To the Graduate Council:

I am submitting herewith a thesis written by Timothy Daniel Choate entitled "Evolutionary Neural Network Design." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science.

Bruce Whitehead

Bruce Whitehead, Major Professor

We have read this thesis
and recommend its acceptance.

K R Kwolek

Accepted for the Council:

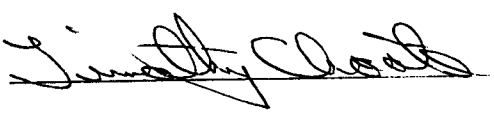
Carl Minkel

Associate Vice Chancellor and
Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from thesis are allowable without special permission provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date 7/17/91

Evolutionary Neural Network Design

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Timothy Daniel Choate

August 1991

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Bruce Whitehead for his invaluable guidance in the preparation of this thesis. I would also like to thank the other committee members, Dr. Kenneth Kimble and Dr. C. S. Lee. The comments of the committee members have resulted in a much stronger thesis for which I am grateful. I would like to thank the United States Air Force and the management of OAO Corporation at Arnold Engineering Development Center for creating an environment in which graduate studies are possible. I would like to thank my colleague, Sean Fuller, for many thought provoking conversations related to this research. Lastly, I would like to thank my wife, Tammy, for her encouragement and support over the years.

ABSTRACT

This thesis describes the application of genetic algorithms to the task of neural network design. The neural network model being configured is a radial basis function network. The network uses a single internal layer of locally-tuned processing elements. Each processing element in the internal layer uses a Gaussian response function to determine its sensitivity to ranges of the inputs. An adaptive weight matrix connects the internal layer to the output layer. Given a suitable number of processing elements in the internal layer, a suitable width for the Gaussian functions, and a suitable learning rate, this particular network model converges faster than traditional learning methods. The genetic algorithm searches for optimal values of the these three parameters by evolving network designs which are increasingly better at solving a particular mapping problem. Simulation results are presented for applications in real-valued function approximation and prediction of a chaotic time series. Results are compared to back propagation networks.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	A. Complex Adaptive Systems	1
	B. General Learning Techniques	2
	C. A Hybrid Learning System	3
II.	LITERATURE REVIEW	5
	A. Neural Networks	5
	A.1 Concepts	5
	A.2 Radial Basis Function Networks	8
	A.2.a Mechanics	9
	A.2.b Mathematical Description	9
	A.2.c Analysis	14
	A.3 Back Propagation Networks	16
	B. Genetic Algorithms	17
	B.1 Concepts	18
	B.2 Mathematical Foundations	20
	B.3 Advanced Research	26
	C. Genetic Neural Networks	28
	C.1 Concepts	28
	C.2 Advanced Research	29

III. EVOLUTIONARY NEURAL NETWORK DESIGN 31
A. System Overview 31
B. Genetic Encoding 33
C. User Defined Parameters to GA,
the Genetic Algorithm 34
D User Defined Parameters to EVALNET,
The Network Simulator 35
E. Summary 37
IV. SIMULATION RESULTS AND CONCLUSIONS 38
A. Problem Selection and Benchmark 38
B. Simulation Platform 39
C. Parameter Settings 39
D. Approximating a Gaussian Function 40
E. Predicting Future Values of a Chaotic
Time Series 45
F. Conclusions 49
LIST OF REFERENCES 52
VITA 56

LIST OF FIGURES

FIGURE		PAGE
1.	Basic Neural Network Processing Element	7
2.	RBFN Partitions in a One Dimensional Space	10
3.	Gaussian RBF Network with Internal Layer Consisting of Locally Tuned Processing Elements	11
4.	Architecture of the ENNDE System	32
5.	Gaussian Function Used to Test Function Approximation Abilities of ENNDE	41
6.	Comparison of ENNDE and BP as They Reduce Mapping Error Over Time for Normal Gaussian Function	43
7.	Average vs. Maximum Fitness of ENNDE as it Solves Gaussian Mapping Problem	44
8.	Chaotic Time Series Posed to ENNDE	46
9.	Comparison of ENNDE and BP as They Reduce Prediction Error Over Time for Chaotic Time Series	48
10.	Average vs. Maximum Fitness of ENNDE as it Solves Chaotic Time Series Prediction Problem	50

LIST OF ABBREVIATIONS

BP	-	Back propagation algorithm
BPN	-	Back propagation network
ENND	-	Evolutionary Neural Network Design
ENNDE	-	Evolutionary Neural Network Design Engineer
GA	-	Genetic algorithm
MGA	-	Messy genetic algorithm
ML	-	Machine learning
MSE	-	Mean Squared Error
NN	-	Neural network
PE	-	Processing element within a neural network
RBF	-	Radial Basis Function
RBFN	-	Radial Basis Function Network

LIST OF SYMBOLS

X	-	Number of input PE's in an RBFN.
Y	-	Number of internal PE's in an RBFN.
Z	-	Number of output PE's in a RBFN.
x	-	Vector of input activations for a RBFM.
y	-	Vector of internal layer activations for a RBFN.
z	-	Vector of output layer activations for a RBFN.
d_i	-	Desired output at output PE z_i.
e_i	-	Error calculated at a given output PE z_i within a RBFN.
μ_j	-	Vector containing the coordinates of the center point of the j-th RBF in the internal layer of a RBFN.
σ_j	-	Scalar value representing the width or variance of the j-th RBF in the internal layer of an RBFN.
W_{ij}	-	One element of the weight matrix connecting the internal and output layer of a RBFN.
alpha	-	The learning rate within a RBFN.
H	-	A specific schema as defined by Holland.
o(H)	-	Order of H, the number of fixed positions in the schema.
δ(H)	-	Defining length of H, distance between the first and last fixed string position within a schema.
m(H, t)	-	Expected number of occurrences of schema H at time t.
f(H)	-	Fitness of schema H.
avg(f)	-	Average fitness of the population.
max(f)	-	Maximum fitness of any individual in population.
K	-	Constant indicating the percentage by which a given schema exceeds the average fitness.
L	-	Length of a string.
p_s	-	Probability that a specific schema will be disrupted by crossover.
p_c	-	Probability that a crossover will occur during recombination of two strings.
p_m	-	Probability that a specific string position will mutate within a genetic algorithm.

CHAPTER I.

INTRODUCTION

A. Complex Adaptive Systems

The development of complex adaptive systems has proven to be extremely difficult using traditional programming languages. The problems stem from the fact that in many cases no algorithmic solution is known. In other cases, the problem does not fit the Von Neuman computing model, that is, the problem has elements which are parallel in nature. Often, such systems require characteristics such as fault tolerance and massive parallelism not normally associated with conventional programming tools and hardware.

It is these types of problems which have lead researchers to pursue new techniques for development of complex adaptive systems. Following the precedent of complex biological systems, the most common solution has been to develop software or hardware which can learn from experience rather than being programmed. This class of systems are collectively

known as machine learning systems (ML). The most promising of such systems are neural networks.

B. General Learning Techniques

At this point many (ML) systems have been proposed. These systems can be roughly classified by the manner in which they learn from their experiences. The most well-known type of learning is supervised learning. Under this model a teacher is assumed which provides the system with the desired output for each input example. The system then adjusts its internal processing to minimize the error between the desired and actual outputs.

