one output for most function mapping problems, certain ANNs such as the LVQ and PNN are not applicable to this class of problems. On the cosine problem, the best network turned out to be the fully connected back propagation algorithm. This ANN had a generalization score of 1.09 which was about 2.2 standard deviations below the mean of 1.74 for all the applicable networks. On the two input cosine problem, one of the best networks was the backpropagation network with multiple hidden layer slabs with different transfer functions developed by using the Neuroshell2 software package. This network had a much faster training time then all other networks and was still able to learn the problem sufficiently. These

results would indicate that backpropagation networks are best at continuous function mapping problems. On the two input cosine problem some networks were not even able to converge such as the self-organizing map, general regression neural network, cascade correlation, and radial basis function network.

LVQ networks, PNNs, and GRNNs performed well on most of the pattern classification problems. One problem, however, is that the implementation of the LVQ network and PNN analyzed in this study are only applicable to problems with more than one output in which the outputs are binary and only one output fires for each training pattern. Due to this criteria, neither network was applicable to the exor problem. The GRNN, however, performed exceptionally on the exor problem. The Neuroshell2 version of the GRNN was able to train on the exor problem to an RMS of .006 in a matter of seconds. When tested the network performed favorably with an RMS of .008. On multiple output pattern classification problems such as the iris classification or the binary decoder problem, the LVQ network and the PNN performed better than the other networks tested. The version of the PNN from Neuroshell2 outperformed the version from NeuralWorks on the iris problem. This is due to the fact that the NeuralWorks version requires special normalization of the input vectors to insure good training. The PNN from Neuroshell2 did well on the iris flower classification problem and was able to train down to an RMS of 0.000 and had a test RMS of .115. The LVQ network performed better with a training RMS of 0.000 and a testing RMS of .092. The backpropagation network from Neuroshell2 also performed well on this problem. The LVQ and PNN also performed extremely well on the binary decoder problem, producing both a training and testing RMS of 0.000. For the noise classification problem, the best network was the backpropagation network from BrainMaker. It had a training RMS of .038 but only a .188 RMS on the test set. It would appear that the network only memorized

the data rather than generalizing it. Since this problem could not be learned well by most of the ANNs studied, it is probably best solved by using other methods.

For the time series analysis problem, most of the networks did well. This is probably because many data values were used to train each network and the data had a high correlation over time. The GRNN network did not converge for this problem so it is evident that this architecture is not well suited for time series analysis. Most of the other networks performed well although backpropagation networks performed faster than the DRS, cascade correlation, and RBFN networks.

For the visual pattern recognition problems, networks which were good at pattern recognition performed well. The two-spirals problem turned out to be extremely difficult for most ANNs. The only architecture which performed well was the PNN architecture available from Neuroshell2. For the character recognition problem, backpropagation networks did well, but the GRNN performed better by almost completely learning both the training and test set. The LVQ network and the PNN were not applicable to the character recognition problem since more than one output has a positive output for certain training patterns. If the problem was restated such that only one output fired for each pattern, than it is assumed that the PNN and LVQ network would also perform well on this problem.

Expert System Implementation

Eclipse Expert System Shell

An expert system shell package was chosen to aid in the implementation of the expert system. After a brief survey of expert system shells on the market and after consulting with experienced users, Eclipse version 3.2c was chosen due to its flexibility and low price (approximately $1,000)[16]. Eclipse is provided by The Haley Enterprise and is a shell which

supports an MS-Windows graphical interface and both backward and forward chaining. It is based on a language known as CLIPS, which is a language written in C which is derived from LISP. One desirable feature of both CLIPS and Eclipse is that they allow the system developer to write C based subroutines which may be incorporated into the expert system. For this application, the forward chaining method of operation was utilized.

Figure 3.3 shows a typical rule in Eclipse. This rule, *example-rule*, would correspond to the following logic: *if* a and b *then* c. Every fact above the symbol, "=>", corresponds to conditions which must be true to satisfy the *if* portion of the rule while everything below the symbol are actions carried out in the *then* portion of the rule. In this case, if the two facts, *fact a* and *fact b* are found in the fact-list, then the rule *example-rule* will then be fired and the fact, *fact c*, will also be added to the fact-list. The addition of *fact c* to the fact-list may then trigger the firing of another rule. In this way a hierarchy of rules may be built in a tree-like fashion in which the firing of one rule will lead down the branch of a decision tree leading to a final solution.

```
(defrule example-rule
     (fact a)
     (fact b)

=>
(assert (fact c)))
```

Figure 3.3: Sample Eclipse Rule

Discussion of Rules

By examining the results of the benchmark tests and by consulting with experts in the field of ANNs, a decision tree was developed (see Figure 3.4) to decide how to select an ANN model for a particular problem. The decision tree was then encoded into a set of rules which form the core of the knowledge-base of the ANNexpert. The Eclipse code for the expert system is shown in Appendix C. This code not only selects the appropriate ANN architecture, but will also select an appropriate software package to implement the network. Helpful tips are also provided by the expert system to aid the user in model development. The ANNexpert assumes that the user already has a problem configured into data patterns with inputs and outputs already defined.

After benchmark testing of 20 variations of ANNs, it was shown that 9 of the algorithms contained in the commercial software packages were particularly useful. By traversing the rule tree of Figure 3.4, the ANNexpert will decide which of these 9 ANNs is most appropriate for a specific problem. As can be seen from Figure 3.4, the decision process is quite complex. The key to the rule tree is to use *a priori* knowledge about the problem to decide which ANN to apply. For example, backpropagation performs well for continuous function mapping problems while PNNs and LVQ networks perform well for pattern classification problems.

The expert system starts out with a menu which asks the user which type of application the problem resembles. The user has six choices: *continuous_function_mapping*, *pattern_classification*, *time_series_analysis*, *visual_pattern_recognition*, *none_of_the_above*, and *more_information*. If the user selects *more_information*, then informative windows describing each type of application will be displayed followed by a repetition of the initial menu. If the user selects the other 5 options then one of the branches A through E of the decision tree will be traversed, as shown in Figure 3.4.

