Applied Intelligence 12, 183–192 (2000)

Predicting Aflatoxin Contamination in Peanuts: A Genetic Algorithm/Neural Network Approach

C.E. HENDERSON AND W.D. POTTER
Artificial Intelligence Center, University of Georgia, USA

R.W. MCCLENDON AND G. HOOGENBOOM
Biological and Agricultural Engineering, University of Georgia, USA

Abstract. Aflatoxin contamination in peanut crops is a problem of significant health and financial importance. Predicting aflatoxin levels prior to crop harvest is useful for minimizing the impact of a contaminated crop and is the goal of our research. Backpropagation neural networks have been used to model problems of this type, however development of networks poses the complex problem of setting values for architectural features and backpropagation parameters. Genetic algorithms have been used in other studies to determine parameters for backpropagation neural networks. This paper describes the development of a genetic algorithm/backpropagation neural network hybrid (GA/BPN) in which a genetic algorithm is used to find architectures and backpropagation parameter values simultaneously for a backpropagation neural network that predicts aflatoxin contamination levels in peanuts based on environmental data. Learning rate, momentum, and number of hidden nodes are the parameters that are set by the genetic algorithm. A three-layer feed-forward network with logistic activation functions is used. Inputs to the network are soil temperature, drought duration, crop age, and accumulated heat units. The project showed that the GA/BPN approach automatically finds highly fit parameter sets for backpropagation neural networks for the aflatoxin problem.

Keywords: genetic algorithm/neural network hybrid, genetic algorithms, neural networks, genetic learning, aflatoxin prediction

Introduction

Aflatoxin contamination in peanuts is a well-known agricultural problem of health concern [1]. This concern produces financial pressures for growers, resulting in efforts to develop methods to predict contamination levels prior to harvest. A relationship between environmental factors and aflatoxin levels has been identified [2]. In particular, factors such as soil temperature and drought are believed to have a strong effect on aflatoxin levels in peanut [3]. Consequently, there is interest in developing a tool that predicts aflatoxin levels based on environmental data. Such a tool could be used to guide crop management decisions and to select a harvest date that minimizes contamination levels.

Artificial neural networks (ANNs) provide a methodology for constructing predictive tools and have been used in prior efforts for the aflatoxin prediction problem. However there are difficulties inherent in the process of constructing ANN models. To make accurate predictions, an ANN requires selecting an appropriate architecture and searching for the connection weights. The relationship between architecture and network performance is poorly understood, and searching for the best architecture is a heuristic task. This search is often conducted by hand in a trial and error fashion and can be tedious and error prone. Searching by hand will be termed the traditional method in this paper. The task of searching for connection weights, known as learning or network training, is also challenging. The

backpropagation algorithm is an effective technique for ANN training, but the algorithm itself requires setting parameters such as momentum and learning rate. Thus backpropagation creates a new problem while solving another. A technique for algorithmically locating and evaluating architectures and ANN parameters would therefore be useful. Such a search will be termed an automated method hereafter.

Genetic algorithms provide a way to search large, poorly understood spaces, and have been used successfully to set ANN parameters. By linking the global search capabilities of the genetic algorithm with the modeling power of ANNs, a highly effective tool for aflatoxin prediction can be constructed.

The goal of this research was to compare the performance of an automated genetic algorithm/ backpropagation hybrid (GA/BPN) with a traditional backpropagation ANN in predicting pre-harvest aflatoxin contamination levels. The specific objectives were as follows. The first objective was to develop a GA representation to allow an automatic search for ANN architecture and parameters using the mean squared error of the backpropagation ANN as the fitness function. The second objective was to perform a comparison with the simple BPN results of Henderson [4] which was a traditional search for ANN architecture and parameters using backpropagation.

The next section provides some background material on genetic algorithm/neural network hybrids as well as the aflatoxin prediction problem. The subsequent section presents the materials and methods used in our approach. Details on our implementation, the data collection and preparation, and model development are then presented. The results and conclusions section contain the tables and graphs supporting our conclusions, as well as a summary of the effectiveness of our approach.

Background

Genetic algorithms (GAs) are search methods based on the mechanics of natural selection and natural genetics [5]. They are effective global search techniques that avoid many of the shortcomings of more traditional search techniques such as hill climbing. In particular, hill climbing is highly susceptible to getting stuck in local minima. A genetic search explores many points in the search space simultaneously, thus blanketing regions and providing a mechanism for avoiding local minima. Genetic algorithms do not rely on gradient information to guide the search, but require only a function that can assign a rough measure of merit to a candidate solution. They have proven effective for locating high quality solutions among large, poorly understood search spaces. These strengths match well with the difficulties of configuring neural networks, so the genetic algorithm was selected to locate network parameter values for our experiment.