Another model is unsupervised learning where the system develops its own internal classifications of the exemplars on which it is trained and no teacher is assumed. The effectiveness of these systems is the consistency of its internal classifications. Often unsupervised learning plays a part in larger supervised learning systems.

Reinforcement learning is a model which assumes an external teacher also. However, this model is different in that the teacher does not supply the correct answer, but rather provides only a scalar evaluation of generated outputs. Thus, this model must search for good answers and adjust its internal processing to increase the likelihood of good answers in the future.

The final learning model is evolutionary learning. This model is different in that it does not result in modifications within the lifetime of an individual, but rather relies on a large population of individuals and a "survival of the fittest" approach to learn the best solution.

C. A Hybrid Learning System

The goal of this research has been to develop a ML system which more closely models learning systems in nature. This has lead to the development of a hybrid system which uses supervised learning in the short-term and evolutionary learning in the long-term. This system is called the Evolutionary Neural Network Engineer (ENNDE). Within ENNDE, the supervised

learning is implemented as a particular type of neural network known as a Radial Basis Function Network (RBFN). The evolutionary component is implemented as a genetic algorithm (GA). Together these two modules comprise ENNDE, a system which is capable of performing tasks such as classification, prediction, and function approximation. The GA searches for optimal configurations for the RBFMs. Each potential configuration is evaluated by simulating the RBFN with a user defined training set. The result is that the ENNDE system searches for RBFN designs which demonstrate the best capacity to learn the user provided training set. The following chapter reviews previous work in the RBFN and GA fields. Each technique is described in detail and the points of interaction are identified.

CHAPTER II.

LITERATURE REVIEW

A. NEURAL NETWORKS

The term neural network (NN) is a general term used to reference all information processing systems modeled after nature's most sophisticated processing system, the animal brain. The first models of neural processing can be traced back to the 1940's and the work of pioneers such as McCullock, Pitts, and Hebb [14,15]. Since that time and especially recently, a tremendous amount of effort has be put into designing more efficient and robust neural models. Although the details of these models vary they all have a simple underlying architecture.

A.1 Concepts

The most basic component of all neural network models is the processing element (PE). The typical function of a PE is to integrate incoming signals over time. A threshold is associated with each PE. When

this threshold is exceeded, the PE fires, sending a signal to other PE's. When viewed as a function over time, this process of summation and thresholding is often referred to as a non-linear transfer function. In biological systems, it is believed that nature's PE (the neuron) works with frequency coded information, however, most artificial neural systems use real-valued frequency approximation to simplify simulations.

The mechanism for transmitting a signal from one PE to another is known by many names (link, connection, synapse, weight). The term connection will be used here. Associated with each connection is a variable weight. The network learns by adjusting these weights. Typically, increasing the weight will strengthen the association between the pre-connection PE and the post-connection PE. Figure 1 shows the basic processing element. An excellent introduction to the various neural models, learning techniques, and application areas can be found in Caudill and Butler [6].

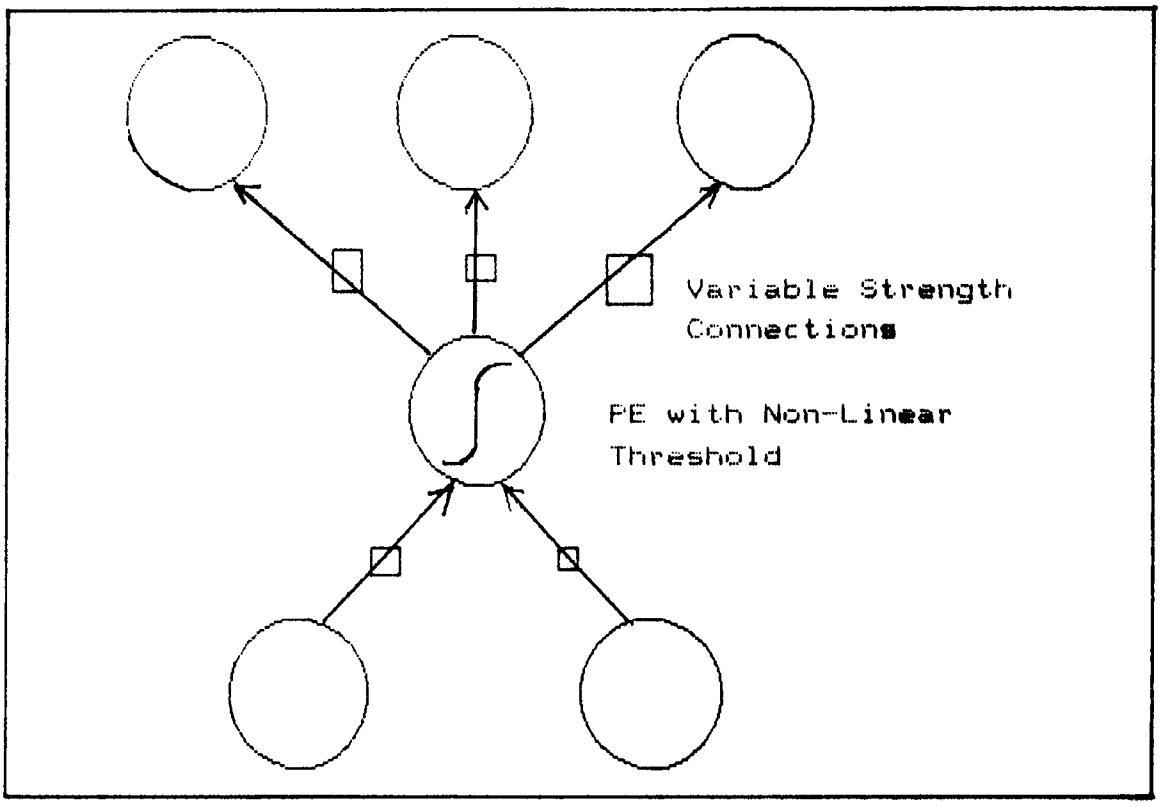


Figure 1. Basic Neural Network Processing Element.

A.2 Radial Basis Function Networks

In biological neural systems, neural cells have been identified which are locally-tuned or selective for some range of their inputs. The neural network analogy of these cells is the Radial Basis Function Network. Mathew describes the RBFN as follows:

> These networks can be viewed as three-layer networks with the weights between the input and hidden layers fixed by a suitable choice of distance metric and basis function. Each PE in the hidden layer corresponds to a radial basis function center. The output units are linear, so their values are linear combinations of the outputs from the hidden units. The coefficients of these linear combinations correspond to the weights between the hidden and output layers... Radial Basis Functions are thus orders of magnitude faster to train than MLP (Multi-Layer Perceptron) networks which use gradient descent, and are considerably faster than MLP's using more sophisticated optimization techniques such as conjugate gradient methods[16].

The RBFN is the basis of the supervised learning within the ENNDE system. The following section describes the particular RBFN used in ENNDE and identifies the configuration parameters which are optimized by the GA.