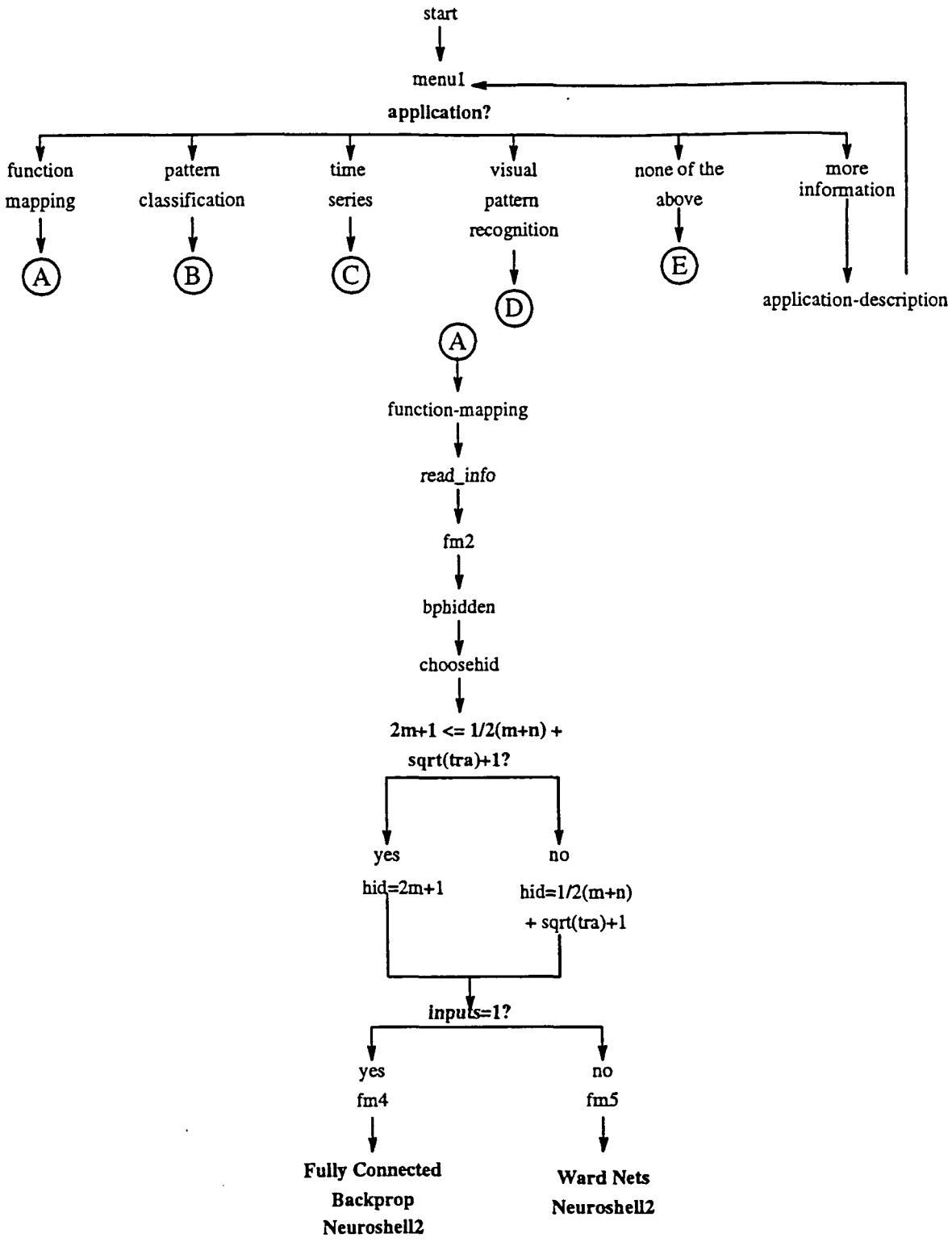

Figure 3.4: Expert System Rule Tree

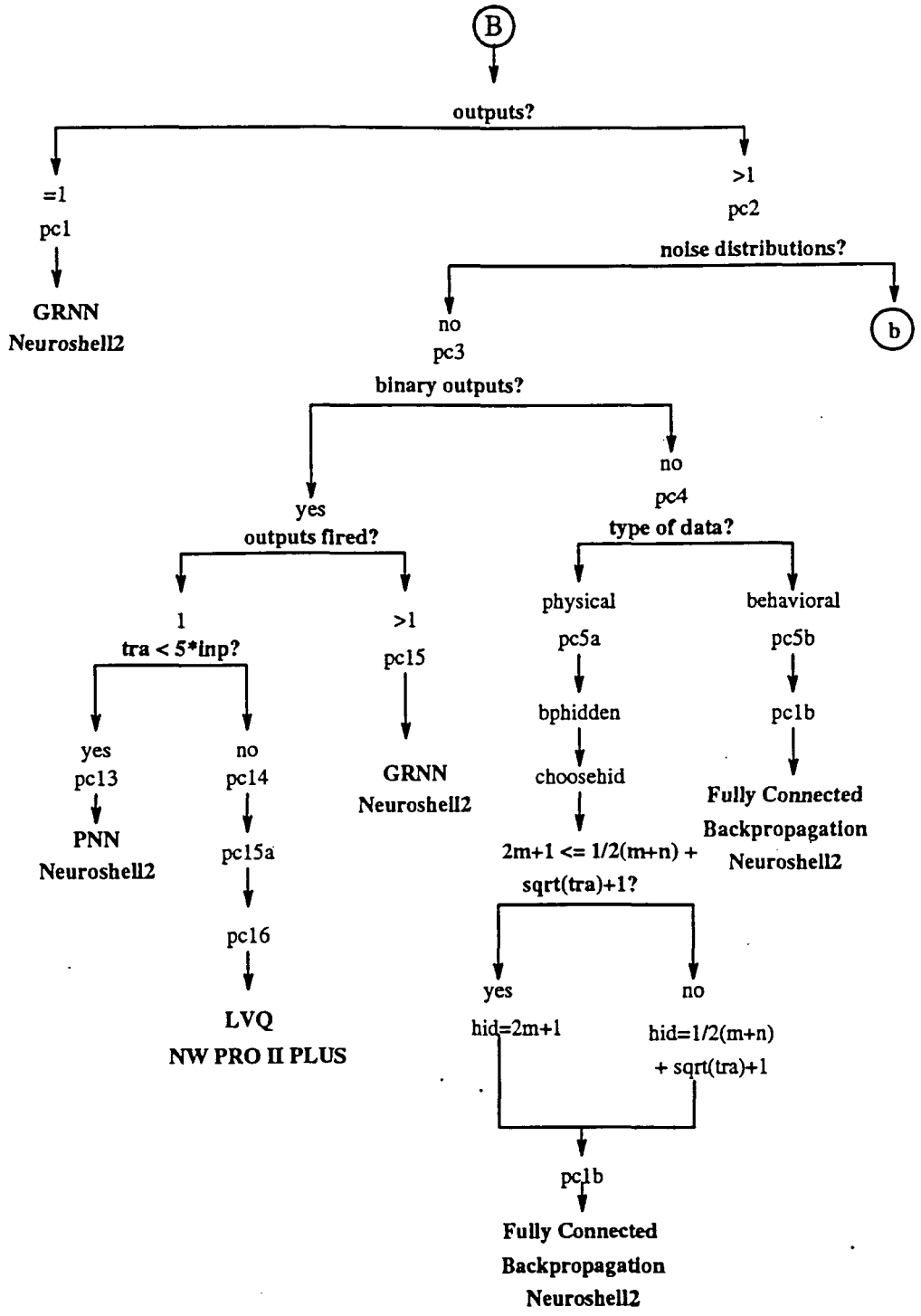

Figure 3.4: Expert System Rule Tree(continued)

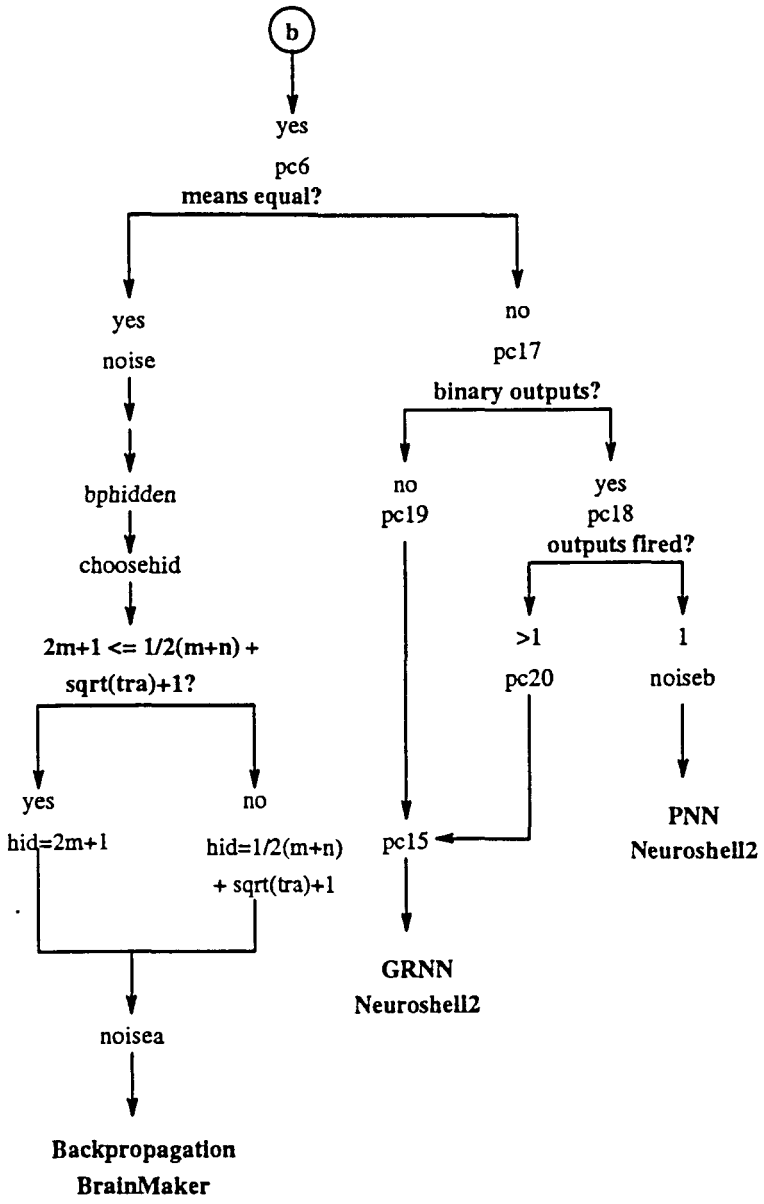

Figure 3.4: Expert System Rule Tree(continued)

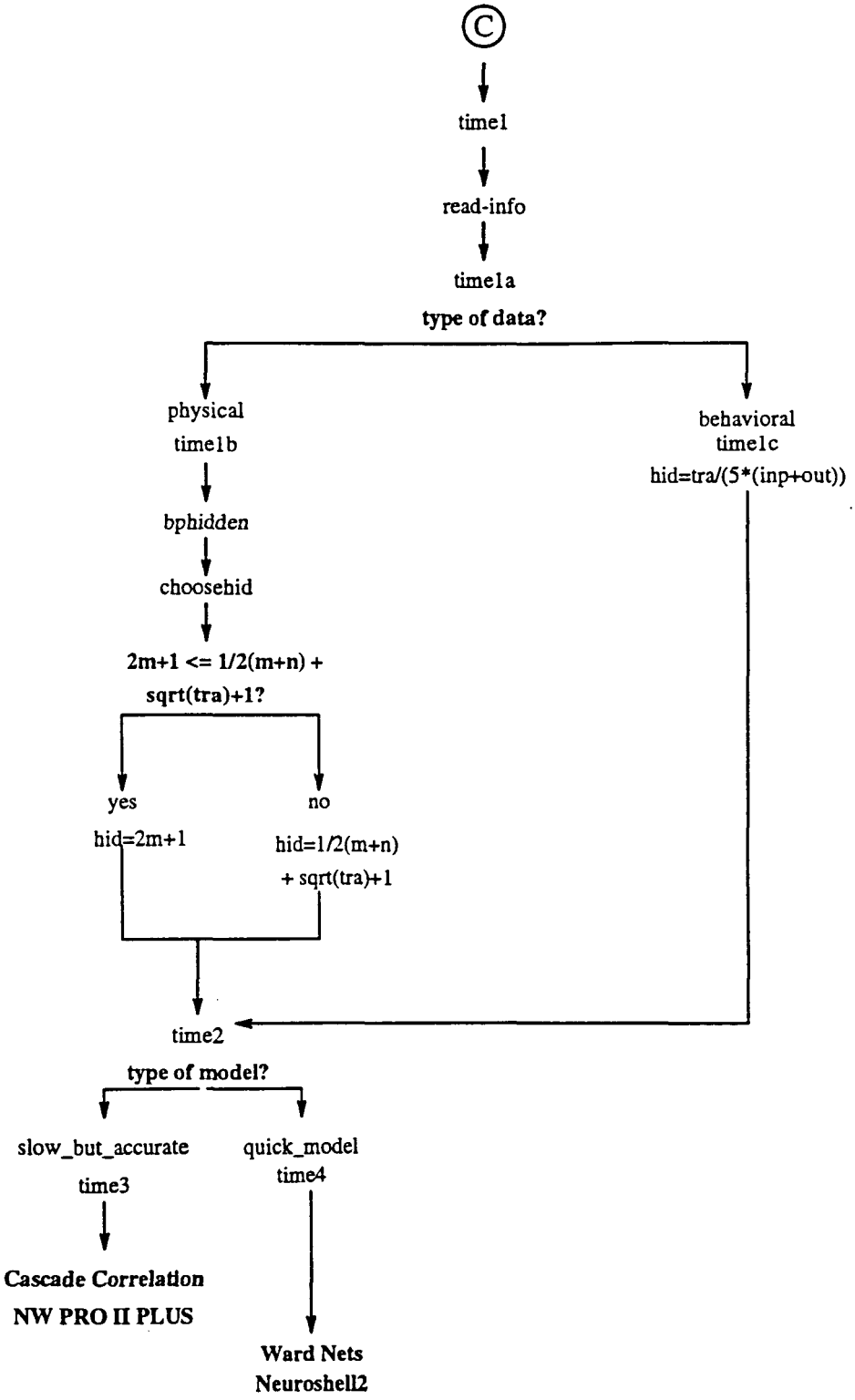

Figure 3.4: Expert System Rule Tree(continued)

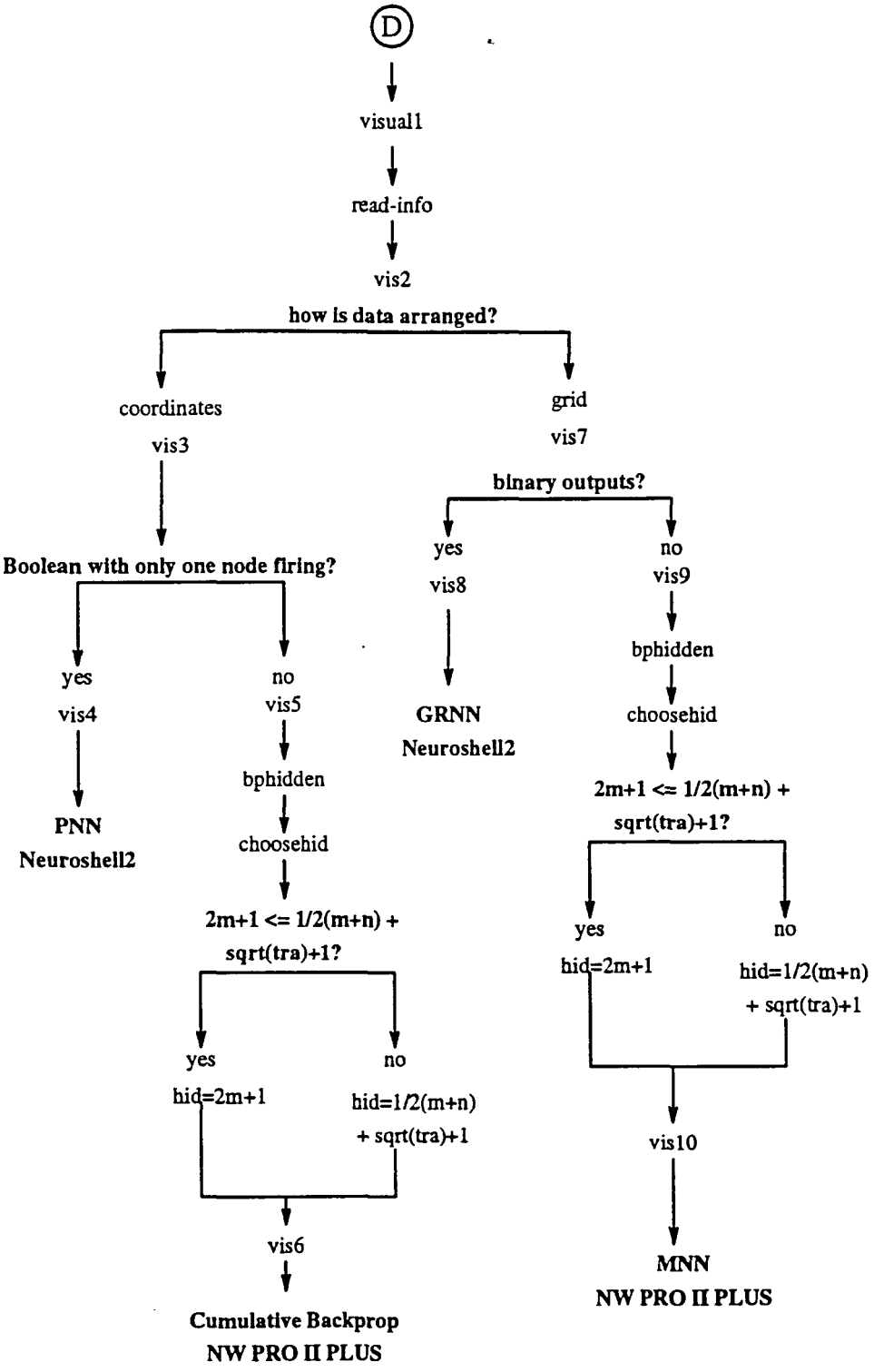

Figure 3.4: Expert System Rule Tree(continued)

