Simultaneous Optimization of Neural Network Function and Architecture Algorithm

Randall S. Sexton, Computer Information Systems, Southwest Missouri State University, 901 South National, Springfield, Missouri 65804, USA Office: (417) 836-6453
Fax: (417) 836-6907 Email: RandySexton@mail.smsu.edu

Randall S. Sexton is an Associate Professor of Computer Information Systems at Southwest Missouri State University. He received his Ph.D. in Management Information Systems at the University of Mississippi. His research interests include computational methods, algorithm development, artificial intelligence, and neural networks. His articles have been accepted or published in *Decision Support Systems*, *INFORMS Journal on* Computing, *European Journal of Operational Research*, *Journal of Computational Intelligence in Finance*, *Journal of End User Computing* and *OMEGA.*

Robert E. Dorsey, Chief Operating Officer, FNC Inc., 606 Van Buren Avenue, Oxford, MS 38655, USA Office: (662) 236-2020 Email: dorsey@fncinc.com

Robert E. Dorsey is the Chief Operating Officer at FNC Inc. He received his Ph.D. in Economics at the University of Arizona. Prior to attending graduate school he worked for fifteen years in the private sector. His research interests include computational methods, experimental economics and artificial intelligence. His articles have been accepted or published in *Decision Support Systems, Journal of Econometrics, Journal of Business and Economic Statistics, Journal of Computational Economics, European Journal of Operational Research,* and other leading journals.

Naheel A. Sikander, Computer Information Systems, Southwest Missouri State University, 901 South National, Springfield, Missouri 65804, USA Phone: (417) 861-6364

Naheel A. Sikander is a graduate student in the MBA program at Southwest Missouri State University. She received her BS in Computer Information Systems at Southwest Missouri State University. Her research interests include neural networks and artificial intelligence. She has previously published in the *International Journal of Intelligent Systems in Accounting, Finance & Management*.

ABSTRACT

A major limitation to current artificial neural network research is the inability to adequately identify unnecessary weights in the solution. If a method were found that would allow unnecessary weights to be identified, decision makers would gain crucial information about the problem at hand as well as benefit by having a network that was more effective and efficient. The Neural Network Simultaneous Optimization Algorithm (NNSOA) is proposed for supervised training in multilayer feedforward neural networks. We demonstrate with Monte Carlo studies that the NNSOA can be used to obtain both a global solution and simultaneously identify a parsimonious network structure.

KEY WORDS: Artificial Intelligence, Backpropagation, Genetic Algorithm, Neural Networks, and Parsimonious [1]

[1] Randall S. Sexton, Computer Information Systems, Southwest Missouri State University, 901 South National, Springfield, MO 65804

1. INTRODUCTION

Although artificial neural networks (NNs) have been successfully applied in many areas of research, one major limitation to current NN practices still exists, which is the inability to identify unnecessary weights in the model. If a method were found that could search for a global solution and simultaneously identify unnecessary weights, managers would benefit from NNs that have better effectiveness and efficiency, as well as better understanding of the problem being solved.

1.1. Effectiveness

Effectiveness in this paper is the ability of the found NN solution to produce good estimates for observations that have not been used for training, or generalization. Generalization beyond the exemplars on which a neural network has been trained is a predominate concern for neural network applications. The improvement of generalization has been the topic of much research [2, 3, 10, 11, 13, 14, 15, 16, 21]. Since the majority of NN research relies on gradient search methods, usually some form of the gradient decent backpropagation (BP) algorithm, the NN is often limited in its ability to generalize. Two limitations imposed by these algorithms are local convergence and an inability to identify unneeded weights in the NN solution.

Gradient search techniques, like BP, converge locally by design. These algorithms search from a single point to another and gravitate toward the optimum along the path of steepest descent to the local solution. These types of algorithms work well for globally concave functions but are poorly suited for functions that have an unknown number of local optima. Local solutions are often characterized by a reasonably good fit in-sample but out-of-sample forecasts that are frequently unreliable. For even simple functions estimated with an NN, error surfaces can be remarkably complex, consisting of multiple local optima. To illustrate this point, an NN was used to estimate the following simple linear function.

$$Y = 3X_1 + 4X_2 \tag{1}$$

The network included 2 input nodes, 6 hidden nodes, 1 output node, and a bias term at the input and hidden node levels, totaling 25 weights for the NN solution. Two datasets of 50 exemplars were generated by randomly drawing X values from a uniform distribution in the range of [0, 1]. The first dataset was used for training and the second for testing. The NN was trained with the genetic algorithm for 1,000 generations, resulting in root mean squared error (RMSE) values of 3.39E-08 and 6.27E-08 for in-sample and out-of-sample, respectively. Although, the function being approximated by the neural network is simply a straight line, the neural network error surfaces can be quite complex. As an example, consider the above problem. Using the solution weights as a starting point, two weights from the input layer were chosen to vary by 0.001 increments for 50 points on each side of the original weight. All other input layer weights were held constant. The weights arbitrarily chosen to vary were the first two weights for hidden node 1. Although, only one combination of weights was chosen to vary for this illustration, the other combinations of weights show similar results.

For each of the 101X101 new weight combinations, the output layer weights were reoptimized given the input weight change and a new sum of squared error (SSE) was calculated. This procedure generated a total of more than 10,000 SSEs, which are plotted in Figure 1.

Figure 1 Approximately Here

This figure represents the error surface that was generated by varying only the first two weights of the problem. Each valley in figure 1 represents a local solution. Although the weights only varied by 0.001 increments it can be seen that multiple local optima are present. Notice the steep sides that are created, which could give gradient search techniques trouble in escaping. Algorithms that converge locally, such as BP, are likely to become trapped in these local solutions, resulting in solutions that perform inadequately when estimating out-of-sample. For complex problems that have multiple local optima, like a NN, finding the global optimum calls for an exhaustive search of the parameter space. This means that gradient algorithms would have to be restarted at a sufficiently large number of points throughout the parameter space in order to be confident that the global solution was found. Unfortunately, the number of necessary starting points to be chosen when using gradient search techniques is unknown and therefore the number selected is often not sufficient to ensure that a global solution has been obtained.

