STATISTICAL APPLICATIONS OF NEURAL NETWORKS

Sangit Chatterjee

Matthew Laudato

Northeastern University
Boston, Massachusetts

November, 1995

STATISTICAL APPLICATIONS OF NEURAL NETWORKS

ABSTRACT

An elementary introduction to the theory of neutral networks is made from a statistical perspective. The networks are estimated by two different methods - one derivative based and the other derivative free. The theory is illustrated by three different examples and where possible results are compared to those obtained from a classical statistical model. The methodology is seen as a new paradigm for data analysis where models are not explicitly stated but rather implicitly defined by the network.

KEY-WORDS

Back-propagation, data analysis, feed forward, genetic algorithm.

STATISTICAL APPLICATIONS OF NEURAL NETWORKS

I. INTRODUCTION

One general problem in statistics is to estimate the parameters needed to fit a given model and norm to a data set. This is traditionally carried out by methods such as maximum likelihood, least squares, various nonparametric methods and others. These methods, however generally place constraints on the model and norm, so that we may only deal with functions that are continuous, differentiable and otherwise well-behaved. In addition, when a different model or norm is chosen, techniques of analysis must change accordingly, often increasing the complexity of the calculations.

We introduce the reader to a method of analysis independent of model and norm that uses an artificial neural network to construct a function that maps a set of N k-variate observations x to the associated univariate y's. Once the function is estimated, new observations can be analyzed without making any assumptions of model or norm. The restriction to univariate y's is only limited to our discussion in this paper. An artificial neural network, in its generality, places no such restrictions on the dimensionality of the y's.

Neural networks construct a function by *learning* (estimating) from repeated presentation of inputs (the x's) and outputs (the y's) and adjusting internal parameters so as to minimize the error between the fitted and desired y. The application domain of neural networks is broad, including handwriting analysis (Martin and Pitman, 1991), cancer drug research (Weinstein, et. al, 1992) and classification (Gallinari, et. al., 1991) In addition Ripley (1994) compares neural networks and linear and non-linear discriminant analysis, while Cheng and Titterington (1994) review the relationship between statistical methodology and neural networks. Our aim in this paper is to introduce the reader to the basic principles of neural networks and to present the results of using a network for estimating sinusoidal, exponential and logistic data.

The paper is arranged as follows. Section II describes traditional neural networks and their operation, and specifically describes the networks we use for this research. We also introduce in this section a nonstandard method of *training* (estimating) a neural network that employs a genetic algorithm instead of the more common calculus based methods. In Section III we discuss how a neural network can be used as a function estimator in place of standard statistical techniques. Section IV present the results of the function estimation for sinusoidal, exponential and logistic functions. Conclusions and a discussion of further directions that such research may take are presented in Section V. Throughout the paper we make a deliberate attempt to maintain the introductory nature of this work and refer the reader to the literature for more rigorous treatments of the mathematical elements involved.

II. WHAT IS A NEURAL NETWORK?

Figure 1 shows a typical neural network consisting of one input, one output and four hidden units. These units (or neurons, as they are sometimes called) are simple processors whose output is a function of the weighted sum of their inputs. In Figure 2, a typical unit is shown in more detail. The output of a neuron is in general a non-linear function (usually a sigmoid) of its input, and can depend on a parameter, the slope of the sigmoid. The arrangement of the neurons and their connections varies, and several network architectures are used in applications, the choice of architecture depending on the data available to train the network. The issue of network choice is further discussed in Section V. In this work we use a *multilayer feed-forward* architecture. Each layer receives input only from the layer directly preceding it, and adjacent layers are fully connected; that is, every neuron in a layer is connected to every neuron in an adjacent layer.

The presence of a hidden layer is a crucial aspect of the network. Features of the data set that normally would be characterized by a statistical parameter instead are represented by numeric connection weights to and from the hidden layer. The hidden units may be considered latent variables and the entire network architecture acts as an implicit function that is to be estimated. This way of thinking leads us to interpret a neural network as a device that implement statistical processes. This distributed representation has several advantages. The connection weights roughly represent how strongly the output of a unit should affect the activity of connected neurons . Because of the sigmoid function these feature mappings onto the hidden layer display a nonlinear sensitivity to both the weights and the observations. Also, since no one hidden unit is responsible for representing a particular feature of the data, neural networks exhibit *graceful degradation* in response to external (i.e., outliers) and internal (i.e. damage to the weights in a real-time system) perturbations (Rumelhart, et. al, 1989). The network remains robust in these situations.