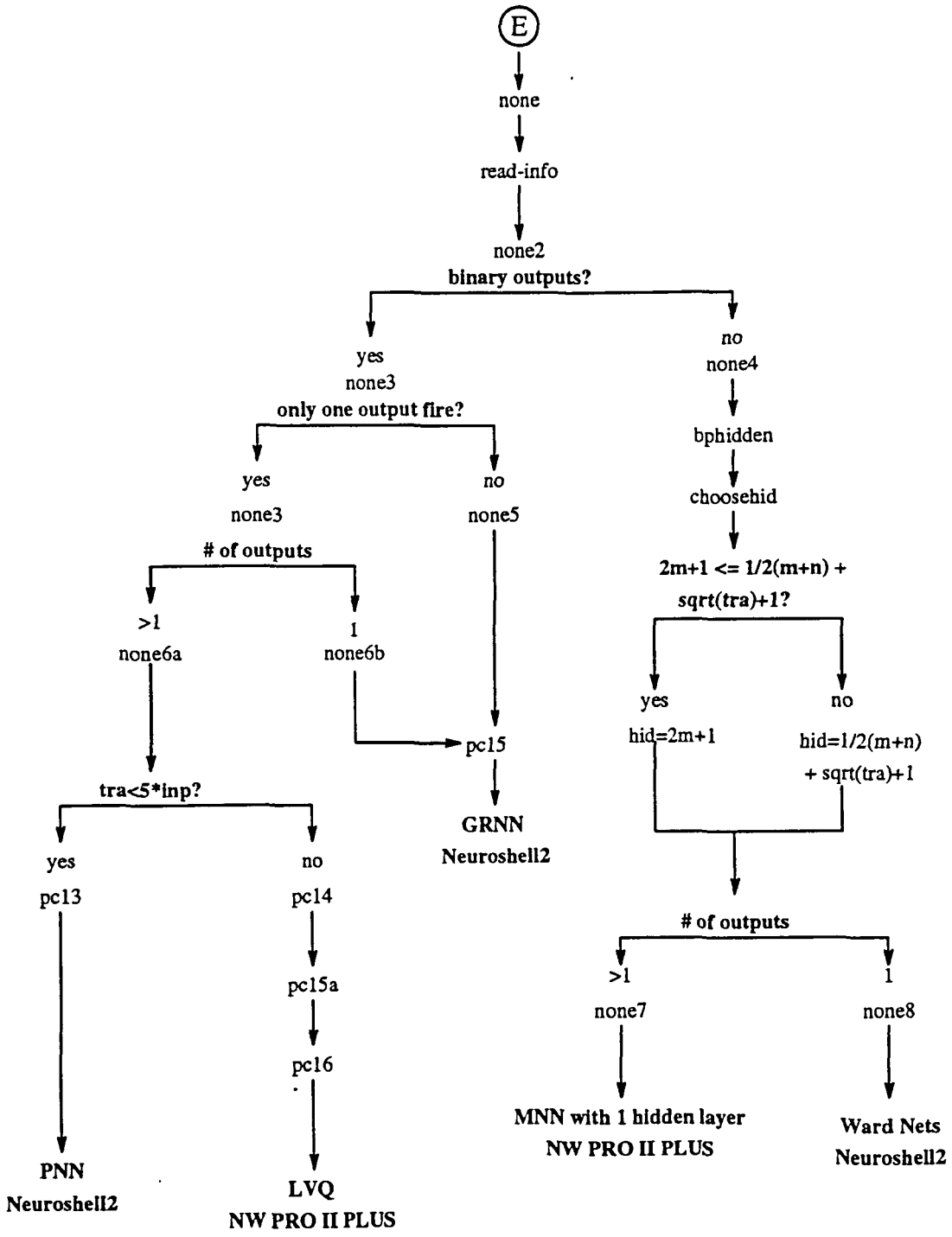

Figure 3.4: Expert System Rule Tree(continued)

If the user selects *continuous_function_mapping* as the application, then branch A will be traversed to determine the appropriate network. Next the user will be prompted for information on the number of inputs, outputs, and training examples in the training set. Since backpropagation networks performed well on the function mapping benchmarks such as the cosine and 2 input cosine problems, the ANNexpert will choose a backpropagation network to solve these types of problems. In order to choose an appropriate number of hidden nodes for the backpropagation network the expert system uses two rules, *bphidden* and *choosehid.* These rules select the number of hidden nodes by calculating the minimum of $2m+1$, where m is the number of hidden nodes, and the result of Equation 2.9. Before comparing the two values, 1 node is added to the result of Equation 2.9 to account for round-off error and to handle problems with extremely sparse data. Using these two rules, the ANNexpert will generate a reliable number of hidden nodes for most problems.

After choosing the number of hidden nodes, the ANNexpert will look at the number of inputs to decide which version of backpropagation is appropriate. If there is only one input, the expert system will choose the fully connected backpropagation algorithm implemented by the Neuroshell2 software package since this algorithm performed best on the single input cosine benchmark test. If there is more than one input, than the ANNexpert will choose the backpropagation algorithm with multiple transfer functions from Neuroshell2 since this network trained the fastest and performed well on the 2 input cosine problem.

Should the user of the ANNexpert select *pattern_classification* as the application, then branch B of the rule tree will be traversed. Again the user will be prompted for the number of inputs, outputs, and training examples. First the ANNexpert will look at the number of outputs for each data pattern. If the number of outputs is one, then the GRNN architecture will be chosen to solve the problem, since the PNN and LVQ network architectures used in this study were not applicable to problems with only one output. A

single output pattern classification problem would be similar to the exclusive-or benchmark problem, which the GRNN performed extremely well on.

If the user states that the problem has more than one output, than the expert system will ask the user if the application involves the classification of noise distributions. If the user replies "no" then the ANNexpert will ask if the outputs are binary. Certain networks such as the PNN and LVQ network perform best on data sets with multiple binary outputs in which only one output node has a positive output. Therefore, if the problem has multiple binary outputs in which more than one output fires, the GRNN algorithm will be chosen due to the inappropriateness of the LVQ network or PNN. If the outputs are binary with only one output having a positive value, then the number of training examples will be examined to determine if a PNN or LVQ network should be used. PNNs can be used well with sparse data, whereas LVQ networks cannot generalize well when testing vectors vary far from training vectors. Therefore, a PNN should be used in cases where data is sparse and an LVQ network should be used in cases where data is plentiful. A simple heuristic is used to determine whether the data is sparse or plentiful. If the number of training examples is less than 5 times the number of inputs, then the data is considered to be sparse. Otherwise, the data is considered plentiful. If the data is sparse then the PNN architecture will be selected whereas if data is plentiful the LVQ network will be chosen to solve the problem.

If the problem has multiple outputs, is not a noise distribution , and has continuous outputs then the ANNexpert will suggest a backpropagation network. This is done because backpropagation networks are good at solving problems with continuous outputs. The expert system will then ask the user about the nature of the data that the network is attempting to classify. As stated earlier, behavioral problems often contain large amounts of noise and thus need large amounts of data to train. For these problems, the number of hidden nodes in the network should be small compared to the number of training examples. Other problems use

physical data which does not generally contain as much noise so more hidden nodes can be used to help lower the network RMS error. The expert system will ask the user to state whether the training data is *behavioral* or *physical.* If the user selects behavioral, then the number of hidden nodes will be calculated using equation 2.8. If the user selects physical data, then the rules *bphidden* and *choosehid* will be utilized to determine the number of hidden nodes as outlined above.

After the number of hidden nodes is selected, the expert system will choose the fully connected backpropagation model, also known as a jump network. The selected software implementation is that found in Neuroshell2. The fully connected backpropagation model is chosen because it performed well on the exor, binary decoder, and iris pattern classification problems. The additional number of weighted connections found in the jump network also increases the classification power of the network.

The sub-branch of branch B deals with the classification of noise distributions similar to the noise classification benchmark test. When this branch is selected, the ANNexpert will first ask the user if the means of the distributions are equal. If the user answers "yes" then the rules *bphidden* and *choosehid* will be fired to determine an appropriate number of hidden units. The ANNexpert will then suggest the BrainMaker version of backpropagation to solve the problem since it had the best results on the benchmark problem.

If the user states that the means are not equal than the problem is much easier for an ANN and is trivial for a PNN to solve. The expert system will ask the user about the nature of the outputs to determine if a PNN is applicable. If a PNN is applicable, the ANNexpert will suggest it. Otherwise, the GRNN network will be the chosen architecture. In either case the suggested software will be Neuroshell2 since the GRNN and PNN implementations performed better on most of the benchmark problems than the versions provided in the NeuralWorks package.

Selecting *time_series_analysis* from the initial menu will cause branch C of Figure 3.4 to be traversed. Next, the user will be prompted for the number of inputs, outputs, and training patterns as above. As with the pattern classification branch, the user will be asked if the data is *behavioral* or *physical* and an appropriate number of hidden nodes will be chosen based on their response. Next the user is presented with a prompt requesting the type of model that is preferred, a *quick_model* or a *slow_but_accurate* model. The cascade correlation model is the best network for time series analysis according to the benchmark tests but it can have longer training times than other models. If the user states that a *slow_but_accurate* model is desired, then the cascade correlation network from NeuralWorks will be suggested. If the user desires a quick_model then the backpropagation model with multiple transfer functions will be chosen since this network also performed well on the time series benchmark test. The suggested software to implement this model will be the Neuroshell2 software package.

If the user selects *visual_pattern_classification* at the initial menu then branch D of Figure 3.4 will be used to determine which ANN to use to solve the problem. As with the other branches, the user will first be prompted for the number inputs, outputs, and training examples. Next the user, will be asked how the data is arranged, *grid* or *coordinates*. Visual pattern recognition problems are typically arranged in these two ways. With a grid system, the inputs to the network form an MxN grid, in which there are M*N input nodes. With a coordinate configuration, the inputs to the network will be the coordinates of a pixel in M-dimensional space, where M is the number of inputs. If the user answers *coordinates* when prompted then the problem is similar to the two spirals benchmark test. If the data is structured such that a PNN may be used to model it, then the ANNexpert will suggest the PNN architecture from Neuroshell2 since it was the only network which could solve the two spirals benchmark problem. If the PNN is not applicable, than a standard backpropagation

model with batch learning will be selected by the expert system since backpropagation has been shown to work well for visual pattern recognition in many applications. The suggested software is the norm-cum-delta backpropagation algorithm from NeuralWorks.

If the user stated that the data was arranged in a grid then the problem is similar to the character recognition benchmark problem. Next the user is asked if the outputs are binary. If the user answer "yes" than the ANNexpert will suggest a GRNN since it performed well on the character recognition benchmark. If the outputs are continuous than *bphidden* and *choosehid* are fired to determine the appropriate number of nodes for a modular neural network to solve this problem. A MNN was chosen because it also performed well on the character recognition problem and can be more powerful than backpropagation for complex problems. The recommended software for the MNN is NeuralWorks, while the recommended software for the GRNN is Neuroshell2.