Overfitting or overparameterization also creates a problem with generalization. By overfitting we mean having too many parameters or weights. Overfitting is also used in the NN literature to mean allowing the objective function (i.e. SSE) to be reduced too much. Unneeded complexity, or dimensionality, in NN models has the adverse effect of allowing the network to overfit training data. By doing so, a trained network that has too many parameters will attempt to fit the particular data points in the training set including the errors rather than the underlying pattern of the data. At the extreme, a sufficient number of parameters could result in the network mapping each data point exactly. This would result in 100% in-sample accuracy but at the cost of losing the generalization needed to forecast new exemplars. For example, if a network had 20 inputs and only 10 of those inputs were relevant to the true model, ideally the contributions of the 10 irrelevant variables would be eliminated. If this is not done, the effect of these irrelevant variables on the in-sample forecasts may be minimal for in-sample data but could be significant for out-of-sample data. However, if those irrelevant variables are identified and the weights removed, they have no chance of affecting the final network estimates. Gradient algorithms, such as BP, are not designed to identify those parameters (weights) in the NN model that are unneeded for approximating the underlying function.

Typically, the addition of parameters will result in an improved in-sample fit. The question is always one of whether or not the improved fit justifies the additional complexity. In the statistical literature this question is known as model specification and a variety of tests have been identified to determine whether or not the additional parameters are justified. Unfortunately, specification tests do not exist for most nonlinear models including neural networks. The concept of the model specification tests can however be used as a guide for addressing the issue of complexity in NN architectures. These tests typically compare two models, one with more parameters and a superior objective function value and one with fewer parameters. For each new parameter that is added there needs to be a statistically significant improvement in the objective function to justify the additional degree of freedom used for the model.

This same concept is used for the NNSOA. At each stage of the optimization process weights can be added or removed. The criterion used for this decision is that a weight can only be added if it improves the objective function by a predefined amount and conversely, a weight will be removed if its removal does not deteriorate the objective function by more than the same amount. For example, for each solution in a generation, a weight can be added or removed by testing its effect on the objective function. Given a criterion, such as the modified root mean squared error discussed later, we can test to see if a weight should be added or removed. To do this, each candidate solution in turn, is plugged into the NN and the training data is ran through it, thereby calculating an objective function value for each solution. One weight in the solution is replaced with a hard zero. The objective function is then recalculated. If the objective function has increased less than the modified RMSE the zero remains, otherwise the previous weight is used.

As weights are added or eliminated during the optimization process, discontinuities are introduced into the objective function. This precludes using search algorithms that require a differentiable objective function and, in fact, preclude most standard hill climbing algorithms. To do the search we use a genetic algorithm.

There are other studies that have explored using algorithms to define an optimal architecture for a NN. For example there have been studies using gradient techniques that allow some of the connection weights to decay to zero or which reduce the size of the network during training [1, 2, 3, 13, 14, 21]. The problem with this approach occurs when trying to identify the correct weights or hidden nodes to remove from the trained NN. One technique focuses on the magnitude of the weights and eliminates weights that are close to being inactive. Although, elimination of these weights will reduce the complexity of the NN structure, any improved generalization capability is unlikely since the weights that are chosen are already arbitrarily close to zero. Further, this does not consider the possibility of the combined impact of several weights but only focuses on individual weights.

The other alternative is to remove active weights or hidden nodes and evaluate the impact. This is similar to a traditional statistical procedure where the model is re-estimated and the overall impact of the variable is tested. Thus, to evaluate the effect of reducing a trained NN's active connections requires the network to be retrained for the new architecture and in both cases it is essential that the global solution must be obtained for comparison.

1.2. Efficiency

Efficiency in this paper is concerned with finding NN models that include fewer weights and/or inputs that still perform well when estimating outputs. By identifying unnecessary weights, a parsimonious solution can be found. This can lead to more efficient networks that only rely on those weights and input variables that are justified by the data. As computer use intensifies, the increasing ease of data collection will result in larger and larger datasets with corresponding increases in potential input variables. Managers need to have tools that will help them to efficiently identify the appropriate model and still enable them to use better decision tools such as the NN. Once relevant inputs are determined, these datasets can be reduced to more manageable sizes. Although computer processor power is increasing the ability to eliminate unnecessary calculations, data storage will always be of great value to managers.

1.3. Understanding of the Problem

A NN solution that not only predicts well but can also determine those weights that are necessary can give managers a better understanding of the problem. Using normal NN research practices, a researcher will include every input variable that they feel could possibly help in predicting the defined output. Although these models might predict with a high level of accuracy in sample, the researcher still does not know which of the inputs are actually contributing to the prediction. By successfully identifying unnecessary weights in a network, additional information is derived by those connections that are still active. For example, the ability to successfully distinguish malignant tumors from benign tumors is of great value, but the additional information that only four out of the nine inputs actually contribute to the prediction, is extremely important information about the problem itself. One managerial drawback in using standard NNs is that they are often treated as black boxes. For given inputs, once trained, the NN provides outputs, without knowing how or why these estimates are true. By determining the relevant inputs in the model, a manager can now have a better understanding of the problem and will be better equipped in making decisions.

In this study a global search technique is used which allows a simultaneous search for an optimal structure as well as the solution best fitting the data. This results in an NN with a parsimonious structure and yet one with a close approximation to the data. A modified genetic algorithm (NNSOA) is used as the global search technique. The GA has been shown to significantly outperform BP for finding NN solutions [23] and further it has been shown to achieve superior solutions to other global search algorithms [24, 25]. The next section describes the GA. This is followed by a description of the modifications to the GA used in this study (NNSOA), the Monte Carlo study, results and conclusions.

2. THE GENETIC ALGORITHM

The GA is a global search procedure that searches from one population of solutions to another, focusing on the area of the best solution so far, while continuously sampling the total parameter space. The GA has recently been shown to perform exceptionally well, for out-of-sample data, at obtaining the global solution when optimizing difficult nonlinear functions [7]. An extension of this research has also shown the GA to perform well for optimizing the NN, another complex nonlinear function [23, 8, 9].

Research combining GAs and NNs began to appear in the mid to late 1980s. More than 319 references can readily be found in literature today [26]. The two primary directions of this past research involve using the GA to improve the performance of BP by finding optimal NN architectures and/or parameter settings, and as an alternative to BP for optimizing the network. Most recently, research has been conducted on simultaneous evolution of architectures and connection weights.

The GA is one of three major forms of a class of probabilistic adaptive algorithms based on the principles of biological evolution. This class is called evolutionary algorithms (EAs) and includes GAs, Evolution Strategies (ES) and Evolutionary Programming (EP). In each of these forms, better regions of the search space are found by means of stochastic process of selection, such as mutation and crossover by evolving initialized populations. GAs emphasizes crossover and mutation operators, while ES and EP emphasize the mutation operator [26, 12]. One topic of concern in evolving NNs is the choice of search operators used in EAs. Both crossover-based and mutation-based EAs have been used. However, use of crossover in past research has not produced consistently good results [26, 12]. However, it has recently been shown that the problem with this research is in the implementation of the GA and not its inability to perform the task [23]. There are several reasons for this. One example is that the majority of this past research encodes each candidate solution of weights into binary strings. This approach works well for optimization of problems with only a few variables but for NNs, with large numbers of weights, binary encoding results in extremely long strings. Given large strings of binary data, the patterns (Schemas) that are essential to the GA's effectiveness are virtually impossible to maintain with the standard GA operators such as crossover and mutation. It has been shown in empirical studies, that this type of encoding is not necessary, and is even detrimental [5, 17].