A.2.a Mechanics

As with most neural networks, RBFN's develop an internal model which partitions the input space. Ideally, a RBF center is positioned in the center of each partition. Figure 2 shows how an RBFN with one real-valued input x can be trained to approximate a real-valued function of x. The function of x is a large Gaussian Normal Curve. The locally-tuned Gaussian basis functions are positioned such that by adjusting the amplitude and width, an approximation can be developed. This technique can be applied to any number of input dimensions. With two inputs (x_1, x_2) the RBF centers are points in two-dimensional space and the RBF approximates a function whose values are represented in the third dimension.

A.2.b Mathematical Description

The RBFN used in this research consists of three layers, an input layer x, an internal layer y, and an output layer z (see Figure 3). Associated with each element of the internal layer is a σ_j and a μ_j

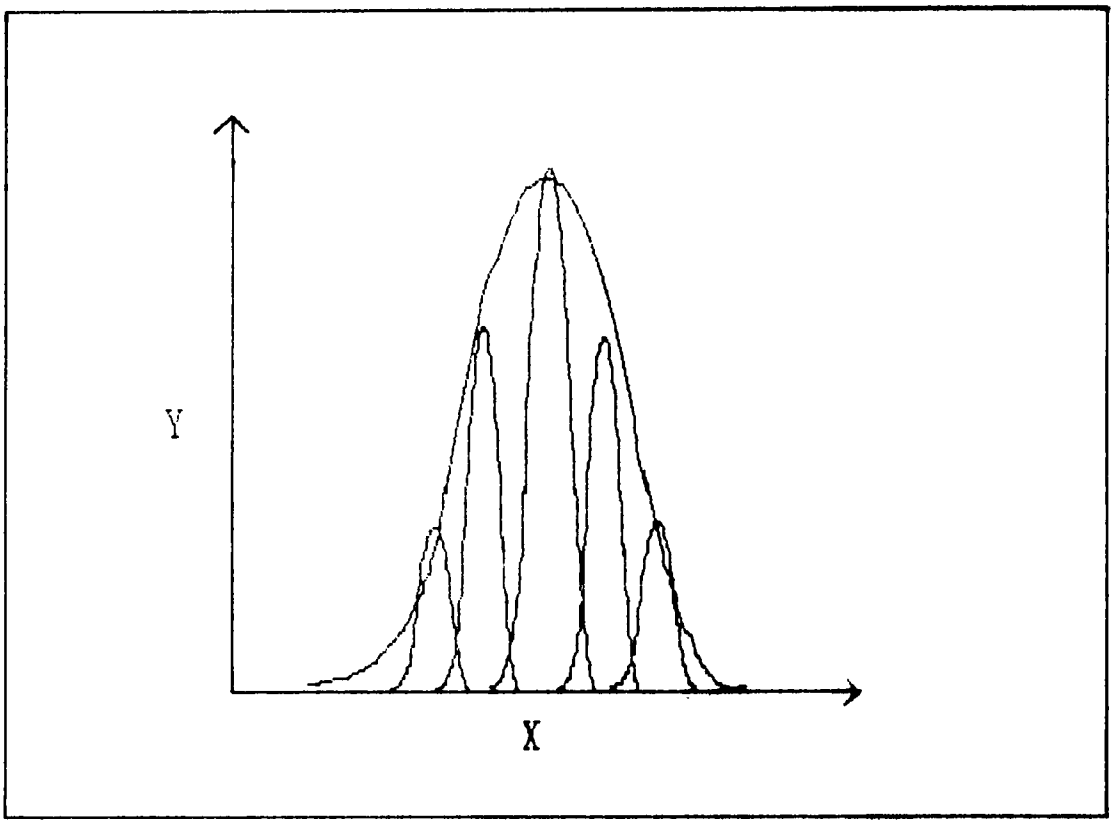


Figure 2. RBFN Partitions in a One Dimensional Space.

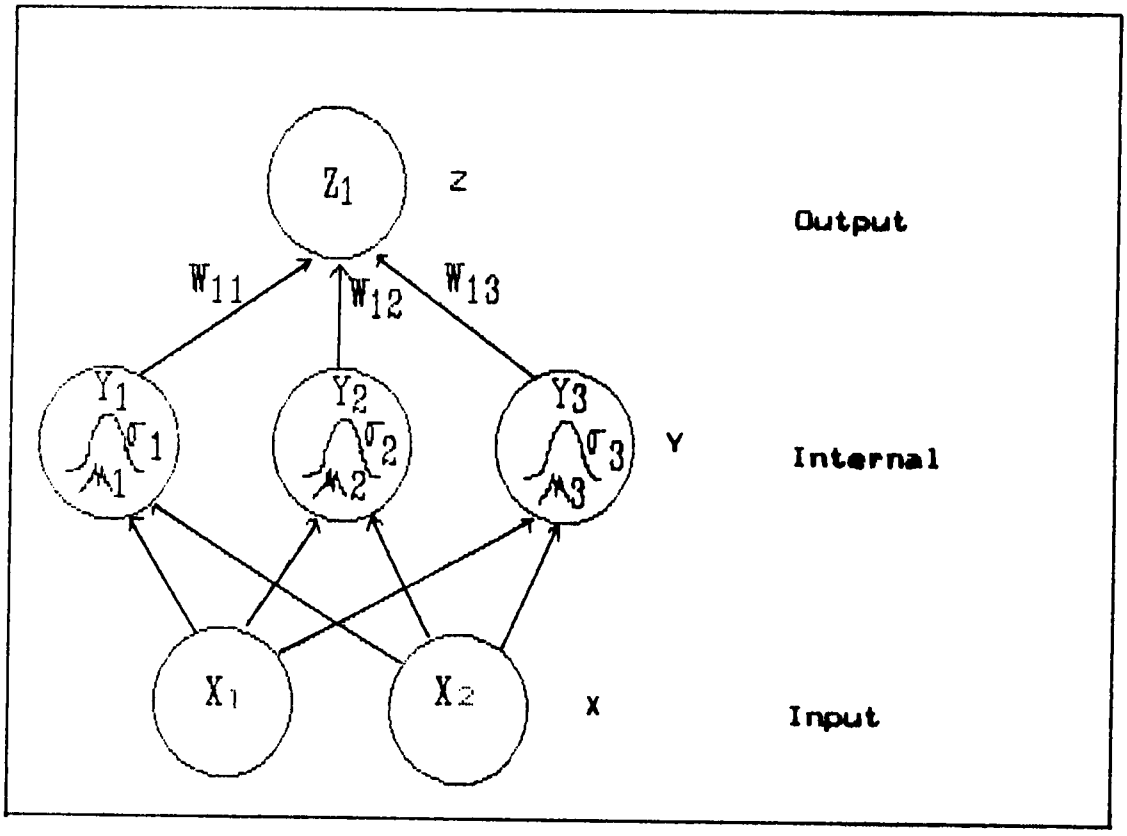


Figure 3. Gaussian RBF Network with Internal Layer Consisting of Locally Tuned Processing Elements.

