ACTUARIAL RESEARCH CLEARING HOUSE
1999 VOL. 1

The Inner Workings of Neural Networks and Genetic Algorithms*

Arnold F. Shapiro, J. Scott Pflumm and Thomas A. DeFilippo[1]
Penn State University

ABSTRACT

Genetic algorithms (GAs) and neural networks (NNs) are currently being used in actuarial and financial modeling. Nonetheless, while there is a general awareness of the nature of these techniques, there is often only vague familiarity with the details of how they are implemented. This article is intended to help alleviate this situation. Its purpose is to present an overview of GAs and NNs, which includes an explanation of what they are, how they work, and an example of how they are implemented.

INTRODUCTION

Actuaries generally have an awareness of the nature of genetic algorithms (GAs) and neural networks (NNs). Most know, for example, that GAs are based on genetics and evolution and NNs are based on the structure of the brain. However, there is often only vague familiarity with the details of how these techniques are implemented.

This is unfortunate. Many actuaries are confronted with problems where GAs and NNs are appropriate. These include problems which require a heuristic solution because of the vagueness of the underlying theory, and situations involving nonlinearities, where there is an emphasis on not making unjustified assumptions about the nature of those nonlinearities. Consequently, one would expect to see these techniques implemented more often.

A plausible explanation of why these techniques are not being used more often is that potential users are not sufficiently familiar with their characteristics and, consequently, forego opportunities for implementation. Assuming this to be the case, the purpose of this article is to help alleviate

---

[1]The authors are affiliated with the Penn State University. Arnold Shapiro is Professor of Actuarial Science and Insurance and Robert G. Schwartz Faculty Fellow. Scott Pflumm is a Research Associate of the Risk Management Research Center. Thomas DeFilippo was a student in the University Scholars Program.

Correspondence should be addressed to Arnold Shapiro at afs1@psu.edu.

*This work was sponsored in part by the Committee on Knowledge Extension and Research of the Society of Actuaries.

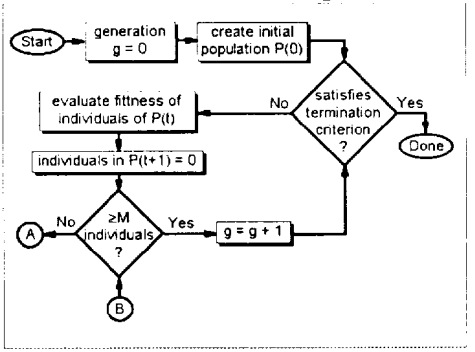
this situation by presenting an overview of GAs and NNs, which includes an explanation of what they are, how they work, and an example of how they are implemented.

GENETIC ALGORITHMS

Genetic algorithms (GAs) are automated heuristics that perform optimization by emulating biological evolution. They are particularly well suited for solving problems that involve loose constraints, such as discontinuity, noise, high dimensionality, and multimodal objective functions.

GAs can be thought of as an automated, intelligent approach to trial and error, based on principles of natural selection. In this sense, they are modern successors to Monte Carlo search methods. The flowchart in Figure 1 gives a representation of the process.

Figure 1
Flow Chart of Genetic Algorithm



As indicated, GAs are iterative procedures, where each iteration (g) represents a generation. The process starts with an initial population of solutions, P(0), which are randomly generated. From this initial population, the best solutions are "bred"with each other and the worse are discarded. The process ends when the termination criterion is satisfied.

As a simple example[2], suppose that the problem is to find by trial and error, the value of x, x = 0,1, ..., 31, which maximizes f(x), where f(x) is the output of a black box[3] [$f(x) = x^2$]. Using the methodology of Holland (1975), an initial population of potential solutions $\{y_j|j=1,...,N\}$ would be

[2]Adopted from Goldberg (1989), Chapter 1, and Vonk et. al. (1997), Chapter 3.

[3]The fact that the output of the black box is $f(x)=x^2$ is provided so that the reader can better follow and evaluate the process. This function, while adequate for this illustration, is not adequate for a general illustration, because the local maximum is the global maximum, so there is no chance for the process to get trapped in a suboptimal solution.

randomly generated, where each solution would be represented in binary form. Thus, if 0 and 31 were in this initial population of solutions, they would be represented as 00000 and 11111, respectively.[4] A simple measure of the fitness of y_j is $p_j = f(y_j)/\Sigma_j f(y_j)$, and it would be the solutions with the highest p_j's that would be bred with one another.

Table 1 summarizes these steps assuming the foregoing black box and an initial population of solutions of size four.