A more effective approach is to allow the GA to operate over real valued parameters. Examples of this approach can be found in [18, 23]. Although, Montana and Davis [18] successfully outperformed BP using the GA, their specific implementation of the genetic algorithm resulted in the crossover operator causing excessive loss of information about the Schema of the parameters. These Schemas were influential in the prior generation selection of the current generation's strings and therefore the loss of this informatio n reduces the effectiveness of the search process. Since the current population was drawn based on the prior generation's ability to perform, excessive loss of the prior generation's Schema (pattern) will slow down if not stop the convergence process. The alternative approach described in [23] also successfully outperformed BP on a variety of problems based on RMSE. The following is a simple outline of the basic GA operators. The following algorithm only uses the GA to search for the input weights. Prior research has found that using Ordinary Least Squares (OLS) for

Evaluation. Each member of the current population is evaluated by a fitness function based on their sum-of-squared error (SSE) value in order to assign each solution a probability for being redrawn in the next generation. The probability is calculated by dividing the distance of the current string's SSE from the worst SSE in the generation by the sum of all distances in the current generation.

Reproduction. A mating pool of 20 solutions is created by selecting solutions from the current population based on their assigned probability. This is done by selecting a random number in the range of 0 and the sum of all probabilities (or 1) and comparing it to the cumulative probability of the current string. When it is found that the random value is less than the current string's cumulative probability, the current string is drawn for the next generation. This is repeated until the entire new generation is drawn.

Crossover. The solutions in the mating pool are then randomly paired constructing 10 sets of parent solutions. A point is randomly selected for each pair in which the parent solutions will switch the weights that are above that point, generating 20 new solutions or the next generation.

Mutation. For each weight in a generation a random number is drawn, if the random value is less than .05, the weight will be replaced by a randomly drawn value in the entire weight space. By doing this, the entire weight space is globally searched enhancing the algorithm's ability to find global solutions or at least the global valley.

Convergence Enhancement. Once 70% of the maximum set of generations has been completed, the best solution so far replaces all the strings in the current generation. The weights of these 20 identical solutions are then modified by adding a small random value to the current weight. These random values decrease to an arbitrarily small number as the number of generations increase to its set maximum amount.

Termination. The algorithm will terminate on a user specified number of generations.

determining the output weights is more efficient and effective [23, 8, 7].

Unlike BP, which moves from one point to another based on gradient information, the GA simultaneously searches in many directions, which enhances the probability of finding the global optimum. A formal description of the basic algorithm used in this paper can be found in [7].

3. MODIFICATIONS TO THE GENETIC ALGORITHM (NNSOA)

The following modifications to the GA are used to create a method by which the GA can now simultaneously search for an optimal objective function value and a parsimonious architecture for the NN solution. These modifications to the standard genetic algorithm result in the proposed NNSOA algorithm.

3.1. Objective function

The most commonly used objective function in NN research is the quadratic loss function typically referenced as SSE or RMSE. This objective function is generally chosen due to the fact that it is well behaved and differentiable, thus allowing derivative based hill-climbing algorithms to be used for optimization. In this study an objective function was chosen that decreases as the error is reduced and as weights are eliminated. It is not a smooth function however, and is discontinuous as weights are added or removed. The GA has been shown to perform well with non-differentiable objective functions [7] and was therefore an optimal candidate for optimizing this modified objective function. The specific objective function used in this study is shown in Equation 2.

$$Min\, E = \sum_{i=1}^{N} \left(O_i - \hat{O}_i\right)^2 + C\sqrt{\frac{\sum_{i=1}^{N} \left(O_i - \hat{O}_i\right)^2}{N}} \tag{2}$$

Here N is the number of exemplars in the dataset, O is the observed value of the dependent variable, Ô is the NN estimate, and C is the number of non-zero weights in the network. The penalty for keeping an additional weight varies during the search and is equal to the current value of the RMSE. This means that the penalty for keeping additional weights is high at the beginning of the training process, however the errors are also high at this point. As the optimization process gets closer to the final solution the errors are decreased and the penalty value becomes smaller. This allows the training process to only eliminate those weights that have little to no affect on the prediction. A static penalty, on the other hand, would tend to dominate the objective function as the solution converges, thereby eliminating weights that are actually contributing to the prediction by over emphasizing the importance of removing a weight relative to improving accuracy. This objective function is arbitrary and was selected through trial and error. It works well for the problems in this study, but the identification of an optimal penalty is left for future research.

3.2. Mutation2

To allow the GA to identify irrelevant weights in the solutions, an additional GA operation (mutation2) was included. After mutation, and before the evaluation of the new generation occurs, each solution in the new generation has a small probability that any of its weights may be replaced with a hard zero. By doing so, when a solution is evaluated that has 1 or more of its weights replaced by a hard zero, the objective function can determine if this solution is better. If the solution results in significantly worse estimates, based on the objective function, it will die out in future generations. Once replication, crossover, mutation, and mutation2 have occurred, the new generation can now be evaluated to determine the new probabilities for the next generation.

3.3. Hidden Node Search

The number of hidden nodes included in each NN trained is automatically determined in the following manner. Each NN begins with 1 hidden node and trained for MAXGEN generations. For this study, MAXGEN was set to 1,000 generations. Once MAXGEN has been reached, the best solution at that point is saved as the BEST solution and an additional hidden node is included into the NN architecture. The NN is reinitialized by using a different random seed for drawing the initial weights and trained again for MAXGEN generations. On completion of this training the best solution for this architecture is compared with the BEST solution. If this solution is better than the BEST solution, it now becomes the BEST solution and is saved for future evaluation. This process continues until three consecutive hidden node additions finds no solution better than the BEST solution. Once this occurs, the BEST solution and its corresponding architecture is trained with an additional MAXGEN generations, which completes the training process. Although the search for additional hidden nodes is terminated after three unsuccessful hidden node additions, this is an arbitrary number. Additional research is warranted for finding an optimal number of failed attempts before terminating the hidden node search. This goes beyond the scope of this study and is left for further research. Bear in mind that even though there are two solutions that may achieve the same value of the objective function, they may differ in their architecture. Dorsey and Johnson [6] demonstrated that the neural network could have a variety of structures that will reduce to the same equivalent structure.