where σ_j is a scalar value representing the variance or width of the RBF and μ_j is a vector containing the coordinates of the RBF center. Thus, μ_{jk} represents the k-th coordinate of the center point for the j-th RBF. It is important to note that each μ_j represents a point in n-dimensional space where n is the number of input PE's. Lastly, an adaptive weight matrix W fully connects the internal layer to the output layer. Assuming that the input vector x has been instantiated, the response function for a given element of the internal layer y_j is calculated as follows:

$$y_j = \exp\left[- \sum_{k=1}^{x} \frac{(x_k - \mu_{jk})^2}{\sigma_j^2} \right]$$

where X is the number of input PE's. This function is radially-symmetric with a single maximum at μ_j. The function drops off rapidly to zero at large radii. The result is a internal PE which responds to a selected portion of the input space as determined by μ_j and σ_j. This signal y is then multiplied through the weight matrix W to obtain the output vector z. A given element of the output vector z_i is calculated as follows:

$$z_i = \sum_{j=1}^{Y} W_{ij} * y_j$$

where Y is the number of internal PE's. Within ENNDE, the GA finds optimal values for Y, μ, σ, and alpha (described later). The remaining learning task involves modifying the weight matrix to adjust the amplitude of the response from the internal PE's. This learning task assumes iteration over a training set which contains groups of example inputs and desired outputs. At each data point within the training set, the inputs are clamped to the x vector and the desired outputs are stored in a vector designated as d. Learning is performed adaptively using a simple learning rule which calculates the error at each output node and adjusts the weights in a manner which minimizes the mean squared error [24]. The error e_i at a given output node z_i is calculated as:

$$e_i = z_i - d_i$$

The weights feeding into z_i can then be adjusted according to the following formula:

$$\Delta W_{ij} = e_i * \text{alpha} * y_j$$

where ΔW_{ij} is the change in the weight W_{ij} from one time step to the next, and alpha is the learning rate determined by the GA. After the algorithm has been exposed to the training set for a user-specified number of times, training is halted, and a test is performed. The test involves exposing the network to the data on which it was trained (or similar test data set) and calculating the total mapping error. This error is then used as a measure of how well the network learned.

A.2.c Analysis

The error minimization technique used for these RBF's is similar to the standard back propagation algorithm. The difference is that the computationally expensive sigmoidal function is not calculated at every node and due to the local response of the internal nodes, many PE's will have no significant activation level. Therefore, a simple optimization is to ignore these PE's when feeding forward the signals and calculating errors. The result is a network model which performs supervised learning orders of magnitude faster than techniques such as back propagation[4]. This high performance under ideal configurations is

the reason this neural network model was selected for genetic design.

It may be obvious from this description that the difficult problem in designing RBFN's is determining an optimal number of internal units, positioning the RBF centers μ_j, and setting the widths σ_j. Several techniques are known for performing this function in an adaptive fashion, the most well-known is the standard k-means clustering algorithm which can find a set of k processing element centers which represent a local minimum of the total squared euclidean distances between the elements of the training data and the nearest of the k centers [25]. The problem with this technique is that it can take many iterations on the training data to converge and results are not necessarily globally optimized [25]. Lastly, techniques such as the k-means algorithm yield no information as to the optimal number of internal units and the optimal learning rate.

The primary considerations when training a RBFN are the number of RBF's to use, the positioning of the RBF's, the widths of the RBF's, and the learning rate. It will be shown in the following sections that

parameter optimization tasks such as this can be performed by genetic algorithms, thus motivating the use of a GA in ENNDE.

A.3 Back Propagation Networks

The BPN is described here because it is the learning technique against which the ENNDE system was bench-marked. The BPN is an example of the class of networks which use a supervised learning rule that minimizes the error in the mapping from a set of training inputs to a set of training outputs. The BPN normally consists of an input layer, one or more hidden layers, and an output layer. These layers are fully connected and a feed-forward approach is used to calculate output activations. The network converges by adjusting the weights and thresholds in the network in manner that descends the gradient of the Mean Squared Error. It has been shown that feed-forward networks of this type can function as universal function approximators [26]. Even though no proof exists that the back propagation learning algorithm will converge for all mapping problems, its demonstrated performance across a broad range of applications has made it the workhorse of neural

networks and a consideration for most classification or mapping tasks. The primary criticism of this technique is that its learning rule is global and does not lend itself to the concepts of locality of processing and parallelism normally associated with neural networks.

The particular version of BPN used in this research contains the momentum optimization technique published by Rumelhart and associates [26]. Since the back propagation works by descending the gradient, this optimization involves storing the delta vector computed each cycle and using it to speed the descent down the gradient during the next cycle of the network.

B. Genetic Algorithms

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and natural genetics [6].

B.1 Concepts

These algorithms operate on a string which is modeled after the chromosome in human genetics. Specific positions in the string are comparable to genes in animal reproduction. The list of all valid values at each position in a string is the alphabet. The most common alphabet for genetic algorithms is {0,1}. At a given time, the algorithm works with a population of these strings. Each string encodes a potential solution or point in the search space. The basic genetic algorithm is shown below:

```
Initialize the population of individual strings
Loop until desired fitness achieved
    Evaluate each individual in the population
    Select Fit Individuals for Mating Pool
    Recombine to form next generation
Endloop
```

Each iteration of the above loop is termed a "generation" of the GA.

As stated, this algorithm is generic and can be applied to any problem domain. To customize the algorithm for a specific application, such as neural network design, one must determine the coding scheme used to map from an individual string to a

particular trial solution for the problem being solved. Once the coding scheme is established, an objective function must be formulated which returns a fitness value for each trial solution. This objective function is called by the step which evaluates all individuals in the population.

The selection of individuals is usually implemented as a roulette wheel simulation in which the fitness of the individual determines the amount of space it will occupy on the roulette wheel. A random number is then generated on the wheel and the associated individual is selected. The recombination step normally involves some exchange of information between two mates during the creation of a new string. Also, randomized operators such as mutation and inversion are often used to prevent stagnation of the algorithm.

The GA may be viewed as a search technique which seeks optimal fitness values within the space of all possible strings. While it utilizes random operators such as mutation, a GA is not a random walk. Through the exchange of information during cross-over, a GA

efficiently exploits historical information to speculate on new search locations with expected improved performance.

In terms of their application to the development of artificially intelligent systems, the most central characteristic of GA's is their robust ability to adapt to a constantly changing environment, an ongoing problem with current AI systems. Ideas such as self-repair, self-guidance, and reproduction are the rule for genetic algorithms, but, are almost unheard of in traditional artificial systems [6].

B.2 Mathematical Foundations

The seminal work in genetic algorithms was begun by John Holland and associates at the University of Michigan in the early 1960's [9-10]. Holland developed Schema Theory, a mathematical explanation of the high performance found in GA's.

To understand the significance of Holland's work some background information is required. A schema is a short pattern found within a string of a GA. For a GA with the alphabet {0,1}, a schema is a string

defined over the alphabet {0,1,*} where the "*" is a wild card symbol. For this alphabet, a valid schema might be *11*00.

Three metrics are defined for schemata. For a given schema H, the order of H, denoted o(H) is the number of non-* positions within the schema. The defining length of H is the distance between the first and last non-* string position, denoted as δ(H). For the following example, the o(H) is 6 and δ(H) is 9.