To illustrate the operation of the network, let us follow a single observation from a set of N data points as it passes through the system. We indicate by the subscripts i, h and o the input, hidden and output layers of the neural network in Figure 1. (Note that Figure 1 depicts the case where the data are univariate in x and y, but the following treatment is more general.) The input to any neuron is labeled I and its output is O. The values of the independent variable for the chosen case I_i, normalized between 0 and 1, are presented to the input neurons on the left side of the diagram. If x is k-variate, then $i \in \{1, k\}$. Each observation is then processed by the network as follows. The input layer neurons serve to distribute the observation to the hidden layer, so the output of an input layer unit is $O_i = I_i$. This is an exception to the sigmoid behavior of the hidden and output neurons depicted in Figure 2. The input to the hth hidden layer unit is a weighted sum over all network inputs:

$$I_h = \sum_i W_{hi} O_i \qquad (2.1)$$

Note the unit labeled **B** in Figure 1; this is the bias unit, whose output is always equal to 1. The weights connecting the bias unit to the hidden and output units represent a threshold for these units; that is, the total input to the neuron must reach the threshold before the unit will have any substantial output. The bias unit serves as an intercept (or a constant) in conventional statistical models.

Thus far we have distributed the network input to all hidden units. The output of the hth hidden unit O_h is the sigmoid function applied to the input given in 2.1:

$$O_h = \frac{1}{1 + \exp(-\sum_i W_{hi} O_i)} \qquad (2.2)$$

Once the hidden layer has been activated, its output is passed to the output layer where it is processed to produce the network output for the current observation:

$$I_o = \sum_h W_{oh} O_h \qquad (2.3)$$

$$O_o = \frac{1}{1 + \exp(-\sum_h W_{oh} O_h)} \qquad (2.4)$$

We have just described one feed forward pass or *cycle* through the network. Given a proper set of connection weights, a network with this architecture can reproduce any arbitrary function of its inputs (White, 1990).

Neural networks can be seen as a general way to parameterize data through arbitrary nonlinear functions from the space of explanatory (causal, factor) variables to the space of response explained (response, dependent) variables. The existence of such functions are often attributed to researchers such as Cybenko (1989), Funahashi (1989) and others but the foundations of such thinking dates back to Kolmogorov. The mathematical basis for a neural network representation goes back to the work of Kolmogorov (1955) which is related to Hilbert's 13th problem of representing a function of several variables by a composition of a smaller number of variables. A later result of Kolmogorov (1957) establishes the theory on a more rigorous footing and may be stated as follows: Any continuous function of several variables may be represented by means of a continuous function of one variable and addition. Thus if z is a continuous function of a finite number of variables $x_1, x_2, .., x_s$ there exists continuous functions f, g, h, .. of a single variable y such that $z(x_1, x_2, .., x_s) = f(g(h(..(y)..)))$. Kolmogorov used his result to study asymptotic characteristics of totally bounded sets in function spaces which later found applications in various numerical methods including neural nets.

The great utility and flexibility of neural networks arises from the application of learning algorithms that allow the network to construct the correct weights, and hence the desired function, for a given set of observations. As the input-output vectors $(x_1, x_2, x_3, .., x_k, y)$ are presented to the network, a learning algorithm adjusts the connection weights until the system converges on a function that correctly reproduces the output. Several choices for learning algorithms have been discussed in the literature, although only a few have seen any practical applications. We use one traditional gradient method, the *backpropagation of errors* (Rumelhart, et. al., 1986), and one novel approach that uses a genetic algorithm to determine the optimal connection weights.

The goal of the learning algorithm is to adjust the connection weights after each cycle so as to minimize E, the sum squared error function of the network output.

$$E = \frac{1}{2} \sum_{n} \sum_{o} (O_o - y_o)_n^2 \qquad (2.5)$$

Here y_o are the known values of the dependent variables, the sum on n is over all observations and O_o is the output of the oth output layer neuron. Note that for the network in Figure 1, the sum on o collapses to a single term, but again we are giving a general treatment that applies to more complex networks. The constant $\frac{1}{2}$ simplifies later calculations. Criteria other than least squares have also been proposed in the literature (Ripley, 1994) including information theoretic measures such as the Kullback-Leibler distance.

The backpropagation of errors method takes advantage of the dependence of E on the weights, and calculates the appropriate partial derivatives to determine how E varies with respect to each weight. The weights are then modified in proportion to $\frac{\partial E}{\partial W}$ and, in our implementation, also in proportion to the weight change on the previous cycle. The weight change during the current network cycle c is given by:

$$\Delta W(c) = -\theta \frac{\partial E}{\partial W} + \phi \Delta W(c-1) \qquad (2.6)$$

where θ and ϕ are parameters described below, and $\Delta W(c-1)$ represents the weight change applied during the previous cycle. The reader is referred to Rumelhart, et. al. (1986, 1989) for the details of the calculations.