4. MONTE CARLO EXPERIMENT

In order to evaluate the performance of the proposed training algorithm, 11 problems were selected that have been used in previous NN research [23, 19, 22, 4, 20]. The first 10 functions are shown in Figure 2 and range from simple linear functions to complex time series and multidimensional production functions. The ranges from which the values for the input variables were drawn are also shown.

Figure 2 Approximately Here

An additional problem (*Building*) was included to test how well the NNSOA performed with real-world data as well as on problems with higher dimensions. This problem was taken from Prechelt's [20] paper where he collected a set of NN benchmark datasets. The *building* dataset was used for predicting energy consumption in a building and originally came from "The Great Energy Predictor Shootout – the first building data analysis and prediction problem" contest, organized in 1993 for the ASHRAE meeting in Denver, Colorado. This dataset included 14 inputs, 3 outputs, 2104 training exemplars, 1052 validation exemplars, and 1052 testing exemplars.

One important use for an algorithm that attempts to achieve the minimum necessary structure for a NN is to identify irrelevant variables by eliminating all of the connections to that variable. To test the NNSOA's ability for identifying irrelevant variables in the model, an additional irrelevant variable was added for each dataset. For example, problem 1 included 3 input variables that were uniformly drawn from the range [0, 1]. Only variables 1 and 2 were used in calculating the output. The third variable was included to test the algorithm's ability to discriminate between relevant and irrelevant variables. The irrelevant variable was drawn uniformly from [0, 1] for problems 1, 2, 5, 6, and 7. For problem 3 the irrelevant variable was drawn uniformly from [0, 2B] and from [0, 4B] for problem 4. For the time series problems 8 and 9 irrelevant variables were included in two ways, generating two different datasets. The first datasets for problems 8 and 9 were generated by including a variable with is value randomly drawn from the range N(0, 1). Results for these two problems incorporating the randomly drawn values are labeled 8a and 9a, respectively. In addition, an irrelevant time lag was used. For both problems an extra t-k lag was incorporated as an explanatory variable. Results for these two problems incorporating the irrelevant time lag are labeled 8b and 9b, respectively. Problem 10 incorporated an irrelevant variable drawn uniformly from the range [0,200]. The *building* problem also included an additional irrelevant variable that was drawn from a uniform distribution in the range of [0,1].

Ten training datasets and one test set were constructed for each of the first ten problems. Each dataset consisted of 100 exemplars. The datasets for the time series problems each began with the same point but differed in the randomly drawn error term. Ten datasets were also constructed for the *building* problem that differed in the randomly drawn values for the irrelevant variable.

4.1. Training with NNSOA

Parameters of the algorithm were set to values recommended by [7]. In using the NNSOA to search globally for a solution and optimal network structure, a modification of the error metric, SSEs is used to effectively identify extraneous connections in the NN structure, shown in equation 2. This modification adds a penalty for every non-zero weight in the solution. The size of the penalty is equal to the current level of the RMSE or the typical error of one observation. By setting the penalty equal to the RMSE we prevent the neural network from eliminating too many weights. As a result, a connection or weight is only eliminated if the SSEs is increased by less than the RMSE. This procedure allows the NNSOA to search simultaneously over both the parameter space and potential architectures. The number of generations used for this preliminary search was set arbitrarily, and may not be optimal. All optimization runs were conducted on a Pentium Pro 200 MHz machine.

4.2. Training with BP

The focus of this study is to explore the ability of the NNSOA to simultaneously find an optimal solution and a parsimonious structure for the NN. Since there is currently no method for determining if a solution is really optimal (or global), we can only compare the found solution with other potential solutions. In order to do this, we first need to demonstrate that the solutions found using the NNSOA are the best solutions. It is apparent that the value of an algorithm that can reduce the NN's structure is minimal if better solutions can be found using other standard training methods, such as BP. Therefore, a baseline comparison is needed to show that the proposed NNSOA can significantly and consistently outperform more standard NN training techniques. For this comparison, using the same datasets, commercial neural network software Neural Works Professional II/Plus by NeuralWare® was used.

Since, NNSOA attempts to simultaneously reduce the structure of the networks, two other comparisons were conducted on structure reducing techniques that used BP for training. These include pruning and a form of Cascade Correlation (CC) algorithm [11], which were both included in the Neural Works software package. The pruning technique is an example of a destructive algorithm.

A destructive algorithm basically starts with a large network and systematically deletes unnecessary nodes during training. The main problem with this technique is the assumption of the initial size of the network. Since we do not know what size of network that will sufficiently capture the underlying problem, we are merely making an educated guess based on the number of inputs and exemplars.

Constructive algorithms have several advantages over destructive algorithms in that they do not assume the initial size of the network. Since many networks with different sizes may be capable of implementing acceptable solutions, constructive algorithms are likely to find smaller network solutions than destructive algorithms. This leads to more efficient networks and possibly, solutions that have better generalization. CC algorithm is an example of a constructive algorithm, which starts out with a minimal structure and adds nodes until the network converges on a solution.

The specific variation of BP used for this study was QuickProp. An epoch is defined as one complete pass through the training set. The comparison with the standard BP algorithm included three different NN architectures that included 3, 6, and 12 hidden nodes in a single hidden layer feedforward NN. The pruning networks started with 20 hidden nodes and in every case reduced the networks to a structure that was less than 20. The CC algorithm networks had a maximum of 20 hidden nodes in which it could use. It should be noted that CC converged with fewer than 20 hidden nodes for every CC network. The "SAVE BEST" option was used for termination of all BP training. This option checked every 10,000 epochs for a reduction in in-sample error for the first 10 problems. For the *building* problem, this option checked every 10,000 epochs for a reduction in the validation dataset. This technique is used to help with "overfitting". NNSOA did not use the validation data for training, giving BP an added advantage. Once 10 consecutive checks were made without reducing the error, the training process terminated.

Since the NNSOA algorithm is a modified version of the GA found in [23], we wanted to show that the solutions found were not significantly different that those that would be found using the original GA. To do this, we included additional comparisons between NNSOA and the original genetic algorithm (OGA). Since the NNSOA algorithm automatically determines the number of hidden nodes and generations for each of its networks and the OGA does not, we needed to determine values for these parameters. To give the OGA the benefit of doubt, we set the number of hidden nodes, for all 10 OGA networks for each problem, to the maximum number of hidden nodes that was used by NNSOA for each problem. For example, the maximum number of hidden nodes used by NNSOA for problem 1 was 4 hidden nodes. The average number of hidden nodes for this problem was 2.3. We set the number of hidden nodes to 4 for all 10 OGA networks for problem 1. We also set the number of generations for the OGA to the maximum number of generations used by the NNSOA out of the 10 networks for each problem. By doing so, the OGA has sufficient complexity and enough generations to at least find solutions as good or better than the NNSOA.