There are at least three major elements of a traditional genetic algorithm: representation, operators, and the objective function. The representation provides a specification for storing the information that is to be evolved toward a solution. The genetic algorithm maintains a pool of potential solutions, which are built according to the choice of representation. Each of these candidate solutions is termed a chromosome. Examples of common representations include storing chromosomes as bit strings, or, alternatively, as strings of real numbers. The sum of the genetic makeup of a chromosome is called its genotype. This genotype decodes to a phenotype, which is an element of the search space, in this case, a set of ANN parameters.

The task of choosing an appropriate representation is specific to the particular problem being solved. Some general considerations may be specified for the problem of evolving neural networks using genetic algorithms. According to Balakrishnan and Honavar [6], there are at least nine key factors to consider when choosing a genetic representation for neural networks.

1. Completeness: Is the representation capable of encoding all phenotypes in the search space?
2. Closure: Does the representation produce only genotypes that decode to acceptable phenotypes?
3. Compactness: How efficiently, in terms of space, does a genotype capture the phenotype?
4. Scalability: How much must the genotype grow in order to represent a larger phenotype?
5. Multiplicity: Do multiple genotypes decode to the same phenotype? Does one genotype conceivably decode to multiple phenotypes?
6. Ontogenetic Plasticity: Is the decoding of a genotype to a phenotype also a function of the environment? In other words, does a given genotype always decode to the same phenotype, regardless of environmental factors?
7. Modularity: Does the genotype encode functionally independent neural network sub-units in a modular way?

8. Redundancy: When sub-units of a neural network exist that are functionally equivalent, does the genotype encode separately for each instance of the sub-unit, or does it encode the sub-unit once and make copies?
9. Complexity: What sort of effects does the representation have on the complexity of the GA as a whole?

Operators provide the machinery by which pools of chromosomes are manipulated. Common operators for a genetic algorithm include a selection scheme, a crossover operator, and a mutation operator. The selection scheme is a method for choosing chromosomes from a pool to be used as parents for a new pool. Crossover is a mechanism for combining parent chromosomes to produce offspring. Mutation modifies chromosomes at random in order to produce a continual source of genetic diversity.

The objective function is of key importance to a GA implementation because it is the sole source of information about the value of candidate solutions. Given a chromosome, the objective function returns a number indicating merit. This measure of merit is used by the selection scheme to determine which solutions survive and which ones perish.

Our work employs a simple GA that uses selection, crossover, and mutation. However, these operators play roles that are not strictly traditional, as will be discussed in the Model Development Section. Many efforts, beginning in the late 1980's, have employed genetic algorithms to evolve parameters for ANNs. Representation is an important issue for such a system and may be generally grouped into direct and indirect encodings. Miller et al. [7] evolved inter-node connectivity for a feed-forward ANN with a fixed number of layers and hidden nodes. Connectivity was represented using a binary string in which each bit corresponded to a pair of nodes in the network, and all possible pairs were represented. A 1 in the chromosome indicated that a connection was present, while a 0 meant that the node pair was not connected. This is a paradigmatic example of what is termed a direct encoding, which means that there is a discrete item (a bit) in the chromosome for each item (a connection) represented in the network. Miller et al. [7] used a specialized crossover operator that ensured that bits were copied in blocks, each corresponding to all of the outgoing links of a single node. It was believed that such groups of links were functional units and that preserving them would aid convergence. The mutation operator consisted of flipping bits according to a probability that was set to a low value. The selection operator used a common fitness proportionate scheme. The fitness function decoded a chromosome to an architecture, trained the network using backpropagation for a preset number of epochs, then returned the sum of the squares of the errors on the training data set for the last epoch. The system was tested on the XOR problem, a real valued four-quadrant problem, and a pattern copier that contained fewer hidden nodes than inputs. The approach was effective, but the three problems were each quite simple and did not provide a full test of the computational power of the method [8].

Marti [9] used direct encoding in a novel way that included two genetic systems. The first system, called the inner system, was used to generate neural network connectivity based on a representation while the outer system evolved the representation itself. Indirect encodings were developed to overcome some of the shortcomings of direct encodings. Because direct encoded chromosomes map directly to a phenotype, they grow in size at a rate equal to that of the phenotype. This may make direct encodings computationally impractical for large problems. Indirect encodings, on the other hand, include small chromosomes that decode to larger networks via sets of generative rules. Kitano [10] provided an elegant example of an indirect representation known as a grammatical encoding. By this scheme, chromosomes encoded a set of grammar expansion rules. Such chromosomes were decoded by repeatedly applying the rules until a network was produced. Fitness was deduced by decoding and training via backpropagation, similar to Miller et al. [7], described above. It was found that this encoding was effective for a number of low complexity problems [8].