The last major branch of the decision tree, branch E, is used when the user of the ANNexpert does not know what type of problem is trying to be modeled. Should the user choose *none_of_the_above* when prompted by the initial menu then branch E will be traversed and the user will be prompted for information on the number of inputs, outputs, and training examples. The next rule fired will ask the user if the outputs for the training patterns are binary. If the problem contains binary outputs, it is most likely similar to a pattern classification problem and should be solved with a PNN, GRNN, or LVQ network. These ANNs are particularly good at dividing training patterns into classes, but work best when the outputs are binary. If the user selects non-binary outputs, then a completely different branch of the decision tree will be traversed. If the outputs are continuous rather than binary then the problem is closer to a continuous function mapping problem or a time series analysis problem and should be solved by a network which is good at solving such problems, such as a backpropagation network.

If the user chooses binary outputs, the next rule to be fired will ask if only one output fires for each training pattern to determine if an LVQ network or PNN is applicable to the problem. If there is only one output then the ANNexpert will choose the GRNN network since the PNN and LVQ network architectures evaluated in this study were only applicable to problems with multiple outputs. If there are multiple outputs and only one output fires, then the expert system will choose a PNN or an LVQ network, depending on the amount of data provided in the training examples. If there is plenty of data, then an LVQ network will be used, since it works well when the data includes examples from nearly every possible training vector. If the data is sparse however, a PNN will be chosen since the PNN can be used effectively in situations where data is incomplete.

If the user answered "no" when asked if the problem had binary outputs then the ANNexpert will then use the rules *bphidden* and *choosehid* to determine the appropriate number of hidden units for a backpropagation network to solve the problem. If the problem has more than one output, then a MNN will be chosen since it is generally more powerful than standard backpropagation for solving complex problems. The MNN essentially acts like several backpropagation networks all trying to solve the problem together and will act like a single backpropagation network if the problem is simple. If the problem only has one continuous output, then it is most likely a function mapping or a time series problem. A Ward network is suggested by the ANNexpert to solve these problems since this network performed well on both of these types of benchmark tests.

CHAPTER 4: DISCUSSION OF RESULTS

Overview

The rules for the ANNexpert were incorporated into the expert system and were tested to insure that all the rules in the decision tree were firing correctly. The rules functioned correctly but there was still a need to test the capabilities of the ANNexpert to insure that the rules in the knowledge base were logical and applicable to actual real world problems. Toward this purpose, two examples from real world applications were selected and a model for each was developed based on the output from the ANNexpert. The first validation problem involves sonar pattern recognition while the second involves time series analysis of electric load forecasting. In addition to these real world problems, a third validation problem was generated which involves the function mapping of a 3-dimensional damped cosine function. These models were compared with alternative networks created by previous researchers and ANNs created using commercial software packages to determine if the ANNexpert models were better than a typical ANN model that a researcher might blindly use for a problem.

Sonar Pattern Recognition

Problem Description

The first validation problem involves the classification of sonar data by using an ANN. The data used was originally used by Gorman and Sejnowski in their work [15]. This is a pattern classification problem in which the neural network must classify the sonar signals into two groups, those bounced off a metal cylinder and those bounced off a cylindrical rock. The data set consists of 208 patterns which are divided in half to form a training set and a test set. Each pattern has 60 inputs and 2 outputs. The 60 inputs correspond to sonar signal inputs and the two outputs act as a flag to discriminate whether the signal is a metal cylinder or a rock. If the first output has a 1, than the object is a metal cylinder. If the second output is a 1, than the object is a rock. All the patterns in the data set are normalized between 0 and 1.

ANNexpert model

The characteristics of the sonar classification problem were input into the ANNexpert so that it could tell what type of ANN to use to model the problem. The number of inputs, number of outputs, number of training examples, and the type of problem (pattern classification) were prompted for by the system. The output of the ANNexpert was a report which gave a description of an ANN which could be used to solve the problem.

The recommendation of the ANNexpert was to use a probabilistic neural network to solve this problem. Furthermore, it specified that the number of input nodes should be 60 and that the number of output nodes should be 2. The ANNexpert suggested one hidden layer for this problem with 104 nodes. The expert system also recommended the Neuroshell2 software to implement the PNN and gave hints for developing, training, and testing the network. The network was trained following the instructions and then tested. Training the network took

only about 7 seconds since the PNN trains in only one pass of the data set. The default smoothing factor of .6 was used for this problem.

The network specified by the ANNexpert performed very well for this problem. After a few seconds of training, the network was able to correctly classify all 104 training patterns in the training set, corresponding to an RMS of zero. The default smoothing factor was again used for network testing. Again the network performed well, correctly classifying 94 of the 104 patterns which corresponds to 90% accuracy.

Other Models

When Gorman and Sejnowski ran their experiments on the sonar classification data [15], they used a slightly modified version of the backpropagation network. They used a learning rate of 2.0 and a momentum of 0.0. Initial weights were uniform random values from -0.3 to +0.3. They ran several different experiments in which they varied the number of hidden units, using a single layer with 0, 2, 3, 6, 12, and 24 units. Each network was trained by 300 epochs over the entire training set. The test results they reported varied from 73.1% to 90.4% accuracy on the test set depending on the number of hidden units used. The best model was found to have 12 hidden nodes. They performed further tests using a nearest neighbor classifier and achieved an 82.7% probability of correct classification. They also tested 3 trained humans on 100 signals randomly chosen from the 208 patterns. The accuracy of the human experts ranged from 88% to 97% correct.

A further test was run to determine how a typical backpropagation model would do on the sonar classification problem. This was done to compare the network created by using the ANNexpert with a typical network that someone might try without consulting the expert system. NeuralWorks Professional II Plus was chosen to implement this model since it

performed reasonably well on most of the benchmark tests and is one of the most well known neural network simulators available.

Using NeuralWorks, a backpropagation network was set up with hyperbolic tangent transfer functions and the norm-cum-delta learning rule. The network had 60 input nodes and 2 output nodes. The network had 1 hidden layer and 12 hidden nodes, the number determined to be best by Gorman and Sejnowski. The network was trained for 104,121 iterations, at which point the RMS bottomed out and reached a local minimum. Training took approximately 11.2 minutes and the network was able to learn the training set to an RMS error of .077. The network was then tested on the testing set to determine how well it learned to generalize with the information it learned from the training data. The RMS was .373, indicating that the network did not generalize well. The output from the network was analyzed to determine the classification accuracy of the network on the test data. The 40-20-40 rule was used to determine the classification of the output. With this designation, an output is considered a 0 if it is below 0.4, and a 1 if above 0.6. If either node had an output between 0.4 and 0.6 then the pattern was considered as a failed classification. When this was done, the network classified 84 patterns correctly and 20 patterns incorrectly, corresponding to an accuracy of 80.8% correct.

Electric Load Forecasting

Problem Description

The second validation problem used to test the ANNexpert involves the time series analysis of electric load data for the area around Omaha, Nebraska. The objective of the problem is to predict the next hourly electric load through the use of the past 6 hourly readings along with information on the month and day that the data comes from. The training

set consists of 3,941 examples of readings from the Omaha Public Power district taken during the winter months of 1990. The test set has 1,909 patterns also taken from the winter months of 1990. Each data pattern has 26 inputs and 1 output. The first 6 inputs correspond to the past 6 hourly observations of the electric load. The next 7 inputs are binary values which act as a flag to indicate the day of the week that the readings were taken. The 14th input is also a binary flag which indicates whether or not that day is a major holiday, since electric demand is lower on holidays. The remaining 12 inputs act as binary flags to indicate which month of the year the readings were taken. The single output for each training pattern is the next hourly electric load that the network is attempting to predict.

ANNexpert Model

The characteristics of the electric load forecasting problem were input into the expert system when prompted by the program. The type of problem(time series analysis), number of inputs(26), number of outputs(1), number of training patterns(3,941), and the nature of the data (behavioral, continuous) were input into the system. The resulting model suggested by the ANNexpert was the cascade correlation version of the backpropagation algorithm. The expert system stated that the network should have 26 inputs, 30 hidden nodes, and 1 output. Furthermore, it suggested that the NeuralWorks software package be used to implement the cascade correlation network. The network suggested by the expert system was created using NeuralWorks and was then trained and tested. The cascade correlation learning algorithm starts out with zero hidden nodes and trains several nodes in series and then inserts them into the network one at a time until a suitable RMS error is reached. The network trained for 2.7 hours before reaching an RMS error of .037. By this time, the network had added 5 nodes to the hidden layer. When tested the network did well, recalling the test data with an RMS error of .038, indicating that the network generalized well.

Other Model

As with the sonar data recognition problem, a typical backpropagation network was used to compare with the network created using the ANNexpert. This backpropagation model was intended to represent a model which a researcher might use without consulting the expert system. This model was also implemented using the NeuralWorks software package and featured norm-cum-delta learning with hyperbolic tangent transfer functions. The network had 26 inputs and 1 output. Using Kolmogorov's $2m+1$ formula, it was calculated that 53 hidden nodes should be used. This network was trained and tested using NeuralWorks. After training for approximately 9.8 hours, it was apparent that the network had reached a local minimum which corresponded to an RMS error of .056 on the training set. The network had an RMS error of .057 on the test set, showing that it learned the training set relatively well, but it did not learn as well or as fast as the ANNexpert network.

Continuous Function Mapping

Problem Description

The third validation problem consists of a continuous function mapping application with 2 inputs and 1 output. The output z of the function must be computed by the network based on the input values x and y. The function to be mapped represents a 3 dimensional damped cosine curve given by the following equation

$$z = (0.5 + 0.5 * \cos(\pi + 4\pi x)) * (0.5 + 0.5 * \cos(\pi + 4\pi y)) * (\exp -(x + y)) \quad (4.1)$$

The training data set consists of 100 random points in which the x and y values were uniformly distributed between 0 and 1. The test set contains 500 similarly generated points with 5% uniform noise added to each input. Both the training and testing set were normalized using a linear interpolation between .1 and .9.

ANNexpert Model

The model suggested by the ANNexpert was a Ward Net, a variation of backpropagation with a hidden layer composed of slabs of hidden layer nodes. Each slab in the hidden layer contains nodes with different transfer functions. The configuration specified by the ANNexpert was a 3 layer network with 2 input nodes and 1 output node. The hidden layer was formed of 2 slabs of 3 nodes each for a total of 6 hidden nodes. One slab used the Gaussian transfer function while the other was composed of nodes which use the complement of the Gaussian as a transfer function. Hints were given by the ANNexpert for training the network with the Neuroshell2 software package using the Netperfect training option.

The network specified by the ANNexpert performed well on the 3-dimensional damped cosine function mapping problem. The network was able to train down to an RMS of .048 in only 3.7 minutes. When the testing set was recalled, the RMS error was .090, which is respectable considering the complexity of the function and the amount of noise in the test set.

Other Model

A standard 3 layer backpropagation network was created with Neuroshell2 to compare with the model generated with the aid of the expert system. The network had 2 inputs and 1 output with a hidden layer of 5 nodes as specified by Kolmogorov's theorem. The hidden layer nodes used sigmoid transfer functions.

The standard backpropagation model performed poorly for this problem. After training for approximately 102.5 minutes the network had apparently fallen into a local minimum which corresponded to an RMS error of .108 on the training set. When tested it was apparent that the network did not learn the problem well, as the RMS error on the test set was only .142.

Results

The results of the sonar pattern recognition problem (see Table 4.1) as proposed by Gorman and Sejnowski [15] show how the ANNexpert can help a user to generate a good ANN model for a typical application. With an accuracy of 90%, the PNN suggested by the ANNexpert performed comparatively with the best model which was painstakingly developed by Gorman and Sejnowski, who are experts in using ANNs for pattern classification problems. They also had to use a specially modified algorithm and perform many experiments with different hidden nodes to obtain their best model. Furthermore, the PNN took only seconds to train compared to the considerable time it took for Gorman and Sejnowski to develop their best model.