5. RESULTS

To show the relevance of the NNSOA's ability to find parsimonious NN solutions, we first need to demonstrate its ability to find superior solutions, these results are shown in Tables 1 and 2. The average RMSE for all 10 runs for each problem was used for comparison of the results for both in-sample (Table 1) and out-of-sample (Table 2) datasets. These values are compared with the GA used in [23], standard BP using three different architectures, pruned networks, and CC networks for all problems.

Table 1 and Table 2 Approximately Here

As can be seen in these tables the NNSOA finds solutions that generate superior estimates. What is interesting, the OGA finds 5 out of 11 in-sample solutions that were better than the NNSOA, but none of the OGA solutions were better than the NNSOA solutions for out-of-sample. Given these results it is clear that by eliminating unnecessary weights in the solution that additional out-of-sample error can be avoided. Clearly, if additional generations of search were conducted, the OGA would be expected to find equal or superior in-sample solutions to the NNSOA for all problems. The reason that the NNSOA finds superior solutions is that by eliminating many of the weights, the search space is dramatically reduced, resulting in a much more rapid convergence. Although, it is apparent that the NNSOA outperforms all three BP configurations, including the pruned and CC networks, for in-sample and out-of-sample datasets based on RMSE, a statistical test is needed to show significant differences between these solutions. A statistical comparison of test data results was conducted using the Wilcoxon Matched-Pairs Signed Ranks (2-tailed P significance) test. To emphasize the superiority of the NNSOA training over BP, the worst NNSOA solution out of the 10 replications for each problem was directly compared with the best solution found from the 50 replications that were trained using BP. The selections of these solutions were based on in-sample RMSE and tested using the out-of-sample estimates generated by these solutions. The NNSOA solutions dominated the BP solutions in every test set at the 99% level of significance. Our results are consistent with other studies that have found that the GA can achieve solutions for the NN that are both accurate for in-sample data and sufficiently general to perform well out-of-sample.

The ability of the NNSOA to find parsimonious NN solutions is shown in Table 3. Ten training runs were conducted for each of the problems. The first three columns include the average number of generations, time for training, and number of hidden nodes in the final optimized NN. For example, in problem 1 the majority of the NNs had two hidden nodes. If you next consider all possible connections between the resultant hidden layer nodes and the input layer nodes, the algorithm eliminates many of these connections as well. This is shown in the next column. For problem 1, on average 55% of all possible connections between the hidden layer nodes and the input layer nodes were eliminated in the final optimized NN.

Table 3 Approximately Here

In problems 1, 2, 3, 4, 5, 6, 7,10 and *building,* all of the connections to the irrelevant variables were eliminated in each of the ten optimization runs. In the four remaining problems they were all eliminated in two of the runs for problem 8a, seven of the runs for problem 8b with the irrelevant lag, and one run each of the two versions of problem 9. The final column of Table 3 shows the percent of the weights associated with the irrelevant variables that were eliminated across all ten training runs. In total, across all twelve problems the NNSOA was able to successfully eliminate the irrelevant variables in 101 of the 130 training runs. It is interesting to note for the real-world *building* problem, the NNSOA not only identified the irrelevant variable that was included for this study, but it also identified 3 other inputs for all 10 runs that were also irrelevant for predicting the outputs. This type of identification can obviously help researchers in collecting future data as well as reducing the complexity of the model. For example in the *building* problem, the first seven inputs corresponded with the days of the week. If the observation was made on the first day of the week, the first input would be set to 1 and the other six inputs would be set to 0. If the observation was made on the second day of the week, the second input would be set to 1 and the other six inputs would be set to 0, etc… Although the NNSOA was found to produce better predictions, it also found that inputs 4, 5, and 6 (Wednesday, Thursday, and Friday) were not needed for prediction. What is interesting is that inputs 2 and 3 (Monday and Tuesday) were used in all 10 of the training sets. Inputs 1 and 7 (Sunday and Saturday) were only used 2 out of the 10 training sets. From this point, a researcher familiar with this problem can begin to analyze why Monday and Tuesday are important for this particular model as well as why Wednesday, Thursday, and Friday were not useful. The 13th input that contained the solar information was only used in 3 out of the 10 training sets. The remaining inputs, excluding the irrelevant random variable that was including, were required in most of the training sets. This is an example of how we gain knowledge of the problem by eliminating unneeded inputs. Although, this cursory analysis does not take into account multicolinearity for those inputs not included in all of the training sets, it does give researchers a place to start to better understand the problem at hand. However, inputs 3, 4, and 5 were not included in any of the training sets, so we can conclude that these inputs do not contain any useful information for prediction and we can better analyze those that infrequently occur, such as the solar variable, as being relevant.

In looking at the average CPU time, some may feel that the training efficiency is costly, especially for problems 4, 7, and the building. However, it should be kept in mind that this is only a one-time process and does not need to be repeated once the solution is found. Although, the experiments were conducted on a 200 MHz machine, which is slow by today's standards, the good solutions that were found and the additional knowledge gained by identifying relevant inputs justify this one time computation cost.

An example of a resultant NN is shown in Figure 3 to illustrate the weights that were eliminated for one of the training runs for problem 1. In this NN, the final optimized version consisted of three hidden nodes. Each of these nodes had the possibility of being connected to three input variables as well as a bias term. Each of the hidden nodes had only one of the four possible connections remaining when the optimization was complete and there were no connections to the irrelevant variable X_3. Therefore, of the twelve possible connections between the hidden nodes and the input layer only three remained after the optimization was complete.

Figure 3 Approximately Here

Additional runs for the *building* problem were conducted to evaluate the efficiency and effectiveness of NNSOA against a destructive method. Instead of automatically searching for the number of hidden nodes in the network, we set the number of hidden nodes to 18. We also trained the networks for 100,000 generations. Two sets of runs were conducted with these parameters. The first set reduced the network using the modified objective function and the second set did not (OGA). The approximate time for each network to terminate was 104 hours. Although these networks were significantly more complex than those that automatically determined network size and with many more generations, the results were not significantly different from the original NNSOA results. Not only did this destructive technique take longer, but the resulting networks for the reduction runs included more inputs than the original NNSOA networks.

6. CONCLUSIONS

The NNSOA is shown to perform well for optimizing a NN while simultaneously eliminating unnecessary weights in the NN structure during the training process. This results in a NN with a parsimonious structure that performs well both in-sample and out-of-sample. Three benefits that result from the ability to eliminate unneeded weights are the potential for better generalization, the identification of irrelevant variables, and reduction in the size of the model. By decreasing the structure of the NN, generalization is likely to improve because the shortage of weights forces the algorithm to develop general rules to discriminate between the input patterns, instead of memorizing special cases. An added benefit in identifying unneeded weights in the solution is the identification of irrelevant variables in the NN model. This identification gives researchers additional information about the behavior of the problem. Also, since unnecessary weights are removed from the model the NN is more efficient.