A recent comparison between direct and indirect encoding schemes may be found in Gruau et al. [11]. In this work, a direct encoding of weights for a fixed architecture neural network was compared with an indirect one, termed a cellular encoding, which described both architecture and weights. An interesting facet of this work is that networks were evolved for a fairly complex set of engineering problems requiring that a simulated cart, termed a pole cart system, be directed in order to keep movable poles upright. Also, the cellular encoding allowed the use of real valued weights, an ability lacking in many prior efforts. It was found that the cellular encoding learned more slowly, but saved a great deal of human design effort by automatically configuring network architectures. Dasgupta [12] used

genetic algorithms to evolve network architectures and weights for a similar pole cart problem. Examinations of strengths and weaknesses of a number of encoding schemes may be found in Roberts and Turega [13] and Balakrishnan and Honavar [6].

A number of efforts have evolved values for backpropagation parameters for neural networks. Harp et al. [14] used a genetic algorithm to set the topology and learning rate for a backpropagation neural network. An innovation of the approach was the inclusion of a mechanism by which the learning rate varied over time and among groups of nodes. The design was found to be effective on several simple problems, including XOR. The primary criterion by which networks were evaluated was speed of convergence to an acceptable solution.

Schaffer et al. [15] also used a genetic algorithm to set the topology and learning parameter for a backpropagation neural network. The search included the momentum parameter and the range of initial weights. Networks were trained on a 4-bit coding problem for a maximum of 200 training epochs. It was found that low learning rates (0.25 and 0.5) and low initial weights ranges (-0.125 to 0.125 and 0.25 to 0.25) were most effective.

Belew et al. [16] used genetic algorithms to choose values for the learning rate and momentum parameters of a backpropagation neural network. Successful networks were trained using unexpectedly high learning rates, but it was concluded that this was a result of limiting training sessions to a low number of epochs.

Chalmers [17] explored the interesting possibility of using genetic algorithms to evolve the learning algorithm. Chromosomes were decoded to produce procedures that used information such as node activation and connection strength to modify connection weights. A number of effective learning rules were discovered. Chalmers [17] found that more complex rules (i.e. those encompassing more tasks) tended to be effective for a broader range of learning problems. It was suggested that the approach might be most useful for unusual network architectures for which effective learning rules were not known.

There have been several efforts at the University of Georgia that use ANNs to model the aflatoxin prediction problem. Parmar et al. [18] trained three-layer feed-forward ANNs using backpropagation. Data were obtained and allocated to a training and test set, and results were compiled using a range of values for the number of nodes in the hidden layer using the traditional method to determine architecture and parameters. Pinto [19] used the peanut field data compiled by Parmar et al. [18] as well as additional data from subsequent years. He created a validation data set in addition to training and test sets. He also partitioned the data into categories of damaged and undamaged samples. At peanut processing plants the kernels are identified and separated into quality categories. Damaged refers to those which have received some mechanical damage during harvesting and transporting. Results were generated for both models across training, test, and validation data sets. In each case, the network design used was the three-layer, feed-forward neural network with backpropagation as the learning algorithm. Similar to Parmar et al., Pinto performed a search for architecture and parameters with the traditional method although he added an evaluation on the validation data set.

Both Parmar et al. [18] and Pinto [19] used the software package NeuroShell 2 (Ward Systems Group, Inc., Executive Park West, 5 Hillcrest Drive, Frederick, Maryland) to train networks. NeuroShell 2 includes an enhanced version of the backpropagation algorithm. Because the code is proprietary, the exact nature of the enhancements is unknown. In order to study the effects of automated parameter setting methods on a backpropagation network, it was necessary to have results from a common implementation of backpropagation. In this case, common is used to mean a simple, publicly available algorithm such as that of Rumelhart et al. [20]. The accuracy could be used as a baseline performance measure by which automated method enhancements could be evaluated. Henderson [4] used a common backpropagation implementation to reproduce the work of Pinto [19] using identical data sets. It was concluded in Henderson [4] that the common algorithm produced results that appeared to be generally representative of the performance of backpropagation when parameters are set by hand and would therefore be useful for a performance comparison study.

Materials and Methods

Model development is usually started by gathering sets of patterns that are generally representative of the problem to be modeled. These data are allocated to a training set, testing set, and validation set. The training set can be used by the backpropagation algorithm to adjust the weights of the network to minimize the sum

of the square error between the observed and predicted output. Periodically during training, the testing data set is applied to the current model in feed-forward mode only to calculate the error. The set of weights which produces the minimum error on the testing data set is saved. In this manner over-training is avoided and the ability of the ANN to generalize is maintained. The validation set serves to evaluate the performance of the network on data not used on training or testing after training is complete. We followed this same approach in data partitioning by using the data sets originally developed by Pinto [19]. For our study we chose to use only the undamaged peanut samples. Since the samples can be categorized and separated, the primary economic value of a harvest is due to undamaged peanut kernels.