String H: 0*1**01*01*

Holland's work specifies three genetic operators which are active during recombination. The first is reproduction which involves the process of evaluating individuals and allowing each to reproduce with a probability proportional to its fitness. The second operator is cross-over. For the purposes of this discussion cross-over is assumed to be a single point crossover between two strings during reproduction as shown
below:

Parent 1 : 0110101110
Parent 2 : 1001101010

x Cross-Over Site = 4

Offspring : 0111101010

Genetic information is copied from Parent 1 until the cross-over site is reached. The remaining information is then copied from Parent 2.

The last operator is mutation. Mutation refers to a random modification of a position in the genetic string to another character in the alphabet. With these definitions Holland developed the mathematics to analyze the effects of these operators when embedded in a genetic algorithm.

Each generation of the GA may be viewed as a single step in a discrete time series. With this view, the effect of reproduction on the expected number of a particular schema within a population can be stated as:

$$m(H, t+1) = m(H,t) * \frac{f(H)}{avg(f)}$$

where m(H,t) is the expected number of occurrences of schema H at generation t, f(H) is the fitness of schema H, and avg(f) is the average fitness of the

population. Thus, the effect of reproduction is quite clear, the above-average schemata grow and the below-average schemata die. If we assume that a particular schema H stays above average an amount K * avg(f), where K is a small positive number, we can rewrite this formula as follows:

$$m(H, t+1) = m(H,t) * \frac{(avg(f) + K * avg(f))}{avg(f)}$$

$$= m(H,t) * (1 + K)$$

Starting at generation t=0 and assuming a constant value for K, we obtain:

$$m(H,t) = m(H,0) * (1 + K)^t$$

Now we see that the net effect of reproduction is an exponentially increasing number of trials granted to above average schemata.

Now we consider the effect of crossover. Clearly crossover can disrupt a schema. When a crossover occurs for a string of length L, the lower bound on the crossover survival probability p_s for a particular schema H is:

$$p_s = 1 - \frac{\delta(H)}{(L - 1)}$$

If crossover is performed with a specified probability p_c, then the survival probability for a particular mating is given by:

$$p_s >= 1 - p_c * \frac{\delta(H)}{L - 1}$$

Now the combined effect of reproduction and crossover may be stated as:

$$m(H,t+1) >= m(H,t) * \frac{f(H)}{avg(f)} * \left[1 - p_c * \frac{\delta(H)}{L - 1} \right]$$

This indicates that above-average schemata with relatively short defining lengths, will be sampled with exponentially increasing rates under the operators of reproduction and crossover.

Finally we consider the effect of the mutation operator. As stated before, mutation is the random alteration of a single point in the string. A schema survives mutation when all of its component positions survive mutation. The probability of mutation for a single position is p_M. For a schema H of order o(H), probability of surviving a mutation is shown as:

$$(1 - p_M)^{o(H)}$$

and for small values of p_M this survival probability can be approximated by:

$$1 - o(H) * p_M$$

This leads to the central equation of Holland's Schema Theorem which shows the expected number of occurrences of a particular schema H in the next generation of a genetic algorithm after reproduction, crossover, and mutation [9-11].

$$m(H,t+1) >= m(H,t) * \frac{f(H)}{avg(f)} * \left[1 - p_C * \frac{\delta(H)}{L - 1} - o(H) * p_M \right]$$

As shown by this equation, the addition of mutation has little effect on the earlier conclusions. A GA performs an effective parallel search of the string space by exponentially increasing the number of trials allocated to above average schemata. The crossover and mutation operators provide an effective means of searching over high performance areas of the search space, but the equation shows that for small values of

p_m and schemata with relatively short defining lengths $\delta(H)$, the exponential term dominates the value of the function.

B.3 Advanced Research

Encouraged by the argument presented above, researchers have successfully applied genetic algorithms to applications including game playing, pattern recognition, function optimization, genetic programming, industrial control, and even the Traveling Salesman Problem [6]. For many applications, the basic GA is sufficient; however, extensions to the algorithm have been developed to improve performance for certain classes of problems. Extensions have included ideas such as adding inversion operators to the recombination step. The inversion operator simply reverses the order of some subset of the string. The idea of crowding has been introduced. This modifies the objective function to lower the fitness of individuals which are over-populating an area of the search space. This works well for problems where there are many equally fit solutions spread throughout the search space. Goldberg has conducted in-depth studies of functions which are GA-difficult, referred

C. Genetic Neural Networks

With researchers studying machine learning in neural networks and genetic algorithms, it is not surprising that the idea of combining the two has emerged.

C.1 Concepts

The basic algorithm is shown below.

```
Initialize the population of NN designs randomly
Loop until acceptable accuracy is achieved
    For each individual design
        Implement Network from Genetic String
        Train the Network for a Specific Problem
        Test Network Accuracy at Solving Problem
        Calculate Fitness based on accuracy
    Endfor
    Select Most Fit Designs for Mating Pool
    Recombine to form next generation
Endloop
```

C.2 Advanced Research

An excellent introduction to the concepts of using genetic algorithms to design neural networks is Caudill [18]. In this article she addresses the basics of encoding network design information into string format for a GA as well as issues related to training. W.B. Dress explains the application areas in dire need of the type of function performed by genetically designed neural systems and describes a system used to evolve a neural-based control mechanism for an autonomous vehicle [1].

Some researchers have attempted to encode all design information about the network into the genetic string. This includes the connection weights, PE thresholds, and even learning parameters[2,22]. In this case no training stage is required. This works well for small problems but the approach does not scale well to larger problems [23].

Recently, more success has been obtained by encoding higher level information in the GA, such as the number of hidden layers, number of hidden PE's, and connectivity information for back propagation

networks. Using this technique a training sequence is executed for each trial network, followed by a test to determine how well learning has progressed.

It seems that compactness of the design information with the genetic code is a critical consideration. In an attempt to minimize the space required by the coding scheme, Kitano has successfully implemented a technique which stores a graph generation grammar in the genetic string [23]. In this concise representation, complex networks can be specified in a very tight string. This seems to support the findings of the Schema Theory presented earlier which predicted that short, highly fit schemata would thrive while schemata with longer defining length would be more susceptible to disruption by genetic operators.

To the knowledge of this researcher, this thesis represents the first attempt to apply a GA to the design of RBFN's. Thus, the architecture and encoding scheme presented in the following chapter is unique.

CHAPTER III.

EVOLUTIONARY NEURAL NETWORK DESIGN

A. System Overview

This chapter describes the ENNDE system in detail. Figure 4 shows the configuration of the ENNDE system. The genetic algorithm module is called GA. The objective function evaluates each new network design is called EVALNET. To use the system, a user must create a configuration file which contains control parameters for GA and EVALNET. Also, a file containing training data must be supplied by the user. The system creates several trace files with statistics such as average fitness, maximum fitness, and minimum error found for each generation of the genetic algorithm. ENNDE is written in standard C and thus is portable to many platforms.