Table 1

Genetic Algorithm Worksheet

No.	Initial Population (Randomly Generated)	x Value	f(x)	$\frac{f_i}{\Sigma f_i}$	Expected count $4\frac{f_i}{\Sigma f_i}$
1	1 0 1 1 0	22	484	0.45	1.80
2	1 0 0 1 1	19	361	0.34	1.40
3	0 0 1 1 1	7	49	0.05	0.20
4	0 1 1 0 1	13	169	0.16	0.60
Sum			1063	1.00	4.00
Average			266	0.25	1.00
Max			484	0.45	1.80

The first step is to randomly generate the members of the initial population, which, in this case, are the binary strings (chromosomes) 10110, 10011, 00111, and 01101. Figuratively, each of these 20 genes could have been determined by flipping 20 coins and assigning each gene a value of "1" or "0," depending on whether the outcome was a "head" or a "tail." The equivalent random x-values are 22, 19, 7, and 13, respectively.

The black box [$f(x) = x^2$] provides the first set of potential solutions: 484, 361, 49, and 169. The sum, average, and maximum of which is 1063, 266, and 484, respectively.

Since the solutions are developed adaptively based on the observed data, and the goal is to maximize f(x), the relative fitness of each of these samples is assigned a value of $f(x_j)/\Sigma_j f(x_j) = f(x_j)/1063$. Thus, the relative fitness of each of the initial values are 45%, 34%, 5%, and 16%, respectively.[5]

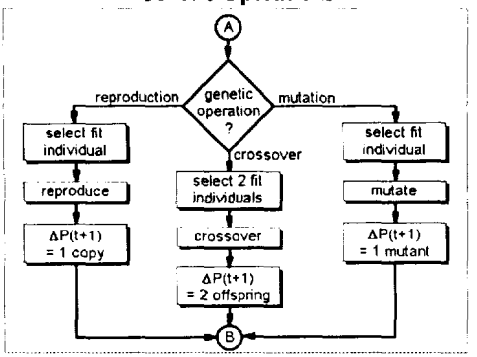
The second step is to use the current population of solutions to generate the next one. A flowchart of this process is depicted in Figure 2. As indicated, there are three ways to develop a new generation of solutions: reproduction, crossover, and mutation.

---

[4] $31 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$.

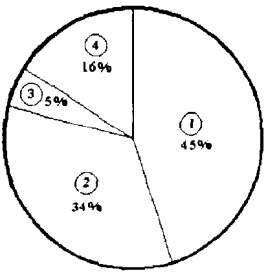
[5] These values have been adjusted to sum to 100%.

Figure 2
Genetic Operations



Reproduction adds a copy of a fit individual to the next generation. This is accomplished by randomly choosing a solution from the population, where the probability a given solution is chosen depends on its p_j value. An intuitive way to accomplish this is to use the notion of a weighted roulette wheel, as depicted in Figure 3, where the wheel can be unbiasly spun four times to randomly choose the next population.

Figure 3
Weighted Roulette Wheel



For example, assuming that outcomes of such spins resulted in the expected count, N p_j [Table 1, col. 6], rounded to the nearest integer, and adjusted to sum to N. Then, two samples would fall within area (1), one sample would fall within area (2), and one sample would fall within area (4). None of the samples would fall within area three. Table 2 shows the situation in this example after the first reproduction. Note that chromosome three has been replaced with a duplicate of chromosome one.

Table 2
Mating Pool After Reproduction

String No.	Initial Population (Randomly Generated)	Reproduced ?	# of times	Mating Pool after Reproduction
1	1 0 1 1 0	yes	2	1 0 1 1 0
2	1 0 0 1 1	yes	1	1 0 0 1 1
3	0 0 1 1 1	no	0	1 0 1 1 0
4	0 1 1 0 1	yes	1	0 1 1 0 1

Crossover emulates the process of creating children, and involves the creation of new individuals (children) from the two fit parents by a recombination of their genes (parameters). In the example, crossover would take place in two steps: first, the fit parents are randomly chosen on the basis of their p_j values; second, there is a recombination of their genes.

Table 3 depicts the process. If, for example, the randomly chosen fit parents were 10110 and 10011, and the randomly chosen crossover point was after the third gene, crossover would result in the two children 10111 and 10010. The improvement as a result of this iteration can be seen by comparing the last two columns of the table.