Implementation

Both custom written code and modified versions of publicly available code were used for implementation of this project. Tools for the ANN portion were coded using C++. Features include the ability to allocate and de-allocate memory resources for networks dynamically and to set parameters such as number of hidden nodes, number of hidden layers, momentum, and learning rate at run time.

The implemented procedure for training via simple backpropagation proceeds as follows:

1. The network is trained on the training data for a specified number epochs using backpropagation.
2. Mean absolute error is calculated for the network across all test set patterns.
3. If test set mean absolute error is the lowest encountered then the network weights are saved.
4. If test set mean absolute error has not improved for a specified number of epochs, training is stopped and the weights that produced the best network throughout the training procedure are restored. Otherwise, go to step 1.

Genetic algorithm functions were obtained by converting and modifying GAlib a set of publicly available classes from the Massachusetts Institute of Technology. GAlib is available at http://lancet.mit.edu/ga/. Versions of our search algorithm were developed to run under Windows and UNIX.

Data Collection and Preparation

The data used by Pinto [19] were obtained from the United States Department of Agriculture Agricultural Research Service (USDA ARS) National Peanut Research Laboratory (NPRL) at Dawson, Georgia. Measurements were taken from florunner peanuts that were grown in environmentally controlled stands. Following harvest, all peanuts were analyzed for aflatoxin contamination. Aflatoxin levels for entire stands were determined using the weighted average of the grade values.

Data sets were available for the years 1985 through 1995 for several drought condition treatments. Each observation consisted of the aflatoxin level and associated environmental values for a specific plot and season. Environmental values included length of drought stress period (days), mean soil temperature (degrees Celsius), crop age (days), and accumulated heat units (degrees Celsius days). Drought stress was the number of consecutive days of drought conditions, while mean soil temperature was the mean temperature of the soil during this period. Crop age was the number of days from planting to harvesting. Accumulated heat units (AHU) was the daily accumulation of heat above 25 degrees Celsius during the drought period. This value was calculated by the following equation from Parmar et al. [18]:

$$\text{AHU} = (\text{MEAN SOIL TEMPERATURE} - 25) * \text{LENGTH OF DROUGHT STRESS PERIOD}$$

These four environmental factors were used as inputs for the ANNs developed in this project.

Because the inclusion of damaged peanuts in the data introduced a great deal of noise, Pinto [19] developed two neural network models. The first (model A) included data for both damaged and undamaged peanuts, while the second (model B) included only measurements for undamaged peanuts. The available data were used to produce two pools, one for model A and one for model B. For each of these pools, the data were partitioned into training, testing, and validation sets. The Pinto [19] data were used later in Henderson [4] with a focus on results of a common backpropagation ANN model. The purpose of our current project was to compare the effectiveness of a genetic search for network parameters to the traditional method for this search using backpropagation. To get meaningful comparisons between results, our current project used

the same data sets used by Henderson [4]. For our research, we limited the study to data sets consisting of undamaged peanut samples.

ANNs were evaluated by comparing predictions against observed or target values for the patterns in the data sets. The R^2 values, the square root of mean squared error (RMSE), and the mean absolute error (MAE) were used as metrics.

Model Development

There are a number of parameters that must be assigned values in order to develop an accurate backpropagation ANN model. Such parameters include number of layers, connectedness of layers, number of hidden nodes, learning rate, and momentum. There is no clear-cut procedure for choosing values for these parameters that will produce the best network model, hence neural network development is often more art than science. The focus of this project was to employ the search power of the genetic algorithm to choose ANN parameters, however problem complexity precluded evolving values for all of these parameters. The number of hidden nodes, learning rate, and momentum were chosen to be evolved, because it was observed in Pinto [19] and in Henderson [4] that these three parameters have a strong impact on the quality of the backpropagation networks that are produced for aflatoxin prediction. An architecture that includes three fully connected layers was effective in Pinto [19] and in Henderson [4], so a similar one is used here.

To employ the genetic algorithm to search for neural network parameters, there are a number of design issues that must be addressed. These issues are related to the representation, selection scheme, crossover scheme, mutation operator and the objective function. Choice of representation dictates many design decisions and should be examined first. The items to be represented are learning rate, momentum, and number of hidden nodes, which may be easily represented as real numbers. The network parameters to be evolved were therefore represented as strings of real values. There are a number of reasons to choose this representation over the traditional bit string for this problem including the following:

1. The real valued representation has been used effectively for many similar problems.
2. Because the search space consists of sets of numeric values, each of which corresponds to a gene, decoding the chromosomes for evaluation is a simple task.
3. Crossover occurs only at real value boundaries, avoiding unwanted large leaps in value as described below.
4. Genotypes are represented succinctly, consisting of three floating-point numbers.
5. Because CPUs have built in floating point processors, manipulation of chromosomes is very efficient.