The training time of a backpropagation networks (how many presentations of the data are required for convergence) tend to be long; this has lead several researchers to modify the basic algorithm to adjust the *learning parameter* θ and the *momentum parameter* ϕ. Vogl, et. al. (1988), for example, report some success by increasing the value of θ when the network shows no improvement after a specified number of cycles, while Hirose, et. al. (1991) vary the number of hidden units if E does not change by more than 1% after 100 weight changes. Rigler, et. al. (1991) rescale the variables to compensate for the slowness of the steepest descent. In this work we opt for simplicity and use a static network that modifies the connection weights only.

The backpropagation method has several drawbacks, the most significant due to the tendency of gradient based methods to become trapped in local optima of the error surface. The variations by Vogl and Hirose are attempts to "jiggle" the network and free it from the local minima, but the if-then nature of such methods introduces additional complexity in the computer code. Whittle (1994) in his discussion of

Ripley's paper comments " .. a point that should be made is that the standard improvement step, the back-propagation algorithm has its limitations. If the problem exceeds a certain size, as measured by the dimension of its input, then back propagation simply chokes and refuses to work." Concerns for the algorithm getting stuck at a local minima are also expressed by Brieman (1994) and others. We propose a second method for training a neural network based on genetic algorithms that has the potential to overcome this limitation.

The genetic algorithm (GA), developed by Holland (1975, 1992) and advanced by Goldberg (1989) and others, is an optimization technique based on principles of population biology. In a GA, a set of candidate solutions to a problem undergo evolution and reproduction in order to arrive at progressively better solutions. Solutions are selected for reproduction by a fitness criteria (generally the value of the norm for the chosen model) in such a manner that the average fitness of the population increases over time. The convergence of the algorithm results from the formation of *schema*, which are local optimal patterns within the representation. Solutions that survive to reproduce contain favorable schema, which are passed on to future generations.

The representation of candidate solutions depends on the problem domain; for problems requiring real number solutions, binary integers are used and mapped onto R. Binary representations allow for the definition of genetic operators that create new integers by mating a pair of integers, or by modifying the bits within a single integer. Figure 3 illustrates the genetic operator *crossover* acting on two n-bit integers. The head of the first is combined with the tail of the second and vice-versa; since the resulting objects are also integers, they represent two new (and potentially more fit) solutions to the problem. Combined with *inversion*, which reverses a substring of an integer, and *mutation*, which applies the NOT operation to individual bits, populations of solutions can evolve towards the desired optimum. For the neural network in Figure 1, there are 13 connection weights, so the problem is to find the set of weights the minimizes the error E in equation 2.5. A candidate solution is thus represented in computer memory as an array of 13 binary integers, and typically 1000 such sets are chosen at random to populate the algorithm. In essence, we are beginning with 1000 separate neural networks and allowing the weights to evolve into a single best network.

The algorithm proceeds by choosing two solutions at random and comparing them for fitness; that is, running the data through the neural net formed by the 13 weights and calculating E. The better of the two solutions has two copies of itself placed in the next generation with typically 75% probability. This is repeated until a new population has been created. The reproduction phase begins by choosing two networks at random from the new population and allowing their members (in this case, the integers corresponding to a particular weight in the network) to crossover. This continues for all members of the two solutions, that is, until all weights in the two candidate networks have mated with their counterparts. The result of this process is two new networks that share characteristics of both parents and which may prove more fit than either parent. To finish the reproductive cycle, solutions are again chosen at random until some fixed percentage (again typically 75%) of networks have mated. Inversion and mutation proceed similarly during each generation, which helps the algorithm to avoid local minima in E.

The two learning algorithms are quite different from one another, but we cannot resist mentioning that both are based, albeit loosely, on ideas borrowed from the natural and cognitive sciences. The neural network itself is a cognitive model that allows features of an environment to be learned and recalled by a system that is self-organizing. The calculus based backpropagation algorithm derives from the original Hebbian learning rule (Rummelhart et. al, 1989), which states that if two units in a pattern associator are both excited by an input, the connection strength between them should be increased. Genetic algorithms on the other hand are population models that similarly adapt to an environment by organizing their members into better solutions to the problem at hand. Neural networks and the GA have been successful in their respective problem domains, both of which include optimization and estimation. We believe that the combination of these techniques and their application to statistical estimation is a first step towards creating a general analysis tool, purely computational in nature and free of model and norm constraints.

III. FUNCTION ESTIMATION USING A NEURAL NETWORK

Although several application domains have been mentioned thus far, we are primarily interested in using the neural network to perform statistical analysis. Normally one might pass a data set through a calculus based algorithm and estimate parameters for the chosen model. The parameters are then used to construct the fitted line or curve, error measures are estimated, and the analysis is considered complete.

While this is certainly an oversimplification, we mention the traditional method to illustrate that our approach here is quite different. The parameters we estimate (the connection weights) do not have any one-to-one relationship with the usual statistical parameters associated with a model. There is no weight in the network that uniquely represents the slope of a line, or the asymptote of an exponential, for example. The fit function is built with no model assumptions whatsoever; this is highlighted by the fact that identical networks were used to estimate sinusoidal, exponential and logistic data sets that we describe further in the next section.