During an execution of the ENNDE system, the GA works only with a population of binary strings explained in Chapter II. The GA knows nothing about the information encoded into the strings. When the GA needs to evaluate a particular string, it simply

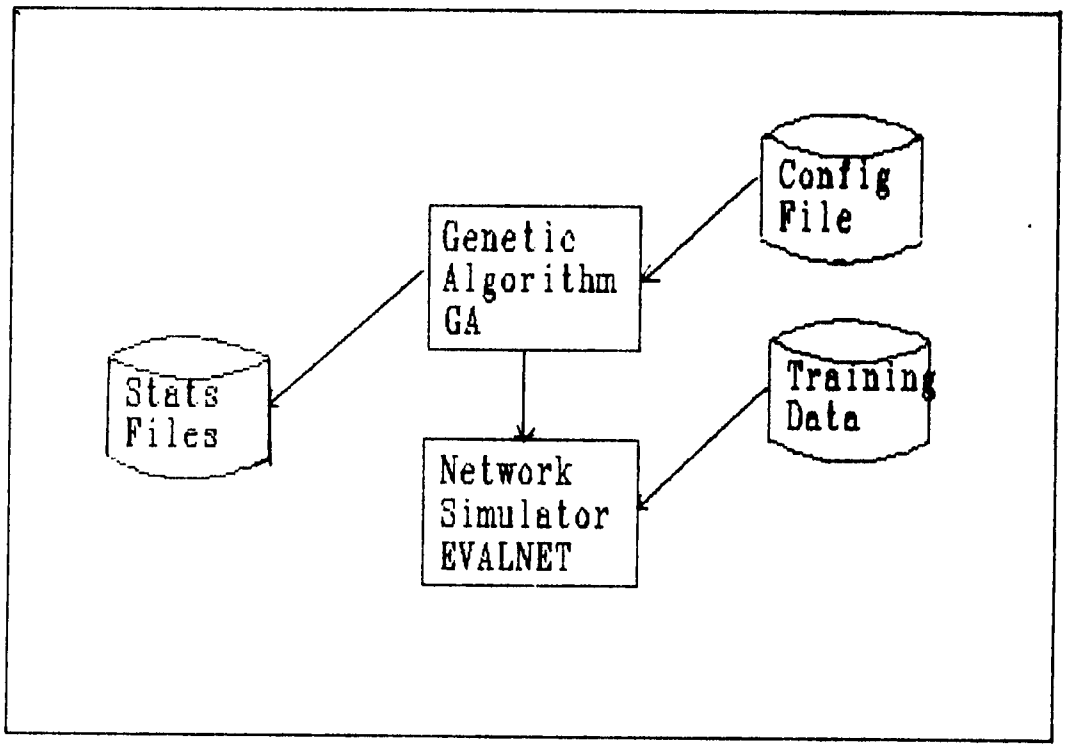


Figure 4. Architecture of the ENNDE System.

passes the binary string to EVALNET. EVALNET decodes the genetic string into a RBFN specification as described in Chapter II, attempts to train the network against the user provided data, tests how well the network learned, and returns a fitness value back to the GA. Using the genetic operators of selection, crossover, and mutation, the GA proceeds to search the space of all possible RBFNs for those demonstrating the most potential to learn the supplied training data.

B. Genetic Encoding

The first task in mapping any design problem to a GA is to determine how a neural network design specification can be encoded into a binary genetic string. In Chapter II the basic parameters to the RBFN model were defined as: the number of Gaussian RBF´s used to partition the input space, the width of these RBF´s, the centers of these RBF´s, and the learning rate. Thus, the number of RBF´s was encoded as an 8-bit binary number. By assuming that the centers are distributed evenly across the data space, the centers can be calculated from the number of RBF´s and does not need explicit representation. For the

ENNDE system, a uniform width is assumed for all RBF's. A 15-bit binary fraction is encoded in the genetic string. This fraction is decoded to mean a fractional part of the input data space which will be the width of all RBF's. Lastly, a 4-bit binary fraction is used to encode the learning rate within a RBFN. All of this information was encoded into a 27-bit string with the following structure:

Positions	Contents
1-8	Y, Number of Internal RBF PE's. Range from 0 to 255
9-24	Fractional part of input domain used as width for each Gaussian. This value determines the values of the RBF centers and widths as described earlier. Range 0 to $(1 - 1/2^{15})$
24-27	alpha, the learning rate used for supervised learning within the RBFN. Range 0 to $(1 - 1/2^{4})$

C. User Defined Parameters to the Genetic Algorithm

The genetic algorithm used in ENNDE is fairly standard. It supports only reproduction, crossover, and mutation as its genetic operators. ENNDE supports user defined control of the critical parameters of the GA via a configuration file. A list of these parameters follows:

Parameter	Usage
POPSIZE	Determines number of designs under consideration in 1 generation of GA
MATEPOOLSIZE	Number of designs selected for mating during each generation.
GENERATIONS	The number of generations to run GA
PMUT	The probability of mutation
PCROSS	The probability of cross-over

By modifying the configuration file, a user may easily experiment with different parameter values. It should be noted that optimal values for some of the above parameters can be automatically calculated, but since ENNDE is an experimental system, these parameters are left to the user's discretion.

D. User Defined Parameters to EVALNET, the Network Simulator

EVALNET serves as the objective function for the genetic algorithm. Its function is to take a genetic string, decode the string into a network design specification, implement the network, train the network on the user provided exemplars, test how well the network learned to solve the problem, and calculate a fitness for the network. Within ENNDE,

the fitness is defined to be 1/(total error), where total error is the sum of the error detected when testing the network. Within ENNDE, the same data set is used for both the training and testing processes.

In order to provide some user control of the EVALNET module, the following parameters may be entered in the configuration file.

Parameter	Usage
DATAFILE	Name of file containing training data
DATALBOUND	Lower bound of input training data
DATAHBOUND	Upper bound of input training data
NUM_IN	Number of input PE's
NUM_OUT	Number of output PE's
NUMTRNCYLES	Number of training iteration performed before network is evaluated
MAXTRAINSECS	Number indicating the maximum number seconds which may be expended training a network. Upon expiration evaluation is forced and fitness is calculated.

By modifying these values a user can modify ENNDE for different problem domains and constrain the situations under which the network must learn. For example, if the user decreased the number of training cycles allowed, ENNDE might be forced to configure a network with a larger alpha value and many internal nodes. This type of flexibility is seldom seen in tradition

AI approaches and is made possible because of the hybrid model used in ENNDE.

E. Summary

This chapter has described ENNDE, a system which supports a hybrid approach to the design of neural networks. The system architecture has been presented. The interaction between the neural network model and the genetic algorithm has been explored, and the user interface has been described. The next chapter presents the results of simulations in which the capabilities of ENNDE were tested.

CHAPTER IV.

SIMULATION RESULTS AND CONCLUSIONS

A. Problem Selection and Benchmark

To test the capabilities of ENNDE system, a function approximation problem and a prediction problem were selected. Benchmarks were performed against the two problems and results are compared to the results of a back propagation network with the same data. The momentum version of back propagation was used. The first problem was to approximate a normal Gaussian curve. This problem was selected first because it is a simple function and analyzing the behavior of the algorithm was manageable by hand. This second problem selected was prediction of future values of a chaotic time series. This problem was selected because it is known to be difficult and has become a standard for testing prediction techniques. Also, it has been shown that classical techniques perform no better than chance for solving the problem as stated [4].