Table 3
Crossover

String No.	Mating Pool after Reproduction	Mate Randomly Selected	After Crossover	New x Value	f(x) new	f(x) old
1	1 0 1 1 0	2	1 0 1 1 1	23	529	484
2	1 0 0 1 1	1	1 0 0 1 0	18	324	361
3	1 0 1 1 0	4	1 0 1 0 1	21	441	49
4	0 1 1 0 1	3	0 1 1 1 0	14	196	169
Sum					1490	1063
Average					373	266
Max					529	484

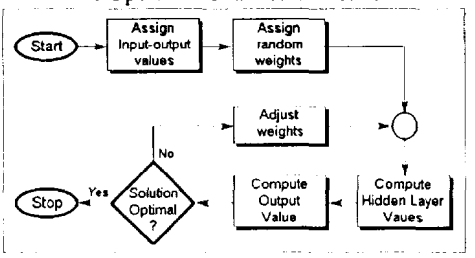
In the final step of the iteration, there is a small probability (~ 0.001) that a particular gene will be subject to mutation. That is, if its value is 0, it will be changed to a 1, and visa versa. This has the potential effect of introducing good gene values that may not have occurred in the initial population or which were eliminated during previous iterations.

NEURAL NETWORKS

Neural networks (NNs) are software programs that emulate the biological structure of the human brain and its associated neural complex and are used for pattern classification, prediction and financial analysis, and control and optimization.

A sketch of the operation of a NN is shown in Figure 4.

Figure 4
The Operation of a Neural Network



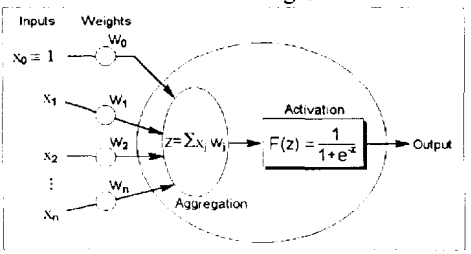
The case depicted involves supervised learning, so that both the inputs and output of the system are known,[6] and the objective is to find a relationship between them. The process begins by assigning random weights to the connection between each set of neurons in the network. These weights represent the intensity of the connection between any two neurons and will contain the memory of the network. Given the weights, the intermediate values (hidden layer) and output of the system are computed. If the output is optimal, the process is halted; if not, the weights are adjusted and the process is continued until an optimal solutions is obtained or a stopping rule is reached.

If the flow of information through the network is from the input to the output, it is known as a feed forward network. If inadequacies in the output are fed back through the network so that the algorithm can be improved, the NN is said to involve back-propagation.

Neural Processing Unit

The core of a NN is the neural processing unit, an example of which is shown in Figure 5.

Figure 5
Neural Processing Unit

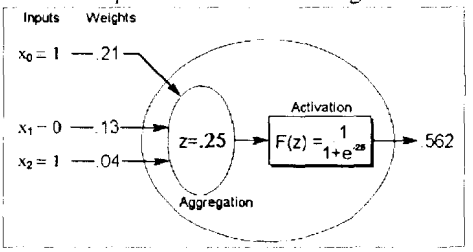


[6]Examples of known output include such things as firms that have become insolvent and claims which are fraudulent.

As indicated, the inputs to the neuron, x_j, are multiplied by their respective weights, w_j, and aggregated. The weight w_0 serves the same function as the intercept in a regression formula.[7] The weighted sum is then passed through an activation function, F, to produce the output of the unit. Typically, the activation function takes the form of the logistic function $F(z)=(1+e^{-z})^{-1}$, where $z = \Sigma_j w_j x_j$, as shown in the figure.

Figure 6 shows an example of how the neural processing unit is implemented. In this case, the input/weight pairs are (1,.21), (0,.13), and (1,.04), and the resulting output is .562.

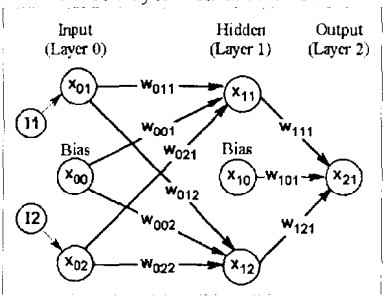
Figure 6
Example of a Neural Processing Unit



A Three Layer Neural Network

A NN is composed of layers of neurons, an example of which is the three-layer NN depicted in Figure 7. Extending the notation of the last section, the first layer, the input layer, has three neurons (labeled x_{0j}, j=0,1,2), the second layer, the hidden processing layer, has three neurons (labeled x_{1j}, j=0,1,2), and the third layer, the output layer, has one neuron (labeled x_{21}). There are two inputs I1 and I2.

Figure 7
Three Layer Neural Network



[7]Anders (1996) investigates neural networks as a generalization of nonlinear regression models.