A chromosome consisted of three real numbers constrained to lie between 0.0 and 1.0. Decoding proceeded as follows: the first and second numbers are used to produce the learning rate and the momentum rate, while the third number is scaled by a preset maximum allowable number of hidden nodes, converted to an integer, and used to set the number of hidden layer nodes. To reduce computational demands, an upper limit of 21 was placed on the number of hidden nodes for the networks examined in this experiment. The most effective networks in Pinto [19] and Henderson [4] contained no more than 14 hidden nodes, so the limit of 21 was expected to be sufficiently high.

Our representation led to some interesting results. Typically, crossover produces a large portion of genetic diversity during a search. However, with a real valued representation scheme, crossover occurred only at real number boundaries. Hence, crossover played a more limited role than usual in producing new genotypes. Mutation was the only mechanism that could modify a real value within a chromosome. For this reason, mutation played an unusually large role in seeking new genetic material for this experiment. A crossover scheme was selected based on preliminary results. Several tests were conducted using uniform, two-point, and one-point crossover schemes. Single point crossover consistently produced better convergence and better final solutions as expected.

Because mutation played an unusually important role for this implementation, great care was taken in its development. Consider a hypothetical, highly fit chromosome that has just been produced by crossover. Because fitness is high, presumably the real values in the string are close to optimal. This may not always be the case, but it is the assumption that this will *often* be the case that underpins the power of the GA. A traditional mutator might traverse the bits in the chromosome, flipping an occasional bit based on a biased coin toss. However, with real values this is not desirable. Flipping a bit in a floating-point number will

frequently result in new values that are much greater or smaller than the original. Because the real numbers are assumed to already be close to optimal values, large jumps are undesirable. For this reason, a customized mutator was implemented for use with the real valued strings. The mutator picked a new value based on a distribution around the value being mutated. In effect, it added or subtracted a small number from the allele. Hence, the mutator should provide significant new genetic diversity, but not destroy valuable alleles by changing them drastically. Because our implementation relied heavily on the mutator to produce diversity, a relatively high mutation rate of 0.08 was chosen.

For a given genotype, the fitness of its phenotype is not unique for this problem. This is because the objective function employs backpropagation, which relies on an initial random seeding of the weight space to provide a search starting point. Hence, a genotype that can lead to an excellent network may receive a poor fitness rating due to poor initial weight assignments. The initial weights may be such that backpropagation gets stuck in a local minimum. To avoid this problem it is essential to evaluate promising genotypes multiple times. Luckily, the GA provides just such a mechanism in selection. Many times selection is merely a way for a GA to choose the most promising genotypes to use as parents, but in this case there is an added bonus. If selection has no bias against selecting identical chromosomes from a pool, then the genotypes will get evaluated more than once. After a few generations, there will be many copies of the same fit chromosome in a population. Typically, this leads to premature convergence and is to be avoided. However, with our specialized mutator and high mutation rate there is still strong pressure toward diversity. For this reason, selection is of unusual importance to this implementation, as it provides a mechanism by which potentially strong genotypes may be evaluated more than once. Standard roulette wheel selection provides all of the features that are desirable and was chosen.

The objective function provides all of the evolutionary pressure for a genetic search, so its design is a key concern. For aflatoxin prediction, the problem is to develop a genetic algorithm that will evolve parameters that, when used with backpropagation, will lead to accurate ANNs. So the measure of fitness for a chromosome was chosen to be a function of its accuracy with backpropagation. For a given chromosome to be evaluated, the chosen objective function proceeded as follows:

1. The chromosome was decoded to produce values for learning rate, momentum, and number of hidden nodes, as described previously.
2. A three-layer feed-forward network with the indicated number of hidden nodes was allocated dynamically and seeded with random weight assignments.
3. The network was trained via simple backpropagation using the method described previously.
4. The square of the training set mean absolute error (the difference between observed and predicted aflatoxin level) for the final network was returned as the fitness value.

In summary, our GA used mate selection, crossover, and mutation in a fixed generation processing environment. We used roulette wheel selection, one-point crossover, and an increment/decrement mutation operation. Our mutation probability was 0.08. We made numerous runs with crossover probabilities ranging from 0.5 to 0.8, population sizes ranging from 500 to 5000, and maximum generations ranging from 1000 to 20000. Our fitness value included the error value plus the selection mechanism with a 15-unit tolerance (discussed shortly).