The methodology used to estimate functions with the neural network discussed in Section II is relatively straight forward. The software was designed and developed by the authors to handle the general case of multivariate data, but for this research our data sets each had univariate x and y. During each cycle of the network, all N data points are presented in sequence as input-output pairs to the network. The input is the value of x_n for the current observation, so that $I_i = x_n$. This value is passed through the feed-forward network to generate the network output O_o. The sum squared error in equation (2.5) is calculated using the current values of $y_{n.}$ and O_o and is added to a running error sum. After the entire data set has passed through the network, the value of total E is known, and the learning algorithm is invoked to adjust the weights. This ends the current cycle. For a typical small data set of 100 (x_n, y_n) pairs, a cycle thus consists of passing all 100 points through the network, accumulating E, and adjusting the weights. The process is repeated until the network converges or until a fixed number of cycles have passed. (This method of processing the data is often call 'batch' mode as opposed to adjusting the weights after each observation enters the network which is called 'on-line' mode. In the latter mode, the minimized error may depend on the sequence of observations and care has to be taken to remove any such dependence.)

We typically choose a convergence criteria of 10%; that is, the network has converged when its output for each observation is within 10% of the known value. For backpropagation training, the values of the learning and momentum parameters are typically $\alpha = .25$ and $\beta = .04$. These values are relatively stable and have produced consistent results in our simulations. In the genetic algorithm training method, we allowed the GA to search for weights on {-15, +15}, which was again chosen from experience with the networks; weights rarely fall outside of this range. Also we used a population size of 1000 candidate networks, with 75% of the population undergoing crossover, 65% experiencing inversion, and 1 in 1000 bits being mutated for our GA training. Our research into the properties and applications of the GA (Chatterjee and Laudato, 1993, 1994) has led us to believe that these values are robust for problem domains with low to medium complexity.

When the network has been sufficiently trained, the weights are saved to a file so that the network can be recreated at any time in order to classify any new observations obtained. The network then behaves in some sense as a "black box" that is fed input and produces output. We then run the network by passing the x values through and generating the fitted y's. The curves produced by this method are the desired function estimations, which are compared in the next section to the results from traditional statistical analysis.

IV. PRESENTATION OF RESULTS

In this section we present in graphical and tabular form the results of several runs of the neural network, for three different data sets. The first example generates data from a deterministic periodic function and trains the network. The second example analyzes a set of observations that is best fitted by a parametric model of truncated exponential form and the results are compared with the neural network estimates obtained thorough the two learning algorithms. The third example is a modeling application where a continuous variable is mapped onto a discrete response variable, the parametric form of which is represented by a classical logistic regression model. In each case network architecture is that of Figure 1. The parameters used in the training of the network are given in Table 1. Two measures of fit, namely the sum of squares deviation (Ssq.dev) and the Sum of absolute deviation (Sad), obtained from the neural network using the backpropagation and the genetic algorithm for the three data sets are given in Table 2.

IV.1. The Sin Function: A Deterministic Example

We begin by generating 1000 data points from the function $y = 0.5 + \sin(2\pi x)$, 0 _ x _ 1. The network is estimated using both the backpropagation and the genetic algorithm using the network presented in Figure 1. The data, estimates obtained from backpropagation (Neu fit) and the genetic algorithm (GA fit) are plotted in Figure 4. The fit is excellent for the entire function though both methods fail to reproduce the data at the peak and the trough. The GA fit is somewhat superior to the back propagation fit at the turning points of the function. As can be seen from Table 2, the GA least squares fit is somewhat superior to that of the back propagation method. The sum of absolute deviations are given for comparison purposes.

IV.2. Swimming Data: A nonlinear Least Squares Regression Problem.

Given data (y_t and t, t = 1, 2, .. , T) we are interested in neural network estimates of the data and compare it with fits obtained from a parametric model given by $y_t = \alpha + \beta \exp(-\gamma t) + \varepsilon_t$. The data set (Chatterjee and Laudato, 1994) is the ratio of women's world record time to men's world record time at time t from 1923 (t = 1) to 1992 (t = 69) for 800 meters free-style swimming. The asymptote ($y_\infty = \alpha$, as $t \to \infty$) indicates the ultimate performance ratio of the women's to men's time for the event. We use the SYSTAT software to produce the conventional statistical solution which uses a Newton-Raphson type algorithm for steepest ascent.