The neurons are connected by the weights w_{ijk}, where the subscripts i, j, and k refer to the i-th layer, the j-th node of the i-th layer, and the k-th node of the (i+1)st layer, respectively. Thus, for example, w_{021} is the weight connecting node 2 of the input layer (layer 0) to node 1 of the hidden layer (layer 1). It follows that the aggregation in the neural processing associated with the hidden neuron x_{11} results in $z = x_{00} w_{001} + x_{01} w_{011} + x_{02} w_{021}$, which is the input to the activation function.

Consider the simple situation represented in Figure 8. There are two inputs, 0 and 1, and one output, 1, and the goal is to use a NN to reproduce this result.[8]

Figure 8
Exclusive-or Problem

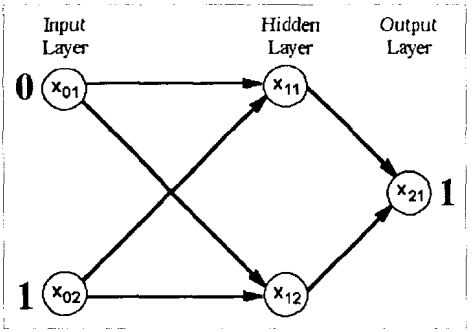
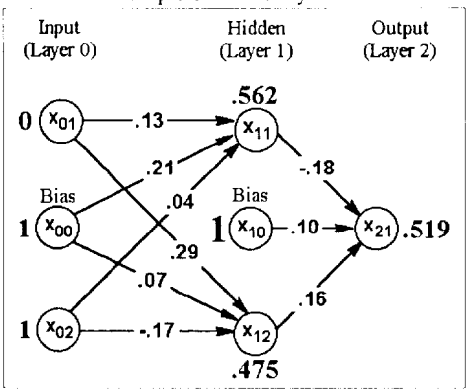


Figure 9 extends the approach begun in Figure 6 to a 3-layer NN. The value assigned to the hidden neuron x_{11}, .562, is the same one computed with respect to Figure 6. Similarly, hidden neuron x_{12} has the input/weight pairs (1,.21), (0,.29), and (1,-.17), and the resulting value is .475 Finally, the output, x_{21}, which is derived using the nodes in the hidden layer, has the input/weight pairs (1,.10), (.562,-.18), and (.475,.16), and the resulting output is .519.

[8] This is a portion of the exclusive-or problem, where x={0,1}, y={0,1}, xor(x,y) = 0, if x = y, and xor(x,y) = 1, if x <> y.

Figure 9
Example of Three-layer NN



A measure of the prediction error in this simple example is $(T-O)^2$, where T=1 is the targeted value and O is the output in a given iteration through the network.[9] After the first iteration, this prediction error is $(1-.519)^2 = 0.231$.

The Learning Rules

The weights of the network serve as its memory. Thus, the network "learns" when its weights are updated, and the general form of this learning can be expressed as

$$w_{ijk}(t+1) = w_{ijk}(t) + \Delta w_{ijk}(t)$$

where $w_{ijk}(t)$ is the weight during iteration t and $\Delta w_{ijk}(t)$ is the adjustment to the weight after the t-th iteration.

The adjustment is done using a learning rule, a common example of which is the Delta rule (Shepard (1997), p. 15)[10], given by

$$\Delta w_{ijk}(t) = \eta \, \delta_{ijk}(t) \, x_{ij}$$

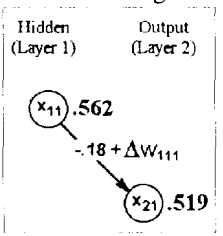
[9]The prediction error can be measured in a number of ways. See, for example, Anders (1996, p. 973) and Shepherd (1997, p. 5).

[10]Other types of learning rules are discussed by Vonk et. al. (1997), p. 12.

where η is the learning rate, which controls the speed of convergence[11], $\delta_{ijk}(t)$ is the error signal,[12] and x_{ij} is the value associated with the j-th node of the i-th layer. Since the case at hand involves three layers, including a single output layer, the adjustments are $\Delta w_{0jk}(t)$ and $\Delta w_{1jl}(t)$, the values are x_{0j} (the input values) and x_{1j} (the hidden values), and the learning rates are $\delta_{0jk}(t)$ and $\delta_{1jl}(t)$.

As an example, consider the revised weight associated with w_{111} (depicted in Figure 10). The revised weight is $w_{111}(t+1) = w_{111}(t) + \Delta w_{111}(t)$. The error signal in this case is $\delta_{111} = 0.120$. [13] Thus, since x_{11} is .562 and assuming the learning rate is 15 percent, Δw_{111} becomes 0.010[14] and the revised weight is -0.17 = -0.18+0.01.