Table 4.1: Sonar Pattern Classification Results

Model	Training Classification Rate	Testing Classification Rate
ANNexpert model	100.0%	90.0%
Backpropagation	100.0%	80.8%
Gorman & Sejnowski (best model)	99.8%	90.4%

The ANNexpert model also outperformed a typical backpropagation model from one of the best neural network simulators on the market with almost a 10% better classification rate. The ANNexpert network also trained 100 times faster than the NeuralWorks backpropagation model, offering a viable solution in seconds, rather than minutes.

The results from the electric load forecasting data (see Table 4.2), although not as dramatic, also show promise for the capabilities of the ANNexpert. The network proposed by the expert system had a training time which was 3 times faster than a typical backpropagation model which a user might use to solve the problem. Both models performed well on the test set, although the ANNexpert model did have a lower RMS error, .038, than the backpropagation model which had a testing RMS of .057.

Table 4.2: Electric Load Forecasting Results

Model	Training Time	Training RMS	Testing RMS
ANNexpert model	2:43:33	.037	.038
Backpropagation	9:48:17	.056	.057

On the 3-dimensional cosine continuous function mapping problem, the network which was generated and trained based on the advice of the ANNexpert also outperformed a standard backpropagation model (see Table 4.3). The standard model could not converge to an RMS below .05 after over 1.5 hours of training, while the ANNexpert model converged in less than 5 minutes. The ANNexpert model also had a substantially lower RMS on the test set.

Table 4.3: Function Mapping Results

Model	Training Time	Training RMS	Testing RMS
ANNexpert model	00:03:39	.048	.090
Backpropagation	1:42:48	.108	.142

These results show that it is feasible to incorporate knowledge regarding artificial neural networks into an expert system, and that by using this expert system a better ANN model can be developed. *A priori* knowledge of the problem is entered by the user of the system and this information then triggers a set of rules which make up the knowledge-base of the expert system. The expert system then produces an ANN model which correctly suits the given problem and is based on the advice of experts in the field. The results of the validation tests are also similar to the results of the benchmark tests, showing that the results of the benchmark tests can be extrapolated to real world problems.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

Conclusions

This work has shown that it is possible and beneficial to create a rule-based expert system to advise a user in the creation of an artificial neural network model to solve various types of problems. This expert system, ANNexpert, was created by incorporating the knowledge of experts in neural computation. The rules for this expert system were supplemented by running experiments on over 150 neural networks to determine which ANNs were best suited to various applications. The ANNexpert was then tested on a series of validation problems to insure that the system was producing correct results. These validation problems have shown that by using the ANNexpert, a better ANN model can be generated than by using a random ANN architecture.

While experimenting on the benchmark data sets several interesting results were discovered. First, it appeared that the backpropagation model did relatively well for most applications, therefore in situations of uncertainty it is probably best to use a backpropagation model. This may explain why backpropagation is currently the most widely used and publicized ANN architecture. Certain other architectures, such as PNNs and LVQ networks were shown to outperform backpropagation in problems which involved pattern classification or visual pattern recognition. The PNN in particular has training times which are orders of magnitude smaller than backpropagation. Another model which showed promise was the GRNN.

The ANNexpert was written using an expert system shell package known as Eclipse. The ANNexpert features a forward chaining structure of if/then rules which lead the user through all phases of model development and implementation. It features an easy to use graphical user interface as well as helpful informative tips for the novice user. It is not only a useful tool for the utilization of ANNs but can also be used as an educational tool.

Future Work

There are many avenues of future work which can be explored and implemented to continue the development of the ANNexpert. Further validation tests should be done to insure that the expert system produces good results for all types of problem. Currently the ANNexpert is aimed toward solving problems in four areas: pattern classification, time series analysis, visual pattern recognition, and continuous function mapping. Additional tests could be performed on problems in different areas, such as optimization or signal processing, to develop heuristics which could be implemented into the expert system to solve problems of specific interest to an industrial user, for example an electric power utility.

Another area of future work would be to explore even more ANN architectures to determine their capabilities. This information could be incorporated into the ANNexpert by the addition of rules to the knowledge-base. For example, various forms of unsupervised ANN models could be tested to determine rules for applying them to solve problems. Due to the modular nature of the expert system knowledge base, new rules could be added without too much further development.

Eventually the ANNexpert could grow and be linked to several types of neural networks automatically to form a hybrid system. Rather than selecting a network and

instructing the user on the setup of the network, the expert system could read in inputs from the user and automatically generate an ANN model for the problem and begin training it. This could be done by dynamically linking the expert system to several software packages or by imbedding several neural network codes into the expert system by creating user-defined subroutines in C. With these and other future changes the ANNexpert will be an even more vital tool for capturing the computational power of artificial neural networks.

BIBLIOGRAPHY

[1] N.Baba. "A New Approach for Finding the Global Minimum of Error Function of Neural Networks" Neural Networks vol 2 (1989): 367-373.

[2] P.T. Baffes. Nets User's Guide. Athens, Georgia: Cosmic, 1992.

[3] E.B. Bartlett. "A stochastic algorithm for artificial neural networks" Neurocomputing 6 (1994): 31-43.

[4] A. Ben-David, Y. Pao. "Self-Improving Expert Systems, An Architecture and Implementation" Information and Management 22 (November 1992): 323-331.

[5] G. Bylinsky. "Computers that Learn by Doing" Fortune (September 6, 1993): 96-102.

[6] M. Caudill. "Expert Networks" Byte (October, 1991): 108-116.

[7] M. Caudill. "Neural Networks Primer, Part 3" AI Expert 7 (June, 1988): 53-59.

[8] M. Caudill. "Neural Networks Primer, Part 4" AI Expert 7 (August, 1988): 53-59.

[9] J. Dayhoff. Neural Network Architectures. New York, New York: Van Nostrand Reinhold, 1990.

[10] C.W. Engel and M. Cran. "Pattern Classification A Neural Network Computes with Humans" PCAI (May/June, 1990): 20-23, 61.

[11] S.E. Fahlman. "An Empirical Study of Learning Speed in Back-Propagation Networks" CMU Technical Report, CMU-CS-88-162, June 1988.

[12] J.Y. Fan, M. Nikolaou, and R.E. White. "An Approach to Fault Diagnosis of Chemical Processes via Neural Networks" AIChE Journal 39 (January 1993): 82-88.

[13] R.A. Fisher. "The Use of Multiple Measurements in Taxonomic Problems" Annals of Eugenics vol 8, pt. 2 (1936): 179-188.

[14] W.J. Freeman. "The Physiology of the Perceptron." Scientific American (Feb, 1991): 78-85.

[15] R.P. Gorman and T.J. Sejnowski. "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" Neural Networks vol 1(1988): 75-89.

[16] The Haley Enterprise, Inc. Eclipse Reference Manual. Sewickley, Pennsylvania: The Haley Enterprise, Inc., 1992.

[17] R. Hecht-Nielsen. "Theory of the Backpropagation Neural Network." Proceedings of the International Joint Conference on Neural Networks, vol. 1, 1989: 593-605.

[18] J. Hertz, A. Krogh, and R.G. Palmer. Introduction to the Theory of Neural Computation. Redwood City, California: Addison-Wesley Publishing Company, 1991.

[19] D.V. Hillman. "Integrating Neural Nets and Expert Systems" AI Expert (June, 1990): 54- 59.

[20] J.J. Hopfield and D.W. Tank. "'Neural' Computation of Decisions in Optimization 2 Problems." Biological Cybernetics 52(1985): 141-152.

[21] J.C. Hoskins, K.M. Kaliyur, and D.M. Himmelblau. "Fault Diagnosis in Complex Chemical Plants Using Artificial Neural Networks" AIChE Journal 37 (January, 1991): 137-141.

[22] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. "Adaptive Mixtures of Local Experts" Neural Computation vol 3 (1991): 79-87.

[23] J. Lawrence and J. Fredrickson. BrainMaker Professional User's Guide and Reference Guide. Nevada City, California: California Scientific Software, 1993.

[24] R.P. Lippmann. "An Introduction to Computing with Neural Nets" IEEE Acoustics Speech and Signal Processing Magazine (April 1987): 4-22.

[25] G.F. Luger and W.A. Stubblefield. Artificial Intelligence Structures and Strategies for Complex Problem Solving, second edition. Redwood City, California: The Benjamin/Cummings Publishing Company, Inc., 1993.

[26] A. Maren, C. Harston, R. Pap. Handbook of Neural Computing Applications. San Diego, California: Academic Press, Inc., 1990.

[27] S. Mehta and L. Fulop. "An Analog Neural Network to Solve the Hamiltonian Cycle Problem" Neural Networks 6(1993): 869-881.

[28] A.A. Minai and R.D. Williams. "On the Derivatives of the Sigmoid." Neural Networks 6(1993): 845-853.

[29] NeuralWare. Neural Computing, A Technology Handbook for Professional II/Plusand NeuralWorks Explorer. Pittsburgh, Pennsylvania: NeuralWare, Inc., 1993.

[30] NeuralWare. Using NeuralWorks, A Tutorial for NeuralWorks Professional II/Plus and NeuralWorks Explorer. Pittsburgh, Pennsylvania: NeuralWare, Inc., 1993.

[31] NeuroDynamX. DynaMind Developer User's Guide. Boulder, Colorado: NeuroDynamX, Inc., 1994.

[32] NeuroDynamX. DynaMind User's Guide. Boulder, Colorado: NeuroDynamX, Inc., 1994.

[33] Y. Ohga and H. Seki. "Abnormal Event Identification in Nuclear Power Plants Using A Neural Network and Knowledge Processing" Nuclear Technology 101 (February 1993): 159-167.

[34] M.O. Poliac, E.B. Lee, J.R. Slagle, and M.R. Wick. "A Crew Scheduling Problem." Proceedings of the IEEE International Conference on Neural Networks vol. 4, (1987): 779-786.

[35] M. Puttre. "Neural Networks Provide Solutions to Nonlinear Problems." Mechanical Engineering (Oct, 1993): 69-72.

[36] A.K. Ray "Equipment Fault Diagnosis--A Neural Network Approach" Computers in Industry 16 (1991): 169-177.

[37] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. "Learning Internal Representations by Error Propagation" Parallel Distributed Processing vol 1. Cambridge MA: M.I.T. Press, 1986.

[38] D.F. Specht. "Probabilistic Neural Networks" Neural Networks 3(1990): 109-118.

[39] D.F. Specht. "A General Regression Neural Network" IEEE Transactions on Neural Networks vol 2, no.6, November 1991: 568-576.

[40] M.L. Tseng. Integrating Neural Networks with Influence Diagrams for Multiple Sensor Diagnostic Systems, Ph.D. Dissertation, University of California at Berkeley, August 1991.

[41] V. Venkatasubramanian and K. Chan. "A Neural Network Approach for Process Fault Diagnosis" AIChE Journal 35 (December 1989): 1993-2002.

[42] Ward Systems Group. Neuroshell 2. Frederick, Maryland: Ward Systems Group, Inc., 1993.

[43] K. Watanabe, I. Matsuura, M. Abe, and M. Kubota. "Incipient Fault Diagnosis of Chemical Processes via Artificial Neural Networks" AIChE Journal 35 (November 1989): 1803-1812.

[44] B. Widrow and M.A. Lehr. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation." Proceedings of the IEEE 78.9 (Sept, 1990): 1415-1441.

[45] X. Xu and W.T. Tsai. "Effective Neural Algorithms for the Traveling Salesman Problem." Neural Networks 2(1991): 193-205.

[46] A.N. Zhou, V. Cherkassky, T.R. Baldwin, and D.E. Olson. "A Neural Network Approach to Job-Shop Scheduling." IEEE Transactions on Neural Networks vol. 2, No. 1 (Jan, 1991): 175-179.