The SYSTAT estimates of the model (with estimates of standard errors in parenthesis) are given by

$$\hat{y}_t = \underset{(0.10)}{1.07} - \underset{(0.10)}{0.24} \exp(\underset{(0.62)}{-1.02}\, t) \quad .$$

A plot of the data along with the fitted curve is given in Figure 5. The estimates of the ratios obtained from using the GA and the back propagation algorithms are given in Figures 6 and 7. The similarity in the estimates of the three figures are striking. The sum of squares deviation of the back propagation and the GA are similar though the corresponding quantity obtained from the parametric model is much larger. This difference can be understood by viewing the parametric model as a constraint on the data and hence the minimized sum of squares are free to vary for the three models.

IV.3. CHD Data: Logistic Regression.

The last example deals with discrete binary data modeled as a function of a continuous variable. The data is from Hosmer and Lemeshow (1989) where the explanatory variable is the age (in years) of a random sample of 100 people and the dependent variable is 1 or 0 whether they have coronary heart disease (CHD) or not. In the parametric model we estimate the parameters of a logistic linear regression model given by

$$\pi(x) = \frac{e^{\beta_o + \beta_1 x}}{1 + e^{\beta_o + \beta_1 x}} \qquad (4)$$

and the log-likelihood to be maximized is given by $L(\beta_0, \beta_1) = \sum_{i=1}^{n} \{y_i \ln[\pi(x_i) + (1 - y_i)\ln[1 - \pi(x_i)]\}$ where y_i is 1 or 0 with probability $\pi(x_i)$ is given by (4). The estimated probabilities $\hat{\pi}(x)$ are obtained from the SYSTAT software and are plotted in Figure 8. The corresponding estimates obtained from the backpropagation and the GA are plotted in Figures 9 and 10. The total sum of squares of errors and the total sum of absolute deviations are also given in Table 2. As in the swimming example, the total sum of squares errors obtained in the parametric model are larger than that obtained from either of the two methods used in estimating the neural network. However, all three methods yield very desirable fits.

V. DISCUSSION AND CONCLUSION

Neural networks or artificial neural networks are a very flexible way to model data for both function estimation and classification purposes. Different networks can be made to represent various statistical models that include projection pursuit methods, generalized additive models, various parametric and nonpararmetirc models for classification of data and others.

The problem of model selection and the functional form of the model in the chosen variables are a difficult problem in data analysis and statistics. The problem of model selection translates in to a choice of architecture for the neural network. The choice of the functional form for the explanatory variables does not arise explicitly in the neural network architecture, the presumption being the internal weights and the sigmoid functions automatically take care of the relevant specifications. The choice of networks among a finite number of alternatives should be guided by similar philosophy that guides variable selection. Too large a network (that is, too many hidden units) runs the danger of overfitting while a network with too few hidden units produces biased fits and prediction. The Occum's razor principle of parsimony clearly applies in network choice. Quantitative principles of Akaike (1974), Schwartz (1978) and Risannen (1987) can be used for network choice though the amount of calculations for considering various alternatives can get to be very high. The issue of model validation is of paramount importance in neural network methodology and has to be done on a data set that is not part of the estimation set. Similar problems occur in conventional statistical methodology but the problems are less acute since other assumptions about the data set, such as Gaussianity can be checked through other diagnostics that are routinely calculated. In the absence of an explicit statistical model, the prediction ability of the network, called its generalizability, is the only true measure of the effectiveness of a neural network.

It appears, superficially at least, as if the methodology of neural networks absolve the statistician / data analyst of important responsibilities but this is very far from truth. Neural networks are clearly not a panacea and should not be considered as an alternative to the usual statistical models but rather another powerful tool added to our disposal. Neural networks are most useful when massive quantity of very high dimensional data need to modeled without any proper model specification available a-priori. The current status of neural network modeling is clearly a mixed bag -- many successful attempts along with some disappointments. As the ability of hardware and software increase in the next decade and with more attention from statistical, data and numerical analysis communities, the true worth of these techniques will become apparent.

REFERENCES

Akaike, H. (1974). Information theory and an Extension of the Maximum Likelihood Principle. In *Second International Symposium on Information Theory,* Editors B. N. Petrov and F. Csaki, 261-281, Akademia Kiedo, Budapest.

Brieman, L. (1994). Comments to Cheng and Titterington (1994). *Statistical Science*, Vol. 9, No. 1, 38-42.

Chatterjee, S. and Laudato, M. and Lynch, L. A. (1994). Genetic Algorithms and Their Statistical Applications: An Introduction. Preprint.

Chatterjee, S. and Laudato, M. (1993). Genetic Algorithms in Statistics: Procedures and Applications. Preprint.

Chatterjee, S. and Laudato, M. (1994). *Gender and Performance in Athletics, Social Biology* (Forthcoming).

Cheng, B and Titterington, D. M. (1994). Neural Networks: A Review from a Statistical Perspective. *Statistical Science*, Vol. 9, No. 1, 2-54.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal function. *Mathematical Control System*, 2, 303-314.