B. Simulation Platform

For larger problem spaces both ENNDE and the BP algorithm can be computationally intensive. For this reason, all simulations were performed on a Silicon Graphics IRIS 4D/240 GTX. This machine demonstrates performance in the 10 MFLOP range. For all plots, time is shown in elapsed seconds. All simulation was performed in single user mode to discount machine loading. The simulation executed in single CPU mode due to the lack of a vectorizing C compiler.

C. Parameter Settings

For all simulations the following parameters were used for ENNDE:

Parameter	Value
PMUT	.05
PCROSS	.5
POPSIZE	20
MATEPOOLSIZE	10
GENERATIONS	10
NUMTRNCYCLES	10
MAXTRAINSECS	20

For all simulations the following parameters were used for back propagation:

Parameter	Value
Hidden Layers	1
Hidden PE's	20
LearnRate	.1
Momentum	.9
Sigmoid Term	2

These values of these parameters were selected to be the same as previous works [4,26] for the purpose of comparability.

D. Approximating a Gaussian Function

The first problem posed to ENNDE was to approximate the Gaussian function shown in figure 5. For this problem, a network with 1 input PE and 1 output PE was sufficient. A training file consisting of 66 data points sampled from the Gaussian curve was created. A data point within the training file consisted of the x value and the desired y value. The task for ENNDE was then to determine the optimal network configuration for learning this function approximation in 10 training cycles. Since the Gaussian is the basis of its RBF functions, it was assumed that the optimal value of 1 processing element in the internal layer might be found. Instead the system converged on a network with 115 internal PE's,

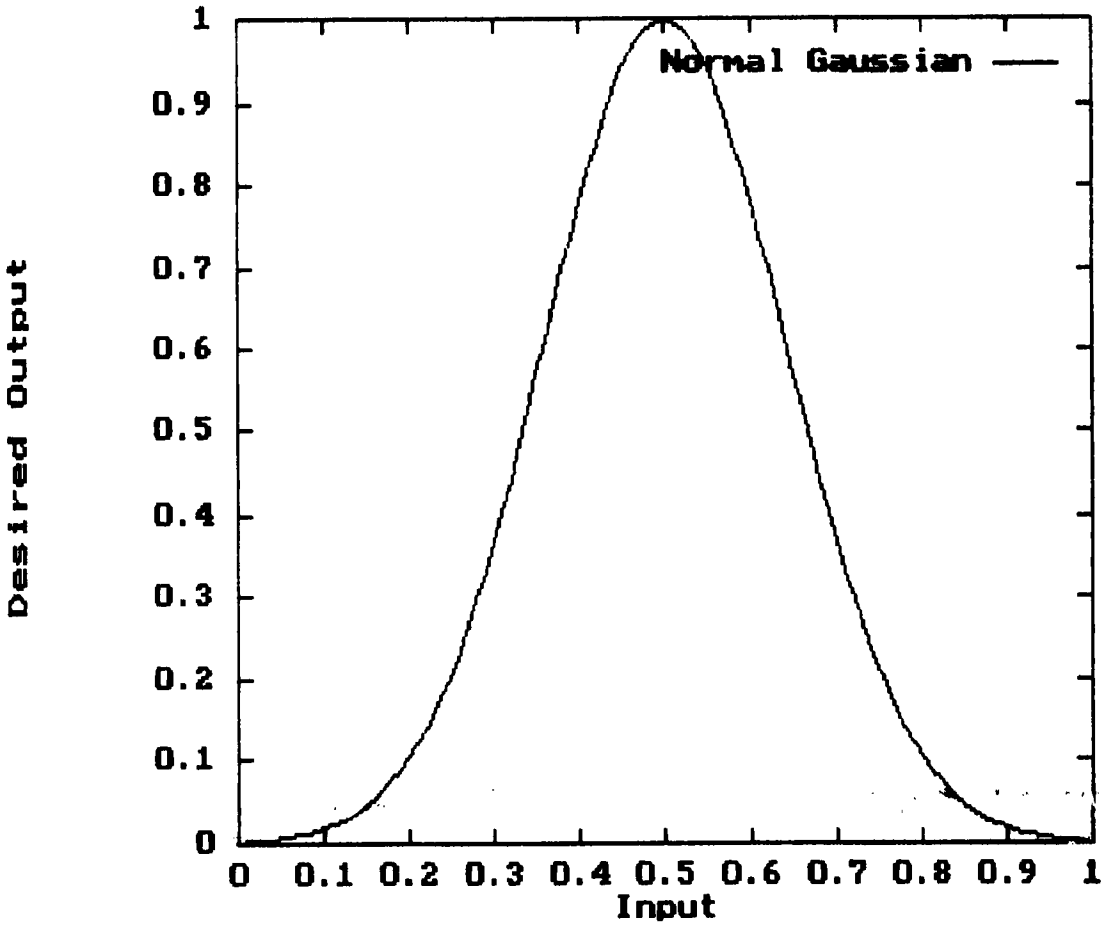


Figure 5. Gaussian Function Used to Test Function Approximation Abilities of ENNDE.

a RBF width of .0426, and a learning rate of .0625. Under this configuration, a mapping was implemented with very small error in eight generations of the genetic algorithm. Analysis of this behavior yielded the observation that the objective function of the genetic algorithm does not penalize a design for the number of internal PE's used. For this reason, the algorithm discovered a design which minimized the error without regard for the number of internal PE's.

Figure 6 compares ENNDE's results to back propagation results. In this plot the apparent lag before a point is plotted for ENNDE is due to the fact that error is only plotted at the end of each generation. The lag represents the length of time required to process the first generation. The fact that BP converges faster for this example is explained by the small area within the RBFN configuration space which yields an accurate result for this problem. The best results generated used a very narrow width for the RBF functions. Notice that in order for the GA to find this configuration it was required to evaluate many RBFN's with wider widths on the RBF's. Notice also that each RBFN was constrained to the lesser of 10 training cycles or 20 seconds. Figure 7 shows the