As can be seen from the results, the NNSOA correctly identified irrelevant variables in the datasets for a large percentage of the replications, including the real-world *building* problem, while still outperforming OGA and gradient training techniques. By identifying those inputs that contribute to the model's prediction, researchers gain additional knowledge about the data without a loss in performance.

Limitations in this study include two areas where future research is warranted. The first is the assignment of the penalty value. The current RMSE was added for every non-zero weight in the proposed solution. This value was set arbitrarily and is not proposed as an optimal value for unneeded weight identification. Future research should be conducted to identify better or optimal penalty values. The second limitation is in the identification of the number of hidden nodes included in the NN architecture. This study trained using 1,000 generations for each additional hidden node. This value was also set arbitrarily and has yet to be determined to be optimal. Although, these areas require additional attention, this paper has shown that the use of the NNSOA to simultaneously find parsimonious solutions is a viable one and should be considered for future NN research.

REFERENCES:

[1] E. B. Baum & D. Haussler, What size net gives valid generalization? Neural Computation, 1, 151-160, 1989.

[2] A. N. Burkitt, Optimisation of the architecture of feed-forward neural nets with hidden layers by unit elimination. Complex Systems, 5, 371-380, 1991.

[3] M. Cottrell, B. Girard, Y. Girard & M. Mangeas, Time series and neural network: a statistical method for weight elimination. In M. Verlysen (Ed.), European Symposium on Artificial Neural Networks (pp. 157-164). Brussels: D. facto, 1993.

[4] W. S. Chan & H. Tong, On tests for non-linearity in time series analysis. Journal of Forecasting, 5, 217-228, 1986.

[5] L. Davis (ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, NY, 1991.

[6] R. E. Dorsey & J.D. Johnson, Evolution of Dynamic Reconfigurable Neural Networks: Energy Surface Optimality Using Genetic Algorithms. Optimality in Biological and Artificial Networks. Daniel S. Levine and W.R. Elsberry Editors. Lawrence Erlbaum Associates. Nillsdale, N.J. pp. 185-202, 1997.

[7] R. E. Dorsey & W. J. Mayer, Genetic algorithms for estimation problems with multiple optima, non-differentiability, and other irregular features. Journal of Business and Economic Statistics, 13(1), 53-66, 1995.

[8] R. E. Dorsey, J. D. Johnson & W. J. Mayer, A genetic algorithm for the training of feedforward neural networks. Advances in Artificial Intelligence in Economics, Finance, and Management (J. D. Johnson and A. B. Whinston, eds., 93-111). Vol. 1. Greenwich, CT: JAI Press Inc, 1994.

[9] R. E. Dorsey, J. D. Johnson & M. V. Van Boening, The use of artificial neural networks for estimation of decision surfaces in first price sealed bid auctions. In W.W. Cooper and A.B. Whinston (eds.), New Direction in Computational Economics (pp. 19-40). Netherlands:Kluwer Academic Publishers, 1994.

[10] H. Drucker & Y. LeCun, Improving generalisation performance using double backpropagation. IEEE Transactions on Neural Networks, 3, 991-997, 1992.

[11] S. E. Fahlman & C. Lebiere, The cascade-correlation learning architecture. Neural Information Processing Systems, 2, 524-532, 1990.

[12] D. B. Fogel, A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems. Simulation, 64(6), 399-406, 1995.

[13] R. Kamimura, Internal representation with minimum entropy in recurrent neural networks: minimizing entropy through inhibitory connections. Network Computation in Neural Systems, 4, 423-440, 1993.

[14] E. D. Karmin, A simple procedure for pruning back-propagation trained networks. IEEE Transactions on Neural Networks, 1, 239-242, 1990.

[15] J. K. Kruschke, Distributed bottlenecks for improved generalization in back-propagation networks. International Journal of Neural Networks Research and Applications, 1, 187-193, 1989.

[16] G. G. Lendaris & I. A. Harls, Improved generalization in ANN's via use of conceptual graphs: A character recognition task as an example case. Proceedings IJCNN-90 (pp. 551-556). Piscataway, NJ: IEEE, 1990.

[17] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin, 1992.

[18] D. J. Montana & L. Davis, Training feedforward neural networks using genetic algorithms. Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 379-384, 1989.

[19] J. O. Moody & P. J. Antsaklis, The dependence identification neural network construction algorithm. IEEE Transactions on Neural Networks, 7(1), 3-15, 1996.

[20] L. Prechelt, A study of experimental evaluations of neural network learning algorithm: Current research practice, Technical Report 19/94, Fakultat fur Informatik, Universitat Karlsruhe, D-76128 Karsruhe, Germany. Anonymous FTP:/pub/papers/techreports/ 1994/1994-19.ps.Z on ftp.ira.uka.de, 1994.

[21] S. G. Romaniuk, Pruning divide and conquer networks. Network: Computation in Neural Systems, 4, 481-494, 1993.

[22] H. Schuster, Deterministic Chaos: An Introduction. Weinheim; New York: VCH, 1995.

[23] R. S. Sexton, R. E. Dorsey & J. D. Johnson, Toward a global optimum for neural networks: a comparison of the genetic algorithm and backpropagation. Decision Support Systems, 22, 171-185, 1998.

[24] R. S. Sexton, R. E. Dorsey & J. D. Johnson, Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. European Journal of Operational Research, 114, 589-601, 1999.

[25] R. S. Sexton, B. Alidaee, R. E. Dorsey & J. D. Johnson, Global optimization for artificial neural networks: A tabu search application. European Journal of Operational Research, 106, 570-584, 1998.

[26] X. Yao, Evolving artificial neural networks. Proceedings of the IEEE, 87(9), 1423-1447, 1999.