The GA in this case minimized the objective function. The implemented scheme was an attempt to be true to the underlying goals of backpropagation training. Termination of training was determined by the test set, and all evolutionary impetus is provided by the training set. The training set mean absolute error is squared in step 4 of the objective function procedure as a way to penalize poor networks more, thus speeding convergence.

There were some marked differences between training a neural network using simple backpropagation and training one via the GA/BPN. An important consideration regarded deciding how to evaluate and compare results. Note, however, that the genetic algorithm and simple backpropagation network produce solutions in different ways. Backpropagation incrementally improves the network on the training set, using the most recent weight assignments as a starting point for each training event. It is expected that the network will show concomitant improvement on the test set as small training set improvements are made. Hence, backpropagation moves the network along a performance continuum, exploiting local information along the way, until a minimum is encountered. The genetic algorithm,

however, does not exploit this type of information, relying entirely upon one number (training set mean absolute error) to provide training pressure. The result is the GA may not proceed smoothly along a continuum toward a solution as with backpropagation. Instead, it will produce pools of networks that are increasingly good fits for the training data, but it is difficult to determine which of these networks provides the best overall solution. Therefore a solution selection scheme was necessary to keep the best networks as they were encountered. Because any given network in a pool has a chance of being the best ever, it is necessary to consider each chromosome of every generation. For this reason, the solution selection mechanism is included as an addition to the objective function, which accesses all chromosomes. The mechanism proceeded as follows:

1. After a network was decoded and trained via backpropagation, its mean absolute errors were calculated for the training and test data sets.
2. If the sum of the two error values was the lowest ever and the two errors were sufficiently close in value, then the network becomes the new best one.

The selection scheme is based on the assumption that the network performing best overall on the training and test data sets will provide the best predictions for new patterns. However, this requires a procedure for determining what best means. Clearly, it is desirable to have a network that has a low total for the two error values. However, it is undesirable to keep a network that has an extremely low value for one set, but not the other. Such a net constitutes over-fitting the one data set, and may not be robust. For this reason, the second criterion of step 2 was included, namely that the errors must be close in value. Step 2 requires a hard coded tolerance value that specifies how close the two error values must be to each other. This tolerance is based on the difference observed in high quality networks from previous studies and is set to an absolute value of 15.0 units. The genetic search was set to terminate after a predetermined number of generations, and the stored best network was returned as the solution. The networks were then evaluated in terms of mean absolute error, root mean squared error, and R^2 value for each of the training, test, and validation data sets. These values were compiled as final results for the experiment.

Results and Discussion

These results were developed using a GA/BPN and compared to those of Henderson [4] as shown in Table 1. Henderson's traditional method of searching for ANN parameters is labeled. Simple BPN. The Simple BPN produced a baseline performance measure for standard backpropagation for the aflatoxin problem. The network chosen for Simple BPN was the one with the least sum of mean absolute error values for training and test data sets. The table gives performance measures for the training, test, and validation data sets. The performance measures listed are R^2, square root of mean squared error (RMSE), and mean absolute error (MAE).

GA/BPN produced a network that was more accurate than the Simple BPN for all three data sets. Moreover, the GA/BPN did not require intervention by the user in terms of the heuristic search for ANN parameters. Hence, the GA/BPN approach is probably the more effective of the two techniques for our aflatoxin problem. Because aflatoxin prediction is similar to many other predictive problems, it is likely that the superiority of the GA/BPN will apply to other problems as well. Figure 1 shows a scatter plot for the training set performance of the network. The horizontal axis represents the observed aflatoxin contamination value, and the vertical axis corresponds to the predicted value for the model. The diagonal line represents the one to one fit of ideal accuracy. The points on this figure show a fairly even distribution around the ideal, indicating a good fit with one apparent outlier. Figure 2 depicts

Table 1. Comparison of GA/BPN network performance to Simple BPN.

	Training			Test			Validation		
Description	R^2	RMSE	MAE	R^2	RMSE	MAE	R^2	RMSE	MAE
Simple BPN[a]	0.69	92	60	0.57	76	56	0.39	98	81
GA/BPN	0.77	80	48	0.70	63	45	0.51	87	68

[a]The name for the traditional search method, by Henderson [4].