APPENDIX A. BENCHMARK TEST RESULTS

The following is a spreadsheet showing the results of all the benchmark tests which were used to supplement the rules for the expert system.

TableA.1: Results of Benchmark Tests

				Training		Test	
Bench Mark	Software	Model	Hidden	RMS	Time(min)	RMS	Test/Training
exor	NW Pro II +	BP GD	5	0.038	0.63	0.045	1.18
	NW Pro II +	BP NCD Tanh	5	0.033	.07	0.039	1.18
	NW Pro II +	CC	2	0.034	2.33	0.063	1.85
	NW Pro II +	DRS	5	0.045	.13	0.055	1.22
	NW Pro II +	GRNN	4	0.042	0.02	0.044	1.05
	NW Pro II +	LVQ	N/A	----------			
	NW Pro II +	MNN	5	0.039	0.03	0.042	1.08
	NW Pro II +	PNN	N/A	----------			
	NW Pro II +	RBF	5	0.031	0.23	0.040	1.29
	NW Pro II +	SOM	5	0.039	2.00	0.038	0.97
	Neuroshell2	BP	4	0.050	.97	0.054	1.08
	Neuroshell2	MH BP	2(x2)	0.055	.08	0.060	1.09
	Neuroshell2	FC BP	4	0.053	.58	0.045	0.85
	Neuroshell2	R BP	4	0.044	2.75	0.050	1.14
	Neuroshell2	GRNN	4	0.006	.02	0.008	1.33
	Neuroshell2	PNN	N/A	----------			
	Nasa Nets	BP	5	0.046	.03	0.051	1.10
	BrianMaker	BP	5	0.037	.68	0.038	1.03

Table A.1: Results of Benchmark Tests(Cont.)

	NeuroDynamX	BP	5	0.038	1.12	0.065	1.71
	NeuroDynamX	Madaline III	5	0.045	8.55	0.050	1.11
bdec	NW Pro II +	BP GD	7	0.042	1.45	0.046	1.10
	NW Pro II +	BP NCD Tanh	7	0.043	0.65	0.047	1.09
	NW Pro II +	CC	7 (does not converge)----------				
	NW Pro II +	DRS	7 (does not converge)----------				
	NW Pro II +	GRNN	8	0.038	0.03	0.039	1.03
	NW Pro II +	LVQ	8	0.000	0.08	0.000	
	NW Pro II +	MNN	7	0.040	0.10	0.042	1.05
	NW Pro II +	PNN	8	0.000	0.02	0.000	
	NW Pro II +	RBF	7 (does not converge)----------				
	NW Pro II +	SOM	7	0.051	0.58	0.053	1.04
	Neuroshell2	BP	8	0.042	.52	0.044	1.05
	Neuroshell2	MH BP	4(x2)	0.055	.18	0.056	1.02
	Neuroshell2	FC BP	8	0.041	1.15	0.045	1.10
	Neuroshell2	R BP	8	0.037	3.83	0.040	1.08
	Neuroshell2	GRNN	8	0.004	.02	0.005	1.25
	Neuroshell2	PNN	8	0.000	.02	0.000	
	Nasa Nets	BP	7 (does not converge)----------				
	BrianMaker	BP	7	0.053	.18	0.055	1.04
	NeuroDynamX	BP	7	0.051	29.72	0.053	1.04
	NeuroDynamX	Madaline III	7	0.047	69.90	0.049	1.04
cos	NW Pro II +	BP GD	3 (does not converge)----------				
	NW Pro II +	BP NCD Tanh	3	0.036	0.20	0.073	2.03
	NW Pro II +	CC	3 (does not converge)----------				
	NW Pro II +	DRS	3	0.038	0.13	0.072	1.89
	NW Pro II +	GRNN	50	0.035	0.07	0.070	2.00
	NW Pro II +	LVQ	N/A----------				
	NW Pro II +	MNN	3	0.041	0.32	0.073	1.78
	NW Pro II +	PNN	N/A----------				
	NW Pro II +	RBF	3	0.037	0.32	0.074	2.00
	NW Pro II +	SOM	N/A----------				
	Neuroshell2	BP	8	0.047	1.28	0.075	1.60
	Neuroshell2	MH BP	4(x2)	0.041	.10	0.072	1.76
	Neuroshell2	FC BP	8	0.053	1.08	0.060	1.13

Table A.1: Results of Benchmark Tests(Cont.)

	Neuroshell2	R BP	8	0.044	2.10	0.072	1.64
	Neuroshell2	GRNN	50 (does not converge)				
	Neuroshell2	PNN	N/A				
	Nasa Nets	BP	3	0.050	.27	0.067	1.34
	BrianMaker	BP	3	0.032	.27	0.068	2.13
	NeuroDynamX	BP	3	0.044	5.25	0.074	1.68
	NeuroDynamX	Madaline III	3	0.047	9.93	0.078	1.66
2cos	NW Pro II +	BP GD	5 (does not converge)				
	NW Pro II +	BP NCD Tanh	5	0.030	5.51	0.059	1.97
	NW Pro II +	CC	5 (does not converge)				
	NW Pro II +	DRS	5	0.050	2.95	0.073	1.46
	NW Pro II +	GRNN	50 (does not converge)				
	NW Pro II +	LVQ	N/A				
	NW Pro II +	MNN	5	0.045	121.00	0.089	1.98
	NW Pro II +	PNN	50 (does not converge)				
	NW Pro II +	RBF	5 (does not converge)				
	NW Pro II +	SOM	5	0.046	0.88	0.084	1.83
	Neuroshell2	BP	9	0.053	3.52	0.093	1.75
	Neuroshell2	MH BP	4(x2)	0.046	.13	0.067	1.46
	Neuroshell2	FC BP	9	0.054	2.43	0.082	1.52
	Neuroshell2	R BP	9	0.040	4.33	0.067	1.68
	Neuroshell2	GRNN	50 (does not converge)				
	Neuroshell2	PNN	N/A				
	Nasa Nets	BP	5	0.050	.55	0.063	1.26
	BrianMaker	BP	5	0.032	.67	0.083	2.59
	NeuroDynamX	BP	5	0.046	22.38	0.065	1.41
	NeuroDynamX	Madaline III	5	0.047	23.65	0.083	1.77
char	NW Pro II +	BP GD	30	0.041	1.62	0.052	1.27
	NW Pro II +	BP NCD Tanh	30	0.036	0.67	0.039	1.08
	NW Pro II +	CC	30 (does not converge)				
	NW Pro II +	DRS	30 (does not converge)				
	NW Pro II +	GRNN	26	0.006	0.07	0.006	1.00

Table A.1: Results of Benchmark Tests(Cont.)

	NW Pro II +	LVQ	N/A----------				
	NW Pro II +	MNN		30	0.037	36.00	0.039	1.05
	NW Pro II +	PNN	N/A----------				
	NW Pro II +	RBF		30 (does not converge)----------			
	NW Pro II +	SOM		30 (does not converge)----------			
	Neuroshell2	BP		34	0.034	2.3	0.039	1.15
	Neuroshell2	MH BP		17(x2)	0.039	.83	0.042	1.08
	Neuroshell2	FC BP		34	0.034	2.33	0.045	1.32
	Neuroshell2	R BP		34	0.032	8.88	0.041	1.28
	Neuroshell2	GRNN		51	0.000	.02	0.000	
	Neuroshell2	PNN	N/A----------				
	Nasa Nets	BP		30	0.046	2.78	0.053	1.15
	BrianMaker	BP		30	0.030	.42	0.034	1.13
	NeuroDynamX	BP		30	0.045	12.65	0.052	1.16
	NeuroDynamX	Madaline III		30	0.046	11.33	0.052	1.13
spir	NW Pro II +	BP GD	10	10	(does not converge)----------			
	NW Pro II +	BP NCD Tanh	10	10	(does not converge)----------			
	NW Pro II +	CC	10	10	(does not converge)----------			
	NW Pro II +	DRS	10	10	(does not converge)----------			
	NW Pro II +	GRNN		194	(does not converge)----------			
	NW Pro II +	LVQ		19	(does not converge)----------			
	NW Pro II +	MNN	10	10	(does not converge)----------			
	NW Pro II +	PNN		194	(does not converge)----------			
	NW Pro II +	RBF	10	10	(does not converge)----------			
	NW Pro II +	SOM	10	10	(does not converge)----------			
	Neuroshell2	BP	8	8	(does not converge)----------			
	Neuroshell2	MH BP	8(x2)		(does not converge)----------			
	Neuroshell2	FC BP	8	8	(does not converge)----------			
	Neuroshell2	R BP		16	(does not converge)----------			
	Neuroshell2	GRNN		199	(does not converge)----------			
	Neuroshell2	PNN		194	0.000	.02	0.070	
	Nasa Nets	BP	10	10	(does not converge)----------			
	BrianMaker	BP	10	10	(does not converge)----------			
	NeuroDynamX	BP	10	10	(does not converge)----------			
	NeuroDynamX	Madaline III	10	10	(does not converge)----------			

Table A.1: Results of Benchmark Tests(Cont.)

irs	NW Pro II +	BP GD	10	0.054	3.02	0.102	1.89
	NW Pro II +	BP NCD Tanh	10	0.032	0.33	0.091	2.84
	NW Pro II +	CC	10 (does not converge)----------				
	NW Pro II +	DRS	10 (does not converge)----------				
	NW Pro II +	GRNN	10	0.002	0.05	0.161	66.26
	NW Pro II +	LVQ	10	0.000	0.50	0.092	
	NW Pro II +	MNN	10	0.040	46.00	0.095	2.38
	NW Pro II +	PNN	100 (does not converge)----------				
	NW Pro II +	RBF	10 (does not converge)----------				
	NW Pro II +	SOM	10 (does not converge)----------				
	Neuroshell2	BP	27	0.032	1.8	0.087	2.72
	Neuroshell2	MH BP	14(x2)	0.030	.27	0.094	3.13
	Neuroshell2	FC BP	27	0.050	2.28	0.094	1.88
	Neuroshell2	R BP	27	0.030	.25	0.094	3.13
	Neuroshell2	GRNN	100	0.000	.02	0.136	
	Neuroshell2	PNN	100	0.000	.02	0.115	
	Nasa Nets	BP	10 (does not converge)----------				
	BrianMaker	BP	10	0.030	.25	0.100	3.33
	NeuroDynamX	BP	10	0.039	21.70	0.124	
	NeuroDynamX	Madaline III	10 (does not converge)----------				
wthr	NW Pro II +	BP GD	11	0.043	0.72	0.035	0.81
	NW Pro II +	BP NCD Tanh	11	0.054	0.43	0.052	0.96
	NW Pro II +	CC	11	0.038	4.97	0.027	0.71
	NW Pro II +	DRS	11	0.042	28.50	0.031	0.74
	NW Pro II +	GRNN	21 (does not converge)----------				
	NW Pro II +	LVQ	N/A----------				
	NW Pro II +	MNN	11	0.040	1.05	0.030	0.75
	NW Pro II +	PNN	N/A----------				
	NW Pro II +	RBF	11	0.048	2.50	0.036	0.75
	NW Pro II +	SOM	11 (does not converge)----------				
	Neuroshell2	BP	23	0.044	.52	0.037	0.84
	Neuroshell2	MH BP	12(x2)	0.044	.2	0.032	0.73
	Neuroshell2	FC BP	23	0.044	.17	0.035	0.80
	Neuroshell2	R BP	23	0.046	.65	0.047	1.02
	Neuroshell2	GRNN	400 (does not converge)----------				

Table A.1: Results of Benchmark Tests(Cont.)