Table 1 - In-sample RMSE Comparisons for NNSOA and BP

Problem	NNSOA	OGA	BP3	BP6	BP12	Prune	CC
1	3.84E-09*	4.93E-05	4.55E-02	4.90E-02	4.77E-02	1.39E-01	8.22E-02
2	1.92E-06*	9.17E-05	5.71E-02	6.23E-02	7.00E-02	1.15E-01	8.96E-02
3	1.08E-03*	5.16E-02	6.77E-01	6.62E-01	6.65E-01	6.52E-01	6.60E-01
4	1.56E-04*	6.37E-03	3.23E-01	3.36E-01	3.49E-01	4.59E-01	4.42E-01
5	2.63E-02	2.38E-02*	2.66E-01	2.68E-01	2.77E-01	1.17E-01	1.07E-01
6	2.24E-07*	3.50E-05	3.03E-01	3.29E-01	3.57E-01	4.46E-01	4.85E-01
7	5.83E-02*	8.85E-02	6.59E+00	6.05E+00	6.53E+00	9.54E+00	7.13E+00
8a	8.30E-01	7.77E-01*	1.00E+00	9.66E-01	9.73E-01	1.05E+00	1.02E+00
8b	8.88E-01	8.51E-01*	1.05E+00	1.02E+00	1.07E+00	1.07E+00	1.06E+00
9a	8.87E-01	8.73E-01*	1.01E+00	1.04E+00	1.03E+00	1.25E+00	1.27E+00
9b	8.84E-01	8.75E-01*	1.30E+00	1.19E+00	1.23E+00	1.59E+00	1.07E+00
10	7.21E+00*	7.50E+00	1.44E+01	1.39E+01	1.42E+01	1.74E+01	1.44E+01
Building	5.42E-02*	5.79E-02	1.08E-01	1.10E-01	1.10E-01	1.42E-01	1.58E-01
* = Best algorithm average in-sample RMSE for problem							

Table 2 - Out-of-sample RMSE Comparisons for NNSOA and BP

Problem	NNSOA	OGA	BP3	BP6	BP12	Prune	CC
1	3.70E-09*	4.96E-05	3.86E-02	3.73E-02	3.69E-02	1.11E-01	5.77E-02
2	2.06E-06*	1.23E-04	5.37E-02	5.79E-02	6.59E-02	1.08E-01	8.50E-02
3	1.17E-03*	7.26E-02	7.26E-01	7.06E-01	7.16E-01	6.88E-01	6.76E-01
4	1.68E-04*	1.00E-02	2.24E-01	1.49E-01	1.06E-01	4.22E-01	4.03E-01
5	4.45E-02*	4.61E-02	3.08E-01	3.06E-01	3.09E-01	1.11E-01	1.03E-01
6	2.48E-07*	3.89E-05	3.10E-01	3.52E-01	3.74E-01	4.52E-01	4.91E-01
7	6.81E-02*	1.65E-01	6.64E+00	6.13E+00	6.62E+00	9.75E+00	7.22E+00
8a	9.31E-01*	1.05E+00	1.01E+00	9.63E-01	9.72E-01	1.05E+00	1.02E+00
8b	8.92E-01*	9.00E-01	1.10E+00	1.05E+00	1.10E+00	1.07E+00	1.06E+00
9a	9.81E-01*	1.01E+00	1.01E+00	1.05E+00	1.03E+00	1.26E+00	1.28E+00
9b	1.15E+00*	1.31E+00	1.32E+00	1.20E+00	1.26E+00	1.77E+00	1.16E+00
10	8.83E+00*	1.07E+01	1.74E+01	1.68E+01	1.73E+01	1.84E+01	1.63E+01
Building	5.43E-02*	5.78E-02	1.06E-01	1.09E-01	1.09E-01	1.43E-01	1.61E-01
* = Best algorithm average out-of-sample RMSE for problem							

Table 3 - NNSOA Training and Architecture Reduction Information

Problem	Avg. Generation	Avg. CPU Time in seconds	Avg. No Hidden Nodes	% of Zeroed Weights	Irrelevant Weights Eliminated	% Zeroed Irrelevant Weights
1	8,500	146.6	2.3	55%	10 of 10	100%
2	15,200	538.5	7.2	44%	10 of 10	100%
3	15,200	363.3	8.5	42%	10 of 10	100%
4	22,100	2827.0	14.7	47%	10 of 10	100%
5	12,400	312.3	5.5	46%	10 of 10	100%
6	12,100	884.7	6.2	42%	10 of 10	100%
7	16,600	2694.2	10.2	40%	10 of 10	100%
8a	15,100	585.1	7.3	47%	2 of 10	73%
8b	12,900	228.5	5.9	47%	7 of 10	93%
9a	13,100	398.9	5.6	36%	1 of 10	62%
9b	11,300	311.0	5.2	35%	1 of 10	50%
10	10,200	297.0	3.9	50%	10 of 10	100%
Building	13,700	2223.3	6.6	90%	10 of 10	100%

Avg. Generation: The average number of generations out of 10 replications for finding optimal NN architectures and function evaluations.

Avg. CPU Time in Seconds: The average CPU time in seconds out of 10 replications running on a 200-MHz Pentium Pro machine.

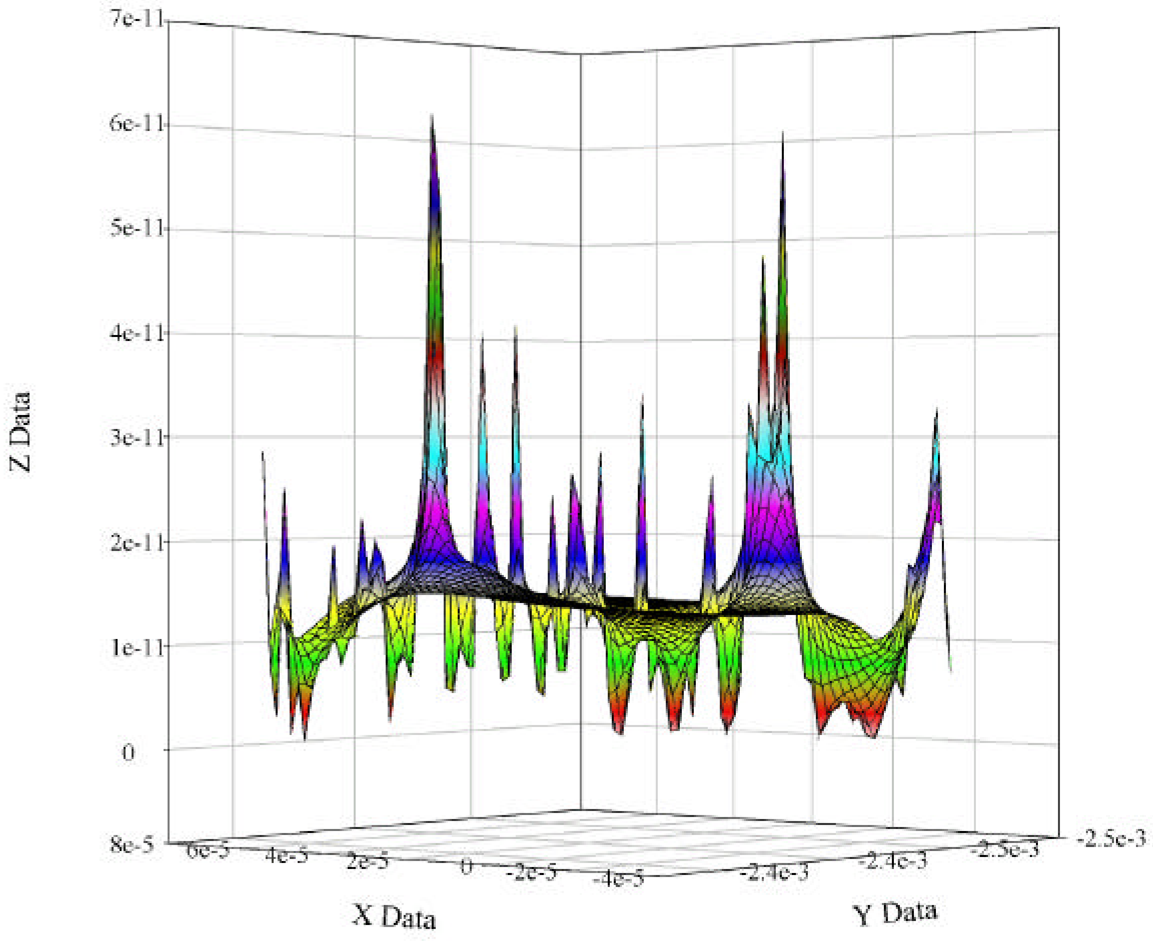
Avg. No Hidden Nodes: The average number of hidden nodes out of 10 replications for each problem found to be optimal.