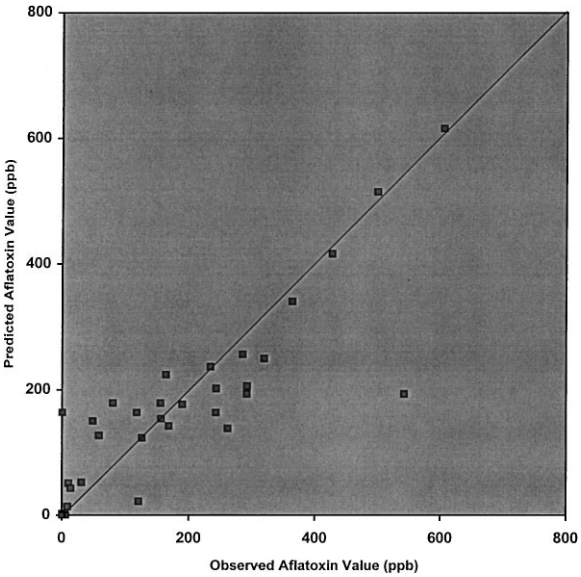

Figure 1. Plot of GA/BPN network predicted aflatoxin values versus observed values for the training data set.

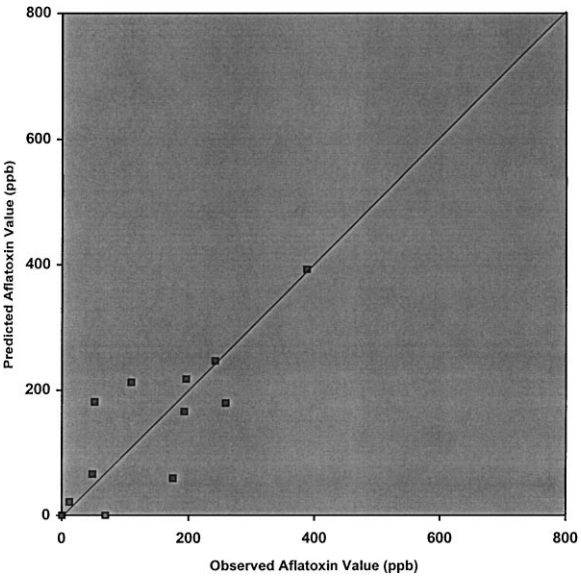

Figure 2. Plot of GA/BPN network predicted aflatoxin values versus observed values for the test data set.

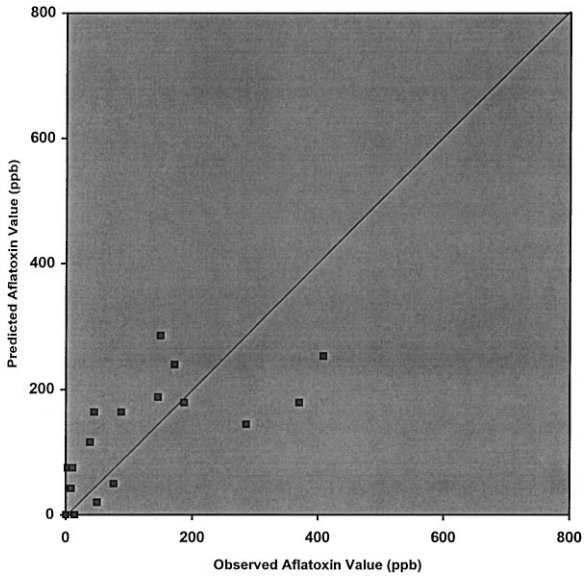

Figure 3. Plot of GA/BPN network predicted aflatoxin values versus observed values for the validation data set.

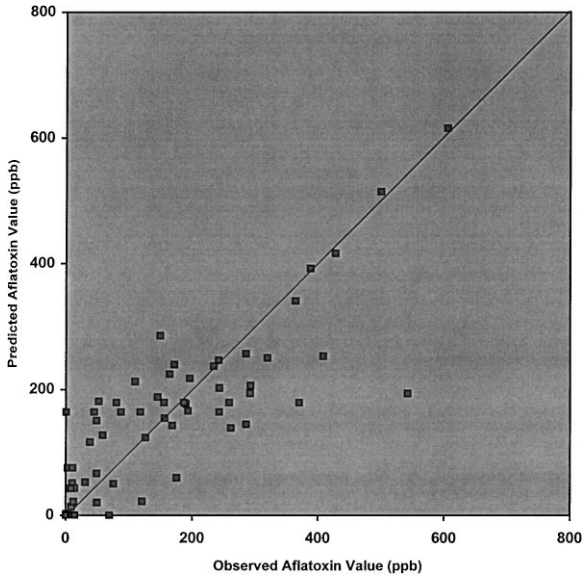

Figure 4. Plot of GA/BPN network predicted aflatoxin values versus observed values for combined data sets.

a similar plot for the test data set. The data spread in this case is similar to that of Fig. 1 and demonstrates a reasonably good fit to the data. Figure 3 shows a plot for the validation data with a less accurate fit to the observed values. From Table 1, it can be seen that the GA/BPN model produced less accurate predictions on the validation data set than on the training or testing data sets. Figure 4 includes the results for all three data sets from the prior three figures.