Funahashi, K. (1989). On the Approximate Realization of Continuous Mappings by Neural Networks. *Neural Networks*, 2, 183 - 192.

Gallinari, P., Thiria, S., Badran, F. and Fogelman-Souile, F. (1991). On the Relatins Between Discriminant Analysis and Multilayer Perceptrons. *Neural Networks*, 4, 349-360.

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

Holland, J. M. (1975). *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor.

Holland, J. M. (1992). *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, MA.

Kolmogorov, A. N. (1955). Estimation of Minimal Number of Elements in $\varepsilon - \text{nets}$ in Various Functional Classes and their Applications to the Problem of Representability of Functions of Several Variables by Superpositions of Functions of a Smaller Number of Variables. *Dokl. Akad. Nauk SSSR*, 101, 192-194. (In Russian).

Kolmogorov, A. N. (1957). On the Representation of Continuous Functions of Several Variables by Means of Superpositions of Continuous Functions of One variable and Addition. *Dokl. Akad. Nauk SSSR*, 114, 953-956. (In Russian)

Martin, G. L. and Pittman, J. A. (1991). Recognizing Hand-Printed Letters and Digits Using Backpropagation Learning. *Neural Computation*, 3, 258-267.

Rigler, A. K., Irvine, J. M., Vogl, T. P. (1991). Rescaling of Variables in BP learning, *Neural Networks*, 4, 225-229.

Ripley, B. D. (1994). Neural Networks and Related Methods for Classification. *J. Royal Statistical Society B*, 56, No. 3, 409-456.

Rissanen, J. (1987). Stochastic Complexity (with discussion). *Journal of the Royal Statistical Society*, Series B, 49, 223-239.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323, October, 9, (533-546).

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1989). *Parallel Distributed Processing*, Volume 1, The MIT Press, Cambridge.

Schwartz, G. (1978). Estimating the Dimension of a Model. *Annals of Statistics*, 6, 461- 464.

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., Alkon, D. L. (1988). Accelerating the Convergence of the BP Method, *Biological Cybernetics*, 59, 257- 263.

Weinstein, J. N., Kohn, K. W., Grever, M. R., Viswanadhan, V. N., Rubenstein, L. V., Monks, A. P., Scudiero, D. A., Welch, L., Koutsoukos, A. D., Chiausa, A. J., Paull, K. D. (1992). Neural Computing in Cancer Drug Development: Predicting Mechanism of Action. *Science*, 258, 16 October 1992, 447-451.

White, H. (1990) Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings. *Neural Networks*, 3, 535-549.

Whittle, P. (1994). Discussion of the paper Ripley (1994). *J. Royal Statistical Society B*, 56, No. 3, 437-438.

Table 1*

	Back Propagation			Genetic Algorithm
PROBLEMS	a	b	Tolerance	Pop Size
Sin Function	0.25	0.04	0.05	100
CHD Data	0.14	0.04	0.25	500
Swim Data	0.25	0.04	0.01	1000

*The number of cycles for the back propagation algorithm are 10,000 while the proportions for XOVER, INV, MUTATE, SELECT are 0.75, 0.25, 0.001 and 0.75 respectively for all three problems .

Table 1. The parameter settings to run the neural network using both the back propagation and the genetic algorithms to run all three problems are given.

Table 2*

Sin Function			
	Back Prop	GA	SYSTAT
Ssq.dev	0.2544	0.1362	Not Applicable
Sad	4.3131	4.4883	Not Applicable
Swimming Data			
Ssq.dev	7.0864 E-3	7.1623E-3	4.6479E-3
Sad	0.8107	0.8026	0.9328
CHD Data			
Ssq.dev	8.9051	8.9050	17.8199
Sad	35.57	35.66	35.42

* Ssq.dev: Sum of squares deviation, Sad: Sum of absolute deviation, SYSTAT: Solution obtained from the commercial software SYSTAT Inc.

Table 2. Two different fit criteria obtained for the neural network through the back propagation and the genetic algorithm are given for the three data sets. An estimate from a parametric model, where applicable is also given.

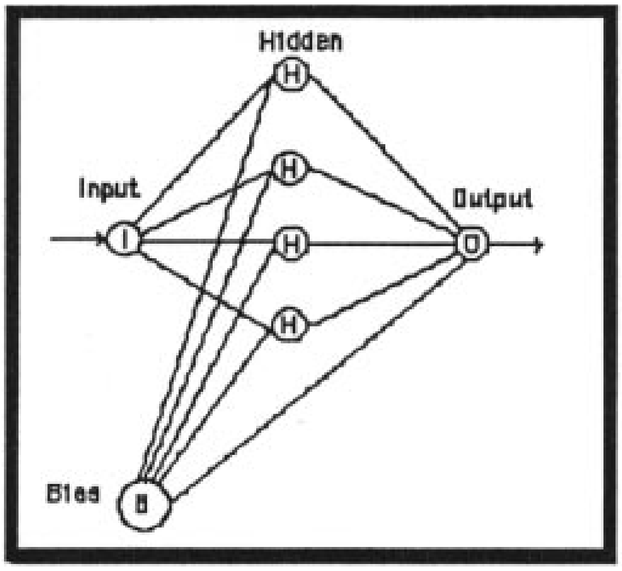

Figure 1. Neural network with one input unit, four hidden units and one output unit.