Figure 10
Revised Weight



The Learning Strategy of a Neural Network

The characteristic feature of NNs is their ability to learn. The strategy by which this takes place involves training, testing, and validation, and is exemplified in Figure 11.[15] As indicated, the clean and scrubbed data is randomly subdivided into three subsets: T1, 60 percent, is used for training the network; T2, 20 percent, is used for testing the stopping rule; and T3, 20 percent, is used for testing the resulting network. The stopping rule reduces the likelihood that the network will become overtrained, by stopping the training on T1 when the predictive ability of the network, as

---

[11] If η is too large, the error term may not converge at all, and if it is too small, the weight updating process may get stuck in a local minimum and/or be extremely time intensive.

[12] Assuming the Delta rule, the error signals become $\delta_{1jl}(t) = F'(z_{1jl}(t))\,(T-O)$ and $\delta_{0jk}(t) = F'(z_{0jk}(t))\,\Sigma_j\,\delta_{1jk}(t)\,w_{ijk}(t)$, where F' denotes the differential with respect to z and $z_{ijk}(t) = \Sigma_i\, w_{ijk}(t)\, x_{ij}$.

Given the logistic form of the activation function, F'=F(1-F).

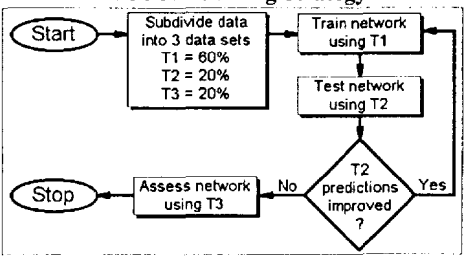
[13] (T-O)O(1-O) = (1-.519).519(1-.519) = 0.120.

[14] 0.010 = 0.15*0.120*0.562.

[15] This figure is based on a discussion of an application by Brockett *et. al.* (1994), p. 415.

measured on T2, is no longer improved.

Figure 11
The NN Learning Strategy



COMMENT

It seems inevitable that GAs and NNs will become significant tools for actuaries. Whether driven by the need for an unbiased solution, or forced to use a heuristic approach because of the vagueness of the underlying theory, or because of the increasing importance of the interaction terms, or simply because of the proliferation of user-friendly GA and NN software and high speed personal computers, the use of these techniques is likely to gain momentum. Hence, it is important for actuaries to be familiar with these techniques. If this article helps in this regard, it will have served its purpose.

SELECTED REFERENCES

Anders, U. (1996) "Statistical Model Building for Neural Networks," *6th AFIR Colloquium.*

Barber, J. C. (1995) "Genetic Algorithms as Tools for Optimization". *Risks and Rewards*, June.

Brockett, P. L., W. W. Cooper, L. L. Golden, and U. Pitaktong. (1994) "A Neural Network Method for Obtaining an Early Warning of Insurer Insolvency," *The Journal of Risk and Insurance*, pp 402-424.

Deboeck, G. Editor. (1994) *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for chaotic financial markets.* (New York: John Wiley and Sons)

Eberhardt, R. C. and R. W. Dobbins. *(1990). Neural Network PC Tools: A Practical Guide* (New York: Academic Press).

Forrest, S. (1996) "Genetic Algorithms". *ACM Computing Surveys*, March.

Gorman, R. P. (1996) "Current Modeling Approaches: a Case Study," Actuarial and Financial Modeling Conference, December 16-17, Georgia State University.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley.

Holland, J. H. (1975) Adaptation in Natural and Artificial Systems, Univ. Michigan Press, Ann Arbor

Lewinson, L. "GeneHunter-GA Software from Ward". *PC AI Magazine*, March/April.

Masters, T. (1993) Practical Neural Network Recipes in C++, *Academic Press*.

Shapiro, A. F., T. A. DeFilippo, K. J. Phinney, and J. Zhang (1998) "Technologies Used in Modeling," *ARCH* 1998.1, p. 47.

Shepherd, A. J. (1997) *Second-Order Method for Neural Networks*, Springer

Smith, M. (1993) *Neural Networks for Statistical Modeling*, Van Nostrand Reinhold.

Von Altrock, C. (1997) *Fuzzy Logic and NeuroFuzzy Applications in Business and Finance*, Prentice-Hall.

Vonk, E., L. C. Jain, and R. P. Johnson. (1997) *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*, Word Scientific