Figure 3 shows that GA/BPN made reasonably accurate predictions for the validation data set observations except for a few marked exceptions. A possible

explanation for the lower accuracy on the validation set is that the data sets are noisy. Predictions for the few outlying patterns, are often inaccurate. This is an indicator that these errors are the result of noisy data. Because there are a relatively small number of validation data points, these few highly inaccurate predictions have a marked effect on the accuracy metrics for the set. Future efforts will focus on obtaining additional aflatoxin observations.

Possibly the greatest value of the GA/BPN approach lies in the ease with which it may be used to find good networks. The only GA parameters that need to be set are crossover rate, mutation rate, and pool size. This may not sound better than standard backpropagation, which our study also require three parameter values (learning rate, momentum, and number of hidden nodes). However, genetic search was fairly insensitive to these settings, finding good networks for every set of GA parameter values that was tried. Simple BPN, on the other hand, was highly sensitive to parameter settings and would find extremely poor networks for the majority of values.

GA/BPN proved to be more accurate than Simple BPN for training networks to predict pre-harvest aflatoxin contamination levels in peanuts. It also required considerably less skilled human interaction by the user. It is expected that these advantages will apply to other problems as well.

References

1. U.S. Food and Drug Administration Center for Food Safety and Applied Nutrition, *Foodborne Pathogenic Microorganisms and Natural Toxins*, U.S. Food and Drug Administration: Washington, DC, 1992.
2. C.N. Thai, P.D. Blankenship, R.J. Cole, T.H. Sanders, and J.W. Dorner, "Relationship between aflatoxin production and soil temperature for peanuts under drought stress," *Transactions of the ASAE*, vol. 33, no. 1, pp. 324–329, 1990.
3. R.J. Cole, T.H. Sanders, R.A. Hill, and P.D. Blankenship, "Mean geocarposphere temperatures that induce preharvest aflatoxin contamination of peanuts under drought stress," *Mycopathologia*, vol. 91, no. 1, pp. 41–46, 1985.
4. C.E. Henderson, "Using genetic algorithms to evolve neural networks," Master's Thesis, Artificial Intelligence Center, The University of Georgia, 1997.
5. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley: Reading, MA, 1989.
6. K. Balakrishnan and V. Honavar, "Properties of genetic representations of neural architectures," in *Proceedings of the World Congress on Neural Networks*, 1995, pp. 807–813.
7. G.F. Miller, P.M. Todd, and S.U. Hegde, "Designing neural networks using genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 379–384.
8. M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press: Cambridge, MA, 1996.
9. L. Marti, "Genetically generated neural networks II: Searching for an optimal representation," in *Proceedings of the 1992 IEEE International Conference on Neural Networks*, 1992, pp. 221–226.
10. H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, pp. 461–476, 1990.
11. F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, 1996, pp. 81–89.
12. D. Dasgupta, "Evolving neuro controllers for a dynamic system using structured genetic algorithms," in *Proceedings of the Tenth International Conference on Industrial and Engineering Applications of Expert Systems and Artificial Intelligence*, Gordon and Breach Science Publishers, 1990.
13. S.G. Roberts and M. Turega, "Evolving neural network structures: An evaluation of encoding techniques," in *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Ales*, France, 1995, pp. 96–99.
14. S.A. Harp, T. Samad, and A. Guhu, "Towards the genetic synthesis of neural networks," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 360–369.
15. D. Schaffer, R.A. Caruana, and L.J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," *Emergent Computation*, S. Forrest, ed. MIT press, pp. 244–248, 1990.
16. R.K. Belew, J. McInerney, and N.N. Schraudolph, "Evolving networks: Using genetic algorithms with connectionist learning," CSE Technical Report CS90-174, La Jolla, CA, University of California at San Diego, 1990.
17. D.J. Chalmers, "The evolution of learning: An experiment in genetic connectionism," in *Proceedings of the 1990 Connectionist Models Summer School*, Morgan Kaufmann: San Mateo, CA, 1990, pp. 81–90.
18. R.S. Parmar, R.W. McClendon, G. Hoogenboom, P.D. Blankenship, R.J. Cole, and J.W. Dorner, "Estimation of aflatoxin contamination in preharvest peanuts using neural networks," *Transactions of the ASAE*, vol. 40, no. 3, pp. 809–813, 1997.
19. C.S. Pinto, "Prediction of aflatoxin contamination in peanuts using artificial neural networks," Master's Thesis, The University of Georgia, 1996.
20. D.E. Rumelhart, G.E. Hinton, and J.L. McClelland, "A general framework for parallel distributed processing," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press: Cambridge, Mass., 1987.