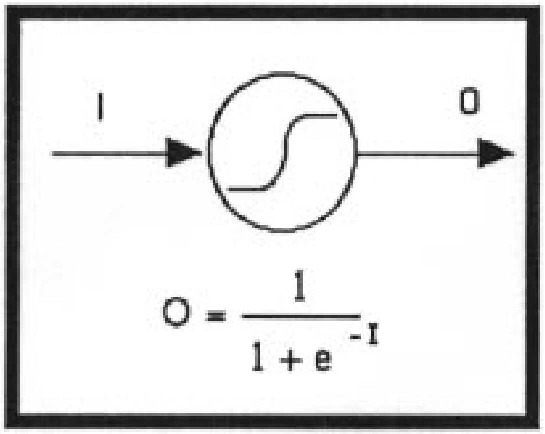

Figure 2. Operation of a typical hidden or output layer neuron. Input neurons have a response of O = I.

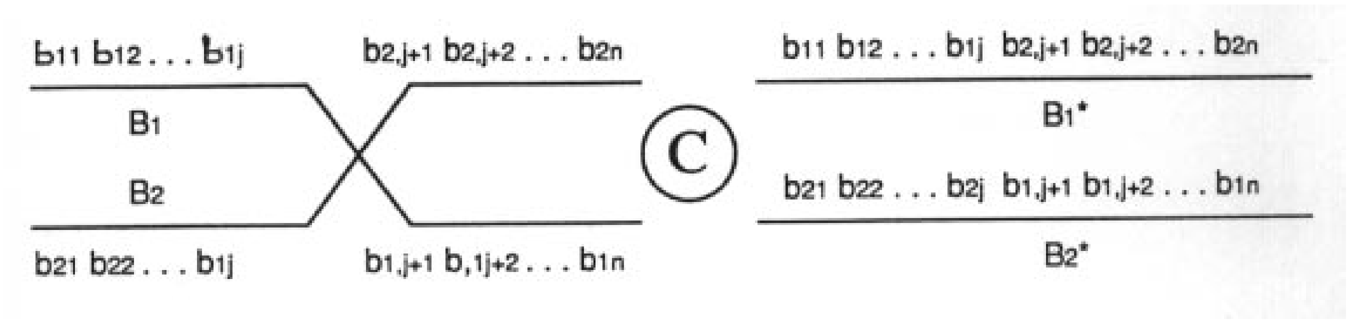

Figure 3: Two candidate solutions B_1 and B_2 undergoing a crossover operation at the randomly chosen bit location j to produce two new solutions B_1^* and B_1^*.

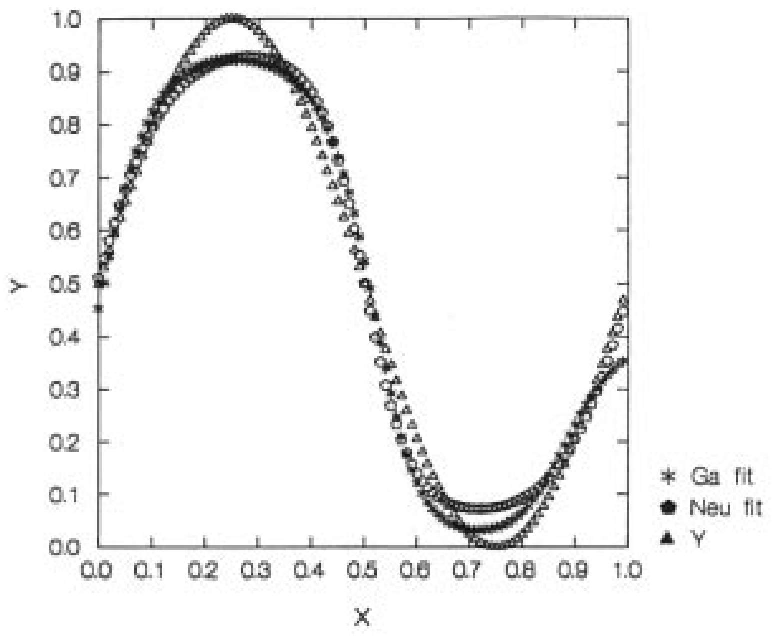

Figure 4. Graph of y = 0.5 + sin(2π x), 0 _ x _ 1 for 1000 data points, the estimates obtained from backpropagation (Neu fit) and the genetic algorithm (GA fit) are plotted. The fit is excellent for the entire function though both methods fail to reproduce the data at the peak and the trough. The GA fit is superior to the backpropagation fit at the turning points of the function.