% of Zeroed Weights: The average percentage of weights in the NN final solution that were valued at zero.

Irrelevant Weights Eliminated: The number of final solutions out of 10 replications that completely eliminated connections to the irrelevant variable.

% Zeroed Irrelevant Weights: The average percentage of irrelevant variable connections that were valued at zero in the final solution.

Figure 1 - 3D mesh plot of error surface



X Data refers to the first weight that is varied and Y Data is the second weight. Z Data is the value of the objective function.

Figure 2 - Problems

1. $Y = X_1 + X_2$ $\qquad X_1, X_2 \in [0,1]$
2. $Y = X_1 * X_2$ $\qquad X_1, X_2 \in [0,1]$
3. $Y = Sin(X)$ $\qquad X \in [0,4p]$
4. $Y_1 = \dfrac{1}{6[5Sin(X_1) + Cos(X_2)]}$ $\qquad X_1, X_2 \in [0,2p]$

   $Y_2 = \dfrac{1}{5[3Cos(X_1) + 2Sin(X_2)]}$
5. $Y = \dfrac{1}{10\left[e^{X_1} + X_2 X_3 Cos(X_1 X_2) + X_1 X_3\right]}$ $\qquad X_1 \in [0,1]; X_2, X_3 \in [-2,2]$
6. Henon Map

   $X_t = 1 - 1.4X_{t-1}^2 + Z_{t-1}$ $\qquad X_0 = 0.2, Z_0 = 0.2$

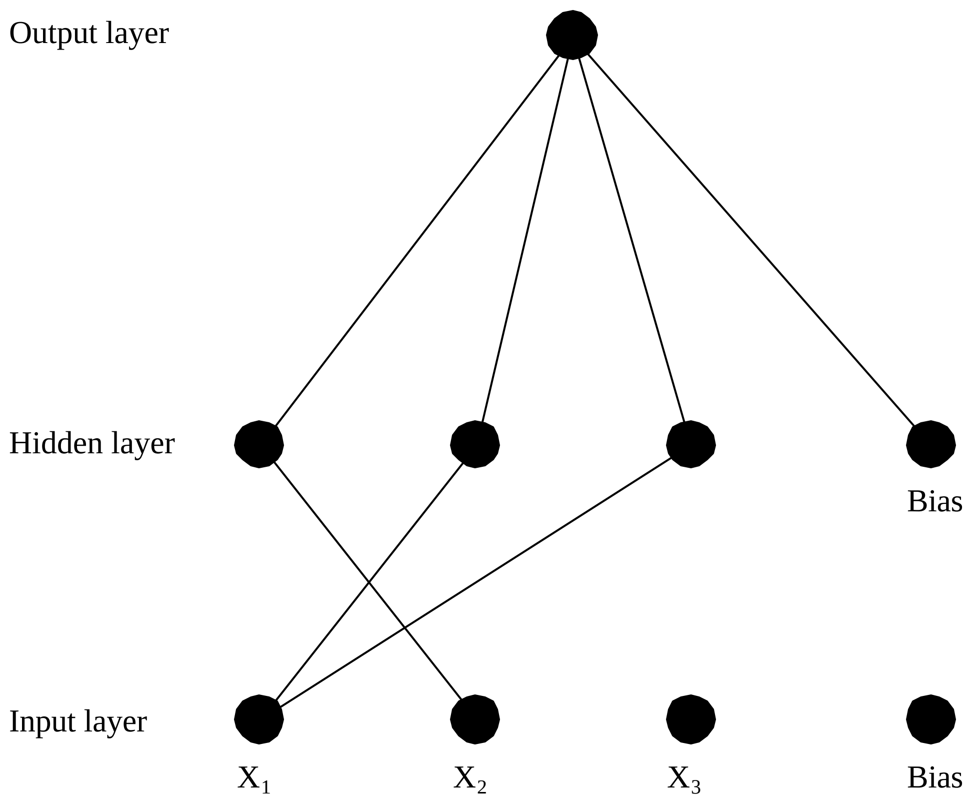
   $Z_t = 0.3X_{t-1}$
7. Lorenz Model

   $\dot{X} = -10X + 10Y$ $\qquad X_0 = 0.0, Y_0 = 0.01, Z_0 = 0.0$

   $\dot{Y} = -XZ + 28X - Y$

   $\dot{Z} = XY - \dfrac{8}{3}Z$
8. $Y_t = 0.4Y_{t-1} + 0.3Y_{t-2} + e$ $\qquad Y_{t-1} = 0.2, Y_{t-2} = 1.0, e_t \approx N(0,1)$
9. $Y_t = \left\{ \begin{matrix} 1 - 0.5Y_{t-1} + e_t, Y_{t-1} < 0 \\ -1 - 0.5Y_{t-1} + e_t, otherwise \end{matrix} \right\}$ $\qquad Y_{t-1} = 1.0, e_t \approx N(0,1)$
10. $Y = 5Arc\tan(X_1 - 50) + 10Arc\tan(X_2 - 100) + 15Arc\tan(X_3 - 150)$ $\qquad X_1, X_2, X_3 \in [0,200]$

Figure 3 - Reduced Neural Network Structure


Output layer
Hidden layer
Bias
Input layer
X_1
X_2
X_3
Bias