	Neuroshell2	PNN	N/A----------				
	Nasa Nets	BP	11	0.035	.53	0.040	1.14
	BrianMaker	BP	11	0.037	4.6	0.035	0.95
	NeuroDynamX	BP	11	0.043	2.62	0.032	0.74
	NeuroDynamX	Madaline III	11	0.043	2.67	0.033	0.77
noiz	NW Pro II +	BP GD	21 (does not converge)----------				
	NW Pro II +	BP NCD Tanh	21 (does not converge)----------				
	NW Pro II +	CC	21 (does not converge)----------				
	NW Pro II +	DRS	21	0.051	497.00	0.738	14.47
	NW Pro II +	GRNN	21 (does not converge)----------				
	NW Pro II +	LVQ	21 (does not converge)----------				
	NW Pro II +	MNN	21 (does not converge)----------				
	NW Pro II +	PNN	21 (does not converge)----------				
	NW Pro II +	RBF	21 (does not converge)----------				
	NW Pro II +	SOM	21 (does not converge)----------				
	Neuroshell2	BP	26 (does not converge)----------				
	Neuroshell2	MH BP	10(x2) (does not converge)----------				
	Neuroshell2	FC BP	19 (does not converge)----------				
	Neuroshell2	R BP	19 (does not converge)----------				
	Neuroshell2	GRNN	200 (does not converge)----------				
	Neuroshell2	PNN	200 (does not converge)----------				
	Nasa Nets	BP	21 (does not converge)----------				
	BrianMaker	BP	21	0.038	1.70	0.188	4.95
	NeuroDynamX	BP	21	0.055	711.00	0.269	4.89
	NeuroDynamX	Madaline III	21 (does not converge)----------				

BP GD : Back Propagation with Generalized Delta Learning and Sigmoid Transfer Functions
BP NCD Tanh : Back Progation with Norm-Cumulative-Delta Learning and Tanh Transfer Functions
CC : Cascade Correlation with Norm-Cum-Delta and Tanh Transfer Functions
DRS : Directed Random Search with Tanh Transfer Functions
GRNN : General Regression Neural Network
LVQ : Learning Vector Quantization
MNN : Modular Neural Network

Table A.1: Results of Benchmark Tests(Cont.)

PNN	: Probabalistic Neural Network
RBF	: Radial Basis Function Network
SOM	: Self Organizing Map
BP	: Standard Back Propagation
MH BP	: BP w/ Multiple hidden layers with different activation functions
FC BP	: Fully Connected BP
R BP	: Recurrent BP w/ dampened feedback
GRNN	: General Regression Neural Network
PNN	: Probabilistic Neural Network
Madaline III	: Back Propagated error using the Madaline 3 training algorithm
exor	: the exclusive or problem
bdec	: a 3 to 8 binary decoder problem
cos	: continuous function mapping of a cosine function
2cos	: continuous function mapping of a 2 input cosine hump
char	: character recognition
spir	: pattern recognition of intertwined spirals
irs	: pattern classification of iris flowers
wthr	: time series weather analysis problem
noiz	: noise classification problem

APPENDIX B. ANNEXPERT CODE

The following is the Eclipse code which was used to generate the ANNExpert.

```
;=====================================================
;********ANN EXPERT:  Artificial Neural Network Expert System
;********Matthew L. Reid
;********Nuclear Engineering Department
;********Iowa State University
;********Copyright © Matthew L. Reid, 1994.  All rights reserved.
;=====================================================


(defrelation rule2 (?))
(defrelation menu (?))
(defrelation application (?))
(defrelation inputs (?))
(defrelation outputs(?))
(defrelation fm (?))
(defrelation examples (?))
(defrelation hid1 (?))
(defrelation hid2 (?))
(defrelation hid3 (?))
(defrelation path (?))
(defrelation path2 (?))
(defrelation hid (?))
 (defrelation bp (?))
(defrelation path1 (?))
(defrelation outtype (?))
(defrelation outtype2 (?))
(defrelation outfired (?))
(defrelation behave (?))
(defrelation noise (?))
(defrelation stat (?))
(defrelation stat2 (?))
(defrelation choice (?))
(defrelation visual (?))
(defrelation vout (?))
(defrelation vout2 (?))
```

```
(defrule start
        (initial-fact)
=>
(notify "Welcome to the A.N.N. Expert, a product of the Adaptive
Computing Laboratory of Iowa State University.")
(notify "Neural networks can be used to solve a large number of problems
in industry. These problems usually fall into one of the following
categories:")
(assert (menu start)))

(defrule application-description
        (application more_information)
=>
(notify "CONTINUOUS FUNCTION MAPPING problems usually consist of finding the mathematical
relationship between an output variable and one or several input variables. Examples would be trying to
predict a process output based on input variables.")
(notify "PATTERN CLASSIFICATION problems assign an input pattern to one of several different output
categories. Examples include credit risk assessment, fault diagnosis, Boolean logic functions and binary
decoders." )
(notify "TIME SERIES ANALYSIS problems attempt to predict a future event based on recent history.
Examples include stock market prediction, and demand forecasting.")
(notify "VISUAL PATTERN RECOGNITION These problems attempt to classify patterns into groups based
on visual features.")
 (assert (menu again)))

(defrule menu1
        (menu start)
=>
(assert (application =(select "Which of the following does your application resemble?"
continuous_function_mapping pattern_classification time_series_analysis visual_pattern_recognition
none_of_the_above more_information))))

(defrule menu2
        (menu again)
=>
(assert (application =(select "Which of the following does your application resemble?"
continuous_function_mapping pattern_classification time_series_analysis visual_pattern_recognition
none_of_the_above ))))

(defrule function-mapping
        (application continuous_function_mapping)
=>
(printout t "You have selected a continuous function mapping problem"  crlf )
(printout t "These types of problems typically contain one or many inputs and
try to model the behavior of a single output variable. "crlf crlf crlf)
(assert (path fm)))

(defrule read-info
        (path ?)
=>
(assert (inputs =(ask "How many input variables are there in your data set?")))
```

```
(assert (outputs =(ask "How many output variables are there in your data set?")))
(assert (examples =(ask "How many training examples are there in your data set?"))))

(defrule fm2
        (path fm)
        (outputs ?outp)
=>
(if (> ?outp 1)
then (notify "If a function mapping has more than one output variable, it is often easier to set up multiple
networks, with each network responsible for learning 1 output"))
(assert (bp hidden)))

(defrule bphidden
        (bp hidden)
        (outputs ?out)
        (inputs ?inp)
        (examples ?tra)
=>
(assert (hid1 =(+ (* 2 ?inp) +1)))
(assert (hid2 =(+ (/ (+ ?inp ?out) 2) (+ 1 (sqrt ?tra))))))

(defrule choosehid
        (hid1 ?hid1)
        (hid2 ?hid2)
=>
(if (<= ?hid1 ?hid2)
then (assert (hid ?hid1))
else (assert (hid ?hid2))))

(defrule fm4
        (path fm)
        (inputs ?inp)
        (outputs ?outp)
        (hid ?hid)
        (test (= ?inp 1))
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Fully Connected Back Propagation neural network
with 1 hidden layer" crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?hid )
(format t  "The output layer should have %d nodes %n" ?outp )
 (printout t "The recommended software for this type of problem is the
fully connected backpropagation architecture provided by Neuroshell2
from Ward Systems Group, Inc." crlf ))

(defrule fm5
        (path fm)
        (inputs ?inp)
```

```
        (outputs ?outp)
        (hid ?hid)
        (test (!= ?inp 1))
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Back Propagation neural network with
2 slabs of hidden nodes with different transfer functions." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "Each hidden layer slab should have  %d nodes %n" (+ 1 (/ ?hid 2)))
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended architecture for this problem
is the Ward Net, provided by Neuroshell2, available
from Ward Systems Group, Inc." crlf ))

(defrule pattern-classification
        (application pattern_classification)
=>
        (printout t "You have selected a pattern classification problem" crlf)
        (assert (path pc)))

(defrule pc1
        (path pc)
        (examples ?tra)
        (inputs ?inp)
        (outputs ?out)
        (test (= ?out 1))
=>
(notify "Most pattern classification problems are set up with multiple outputs, with each output node
representing a class or a code to classify a class")
(assert (path1 pc1))
(assert (bp hidden)))

(defrule pc1b
        (path1 pc1)
        (hid ?hid)
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
=>
 (printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a General Regression Neural Network with 1 hidden
layer" crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?tra )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "This architecture is provided in both the NeuralWare
Professional II Plus software and in Neuroshell2" crlf)
```

```
(printout t "The recommended software for this type of problem is the
General Regression Neural Network provided by Neuroshell2
from Ward Systems Group, Inc." crlf ))

(defrule pc2
        (path pc)
        (outputs ?out)
        (test (> ?out 1))
=>
 (notify "Sometimes, ANN's are used to classify which of several statistical distributions a set of data was
generated from")
 (assert (noise =(no "Are you trying to determine which statistical distribution that data patterns belong to?"))))

(defrule pc3
        (path pc)
        (noise 1)
=>
        (notify "Most pattern classification problems work best with binary outputs, in which each output
node corresponds to a distinct class and only one output has a positive value for each pattern.")
(assert (outtype =(yes "Are the outputs binary"))))

(defrule pc4
        (path pc)
        (outtype 0)
=>
(notify "Often, ANN's are used to classify behavioral data, which involves people and therefore contains large
amounts of noise.  Examples are stock market prediction and credit risk assessment.")
(notify "Other types of classification involve physical data, where a functional relationship is involved")
(assert (behave =(select "What is the nature of your data?" physical behavioral))))

(defrule pc5a
        (behave physical)
        (path pc)
=>
(assert (bp hidden))
(assert (path1 pc10)))

(defrule pc5b
        (path pc)
        (behave behavioral)
        (inputs ?inp)
        (outputs ?out)
        (examples ?tra)
=>
(assert (hid =(+ 1 (/ ?tra (* 5 (+ ?inp ?out))))))
(assert (path1 pc10)))

(defrule pc10
        (path1 pc10)
        (hid ?hid)
        (inputs ?inp)
```

```
        (outputs ?outp)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Fully Connected Back Propagation Network
with 1 hidden layer" crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?hid )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "This architecture is provided in both the NeuralWare
Professional II Plus software and in Neuroshell2" crlf)
(printout t "The recommended software is the Fully Connected Back Prop
 provided by Neuroshell2 from Ward Systems Group, Inc." crlf ))

(defrule pc6
        (noise 0)
=>
(assert (stat =(yes "Do the statistical distributions have the same mean?"))))

(defrule noise
        (stat 1)
=>
(assert (bp hidden)))

(defrule noisea
        (stat 1)
        (inputs ?inp)
        (outputs ?outp)
        (hid ?hid)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "The classification of patterns into their corresponding
statistical distributions is very difficult for an ANN when the means
are the same.  This problem is probably best solved by a rule based technique." crlf crlf)
(printout t "However, if an ANN is going to be used:" crlf)
(printout t "Use a Back Progation Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?hid )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the default Back Prop
provided by BrainMaker Professional from California Scientific Software." crlf ))

(defrule pc17
        (path pc)
        (noise 0)
        (stat 0)
=>
```

```
        (notify "Most pattern classification problems work best with binary outputs, in which each output
node corresponds to a distinct class and only one output has a positive value for each pattern.")
(assert (outtype2 =(yes "Are the outputs binary?"))))

(defrule pc18
        (outtype2 1)
        (path pc)
=>
        (assert (stat2 =(yes "Does only one output have a positive value?"))))

(defrule pc19
        (path pc)
        (outtype2 0)
=>
(assert (outfired 0)))

(defrule pc20
        (path pc)
        (stat2 0)
=>
(assert (outfired 0)))

(defrule noiseb
        (stat 0)
        (stat2 1)
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Probabalistic Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?tra )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the PNN
 provided by Neuroshell2, from Ward Systems Group, Inc." crlf ))

(defrule pc12
        (path pc )
        (outtype 1)
=>
(assert (outfired =(yes "Does only one output have a positive value for each training pattern?"))))

(defrule pc13          ;sparse data so use PNN
        (outfired 1)
        (or (path pc) (path none))
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
```

```
        (test (< ?tra (* 5 ?inp)))
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Probabalistic Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?tra )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the PNN
 provided by Neuroshell2, from Ward Systems Group, Inc." crlf ))

(defrule pc14        ;plenty of data so use LVQ
        (outfired 1)
        (or (path pc)
                (path none))
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
        (test (>= ?tra (* 5 ?inp)))
=>
(assert (hid =(/ (* .10 ?tra) ?outp)))
(assert (path2 round)))

(defrule pc15a
        (path2 round)
        (hid ?hid)
=>
(assert (hid3 =(round ?hid)))
(assert (path2 next)))

(defrule pc16        ;plenty of data so use LVQ
        (outfired 1)
        (path2 next)
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
        (hid3 ?hid)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Learning Vector Quantization network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The Kohonen hidden layer should have  %d nodes %n" (* ?outp ?hid))
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the LVQ
 provided by Professional II Plus,  from NeuralWare, Inc." crlf ))

(defrule pc15       ; multiple outputs firing so use GRNN
        (outfired 0)
```

```
        (or (path pc) (path none))
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a General Regression Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?tra )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the GRNN
 provided in the Neuroshell2 software package by Ward Systems, Inc.." crlf ))

(defrule time1
        (application time_series_analysis)
=>
(printout t "You have selected a time series analysis problem" crlf crlf )
(notify "Time series analysis problems usually try to predict an output based on the past few outputs, for
example using the past five values to predict a 6th value.  Data files should be structured accordingly.")
(notify "Often it is easier to use data that spans long time periods, for example weekly versus daily, because it
contains less fluctuation.")
(notify "With time series prediction it is usually easier to predict the change in an output, such as a stock price,
rather than predict an actual value.")
(notify "If the network is trained on actual values, it will have a hard time generalizing outside of the range it
was trained on.")
(assert (path ts)))

(defrule time1a
        (path ts)
        (outputs ?out)
=>
(notify "Often, ANN's are used to classify behavioral data, which involves people and therefore contains large
amounts of noise.  Examples are stock market prediction and credit risk assessment.")
(notify "Other types of classification involve physical data, where a functional relationship is involved")
(assert (behave =(select "What is the nature of your data?" physical behavioral))))

(defrule time1b
        (path ts)
        (behave physical)
=>
(assert (bp hidden)))

(defrule time1c
        (path ts)
        (behave behavioral)
        (inputs ?inp)
        (outputs ?out)
        (examples ?tra)
=>
```

```
(assert (hid =(+ 1 (/ ?tra (* 5 (+ ?inp ?out)))))))

(defrule time2
        (path ts)
        (hid ?hid)
=>
(notify "A good algorithm exists for time series analysis but it takes longer to train than other methods.")
(assert (choice =(select "Do you have time to wait, or would you like a quick, less accurate model?"
slow_but_accurate quick_model))))

(defrule time3  ;Cascade Correlation model
        (choice slow_but_accurate)
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
        (hid ?hid)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Back Propagation Neural Network with 1 hidden layer
and Cascade Correlation training" crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?hid )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the Back Propagation code
provided by Professional II Plus,  from NeuralWare, Inc." crlf )
(printout t "Use Norm-Cum-Delta Learning with Tanh transfer functions and
Cascade Correlation training." crlf ))

(defrule time4 ; Ward nets model
        (choice quick_model)
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
        (hid ?hid)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Experiments have shown that networks with multiple slabs
of hidden layer neurons, each with different transfer functions perform
well for time series analysis" crlf)
(printout t "Use a Back Progation Neural Network with 2 slabs of hidden
layer nodes. " crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "Each hidden layer slab should have  %d nodes %n" (+ 1 (/ ?hid 2)))
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the Back Propagation code
with multiple hidden slabs provided by Neuroshell2, available from
Ward Systems Group, Inc. " crlf ))
```

```
(defrule visual1
        (application visual_pattern_recognition)
=>
(printout t "You have selected visual pattern recognition" crlf crlf )
(assert (path vis)))

(defrule vis2
        (path vis)
        (inputs ?inp)
=>
(notify "When dealing with visual pattern recognition the data is usually organized in one of two ways, the
first is to have 2 inputs corresponding to the XY coordinates of a pixel, or feature.")
(notify "The second method is to have an MxN grid with M*N input nodes, where each input node
corresponds to a pixel, or feature.")
(assert (visual =(select "How is your data arranged?" coordinates grid))))

(defrule vis3 ; The coordinate path
        (visual coordinates)
=>
(assert (vout =(yes "Is the output a binary value, with only one node firing for each pattern?"))))

(defrule vis4
        (vout 1)
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Probabalistic Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?tra )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the PNN
 provided by Neuroshell2, from Ward Systems Group, Inc." crlf )
(printout t "For these types of problems, a low smoothing factor
2is recommended, between .05 and .1" crlf crlf ))

 (defrule vis5
        (vout 0)

=>
(assert (bp hidden)))

(defrule vis6
        (vout 0)
        (hid ?hid)
        (inputs ?inp)
        (outputs ?outp)
```

```
=>
 (printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Back Propagation Neural Network with 2 hidden layers" crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The first hidden layer should have  %d nodes %n" ?hid )
(format t  "The second hidden layer should have  %d nodes %n" ?hid )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the Back Propagation code
provided by Professional II Plus,  from NeuralWare, Inc." crlf )
(printout t "Use Norm-Cum-Delta Learning with Tanh transfer functions." crlf )
(printout t "Depending of the complexity of the patterns, this can be a
tough problem for Back Propagation " crlf )
(printout t "If problems develop, try to convert the output
to a binary form with only one node giving an output at a
time and use a PNN." crlf crlf ))

(defrule vis7   ;  input grid
        (visual grid)
=>
(assert (vout2 =(yes "Are the outputs binary?"))))

(defrule vis8
        (vout2 1)
        (examples ?tra)
        (inputs ?inp)
        (outputs ?outp)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a General Regression Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?tra )
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the GRNN
provided in Neuroshell2 by Ward Systems, Inc.." crlf ))

(defrule vis9
        (vout2 0)
=>
(assert (bp hidden)))

(defrule vis10
        (vout2 0)
        (hid ?hid)
        (inputs ?inp)
        (outputs ?outp)
=>
(printout t "================================" crlf)
```

```
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "==============================" crlf)
(printout t "Use a Modular Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?hid)
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the MNN
provided by Professional II Plus,  from NeuralWare, Inc." crlf ))

(defrule none1
        (application none_of_the_above)
=>
 (assert (path none)))

(defrule none2b
        (path none)
        (inputs ?inp)
=>
(assert (outtype2 =(yes "Are the outputs binary?"))))

(defrule none3
        (outtype2 1)
        (path none)
=>
        (assert (stat2 =(yes "Does only one output have a positive value?"))))

(defrule none4
        (path none)
        (outtype2 0)
=>
(assert (bp hidden)))

(defrule none5
        (path none)
        (stat2 0)
=>
(assert (outfired 0)))

(defrule none6a
        (outputs ?outp)
        (test (> ?outp 1))
        (path none)
        (stat2 1)
=>
(assert (outfired 1)))

(defrule none6b
        (outputs ?outp)
        (test (= ?outp 1))
        (path none)
        (stat2 1)
```

```
=>
(assert (outfired 0)))

(defrule none7
        (path none)
        (hid ?hid)
        (inputs ?inp)
        (outputs ?outp)
        (test (> ?outp 1))
        (bp hidden)
=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Modular Neural Network with 1 hidden layer." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "The hidden layer should have  %d nodes %n" ?hid)
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended software is the MNN
provided by Professional II Plus,  from NeuralWare, Inc." crlf )
(printout t "If this doesn't work, try using Back Prop with the
BrianMaker Software System from California Scientific Software." crlf ))

(defrule none8
        (path none)
        (inputs ?inp)
        (outputs ?outp)
        (hid ?hid)
        (test (= 1 ?outp))

=>
(printout t "================================" crlf)
(printout t "ANN EXPERT SUMMARY REPORT" crlf)
(printout t "================================" crlf)
(printout t "Use a Back Propagation neural network with
2 slabs of hidden nodes with different transfer functions." crlf)
(format t "The input layer should have %d nodes %n" ?inp)
(format t  "Each hidden layer slab should have  %d nodes %n" (+ 1 (/ ?hid 2)))
(format t  "The output layer should have %d nodes %n" ?outp )
(printout t "The recommended architecture for this problem
is the Ward Net, provided by Neuroshell2, available
from Ward Systems Group, Inc." crlf ))
```

```
(defrule error
        (inputs ?inp)
        (outputs ?outp)
        (examples ?tra)
        (or (test (< ?inp 0)) (test (< ?outp 0)) (test (< ?tra 0)))
=>
(printout t "Invalid input number, please reset the program" crlf crlf))
```