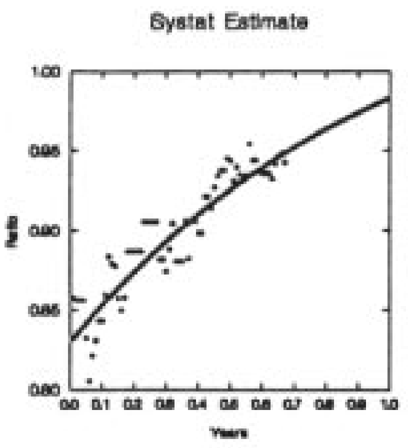

Figure 5

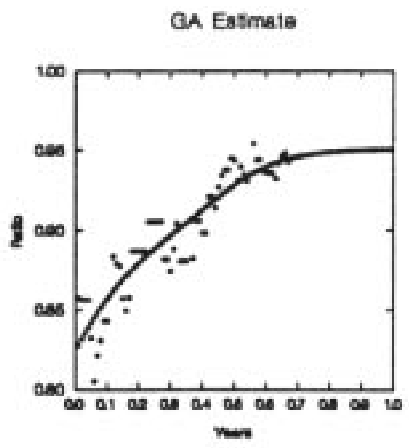

Figure 6

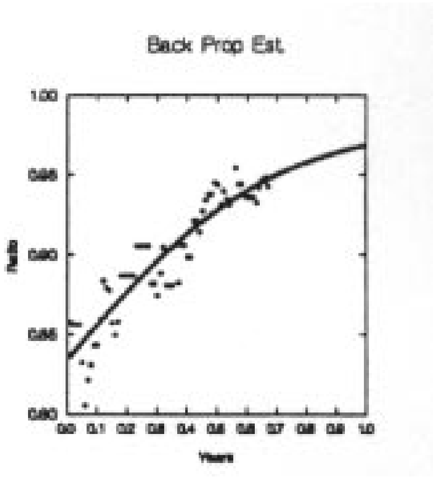

Figure 7

Figures 5, 6and 7. The least square fit of the parametric model $y_t = \alpha + \beta \exp(-\gamma t) + \varepsilon_t$ obtained from the program SYSTAT with the data superimposed are given in Figure 5. The fit from the genetic algorithm GA estimate) and the backward propagation algorithm (NNet Est) with the data are given in Figures 6 and 7.

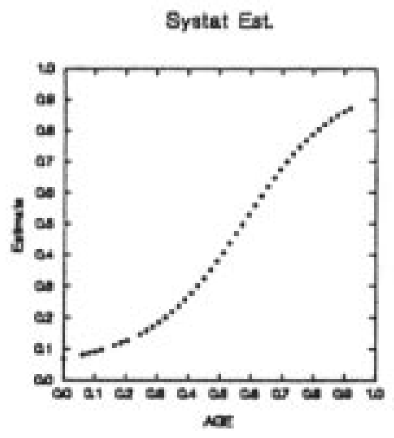

Figure 8

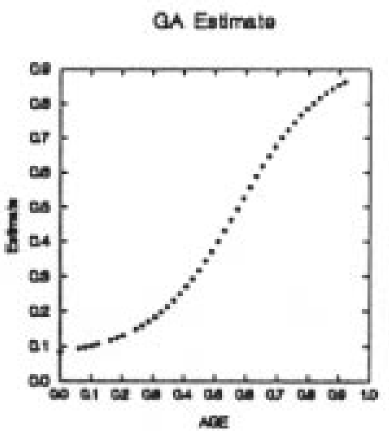

Figure 9

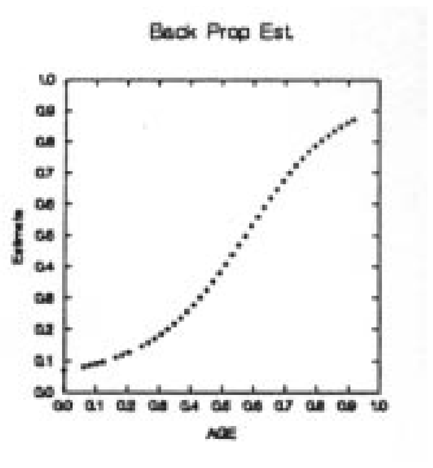

Figure 10

Figures 8, 9 and 10. The fitted estimates obtained from the parametric logistic model to the CHD data is given in Fgiure 8. The neural network estiamtes obtained for the same data using the GA and the back propagation method are given in Figures 9 and 10.