APPENDIX C. ANNEXPERT USER MANUAL

Installing Eclipse

In order to run the ANNexpert, the Eclipse v3.2c expert system shell must first be installed on your computer's hard drive. First, a directory must be created in order to install eclipse. To make a directory on the c drive named **eclipse** type the following at the DOS prompt: **md eclipse**. Next the eclipse system must be installed into this directory. Insert the floppy disk in drive a on your computer and type the following: **install a: c: \eclipse**. Once eclipse is installed, copy the file **annxpert.clp** to the directory in which you will be working by using the DOS copy command.

Initializing Eclipse

Once Eclipse has been installed on your hard drive (assuming drive c) the executable file to start the program will be found in the following directory **c:\eclipse\windows\msc\toolkit**. The name of the executable file is **ewin2tk.exe**. This file may be executed by selecting it with the mouse in the MS Windows File Manager and double clicking on it. The directory also contains a file called **ewin2tk.dll** which contains icons which may be used to set up an Eclipse program item in the MS Windows Program Manager.

Once the **ewin2tk.exe** file is executed a window will be brought up which will be labeled **T.H.E. Eclipse**. This window will have a menu bar across the top with the following menus:

- **File**
- **Execute**
- **Knowledge**
- **Breakpoints**
- **Window**
- **Fire**
- **Stop**

Loading the ANNexpert

Using the mouse select **Load** from the from menu, followed by **Eclipse**. A window will be brought up labeled **Load Eclipse**. In this window select the path for the directory in which you copied the file **annxpert.clp**. Select this file from the menu and click **OK**.

A file window will pop up asking to **Reset the knowledge base?**. Select **yes**. This will be followed by a window asking **Run the knowledge base?**. Again select **yes**. This will begin execution of the ANNexpert program.

Using the ANNexpert

Two introductory windows will appear on the screen. Select **OK** to dismiss them. These will be followed by a series of menus and questions about the problem you are attempting to model. Simply follow the directions and answer the questions posed by the expert system and the program will generate a description of the appropriate model to use for the given problem.

The ANNexpert assumes that the user already has a problem to be modeled which has been formulated into a set of inputs and outputs which are contained in a training file. Data patterns in the training file should be arranged sequentially with inputs followed by outputs for

each training vector. It is best to have an independent file for testing the resulting ANNs. Many researchers set aside about one third of their data for a particular problem and reserve it to test the generalization ability of the neural network model. Both training and test files should be saved in an ASCII format.

When the expert system is done reading in the description of the problem from the user, then an output report will be printed to the **Standard Output** window. If a problem developed while executing the program, select **Stop** from the menu bar to end execution. When done with a program run, it is helpful to clear the **Standard Output** window between runs by selecting **Wipe** from the pull-down menu in the top left corner of the **Standard Output** window.

To run the program again, select **Reset** from the **Execute** menu. When this is done all previous problem characteristics will be erased from memory and the program will be executed again.

By selecting the command **Update eclipse.ini** from the **File** menu, the path you selected to load the **annxpert.clp** file will be set as the default saving time when loading the program during future runs.

To exit Eclipse, select **Exit** from the **File** menu.