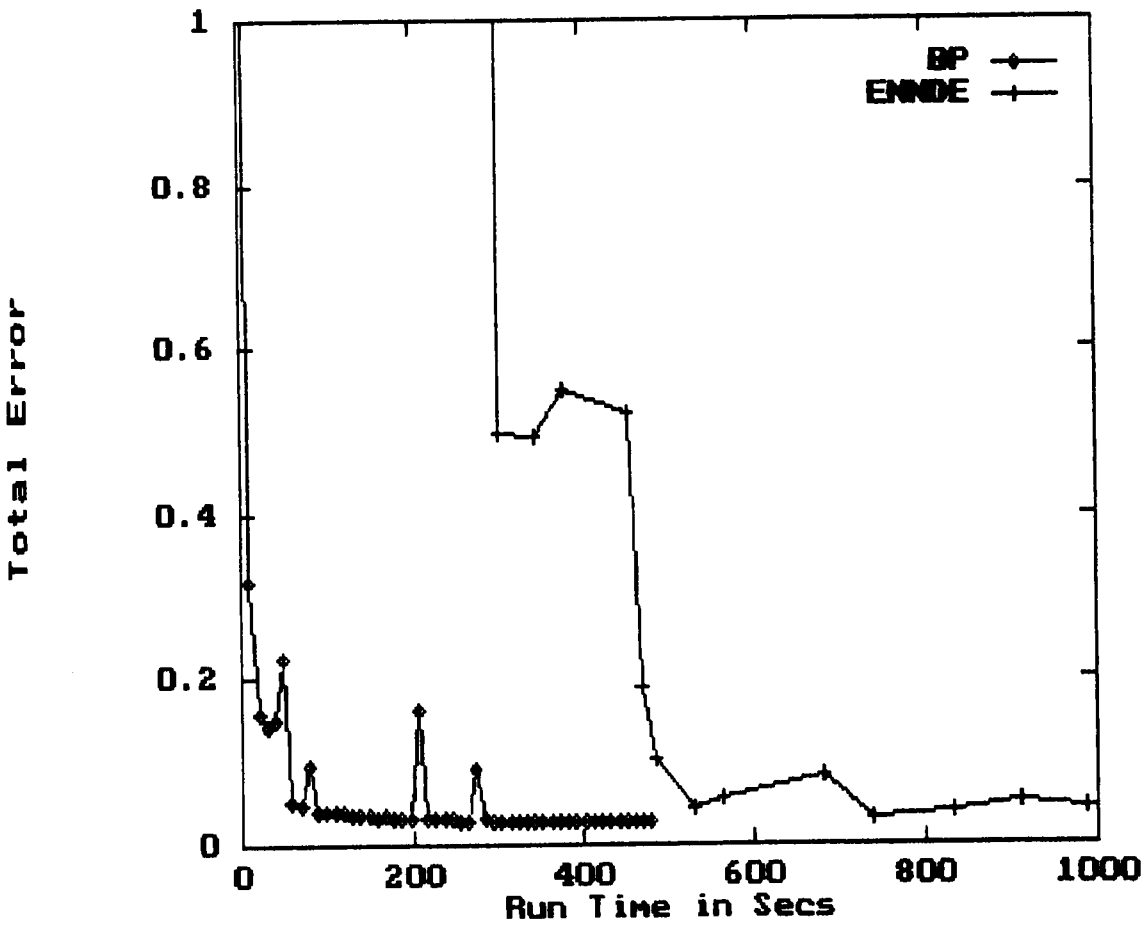


Figure 6. Comparison of ENNDE and BP as They Reduce Mapping Error Over Time for Normal Gaussian Function.

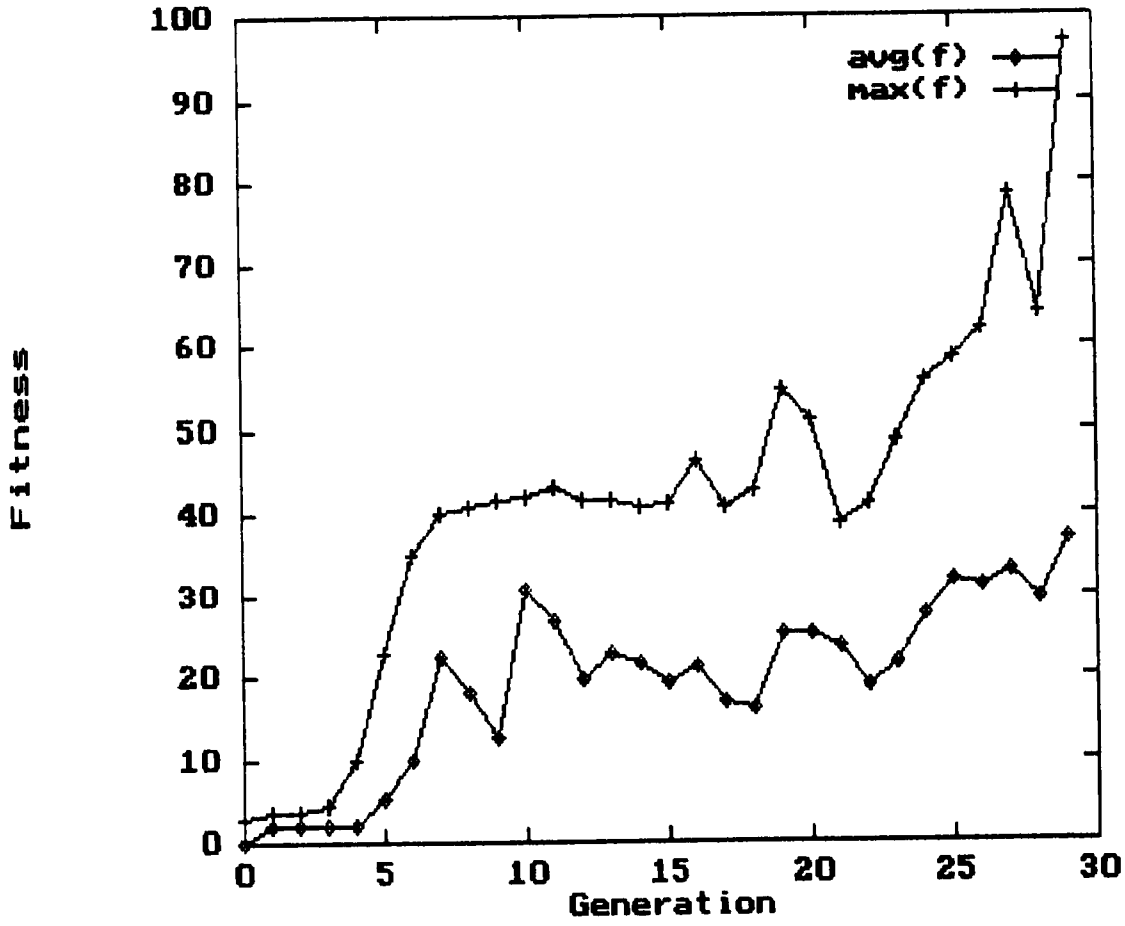


Figure 7. Average vs. Maximum Fitness of ENNDE as it Solves Gaussian Mapping Problem

average and maximum fitness of the populations of network designs as ENNDE converged on a solution. This plot is typical of a GA gradually optimizing over a complex search space.

E. Predicting Future Values of a Chaotic Time Series

The second problem selected for ENNDE was the difficult task of predicting future values of a chaotic time series, as described by Moody and Darken [4]. This chaotic time series results from integrating the Mackey-Glass differential equation (expressed as a difference equation):

$$\frac{dx[t]}{dt} = -bx[t] + a * \frac{x[t - T]}{1 + x[t - T]^{10}}$$

where a = 0.2, b = 0.1, and T = 17. The formula above and parameters are taken directly from Moody and Darken [4]. Figure 8 shows the resulting time series. The characteristic time (ie. wavelength) of the series is approximately 50. Classical techniques such as global linear autoregression and Gabor-Volterra-Wiener polynomial expansions are known

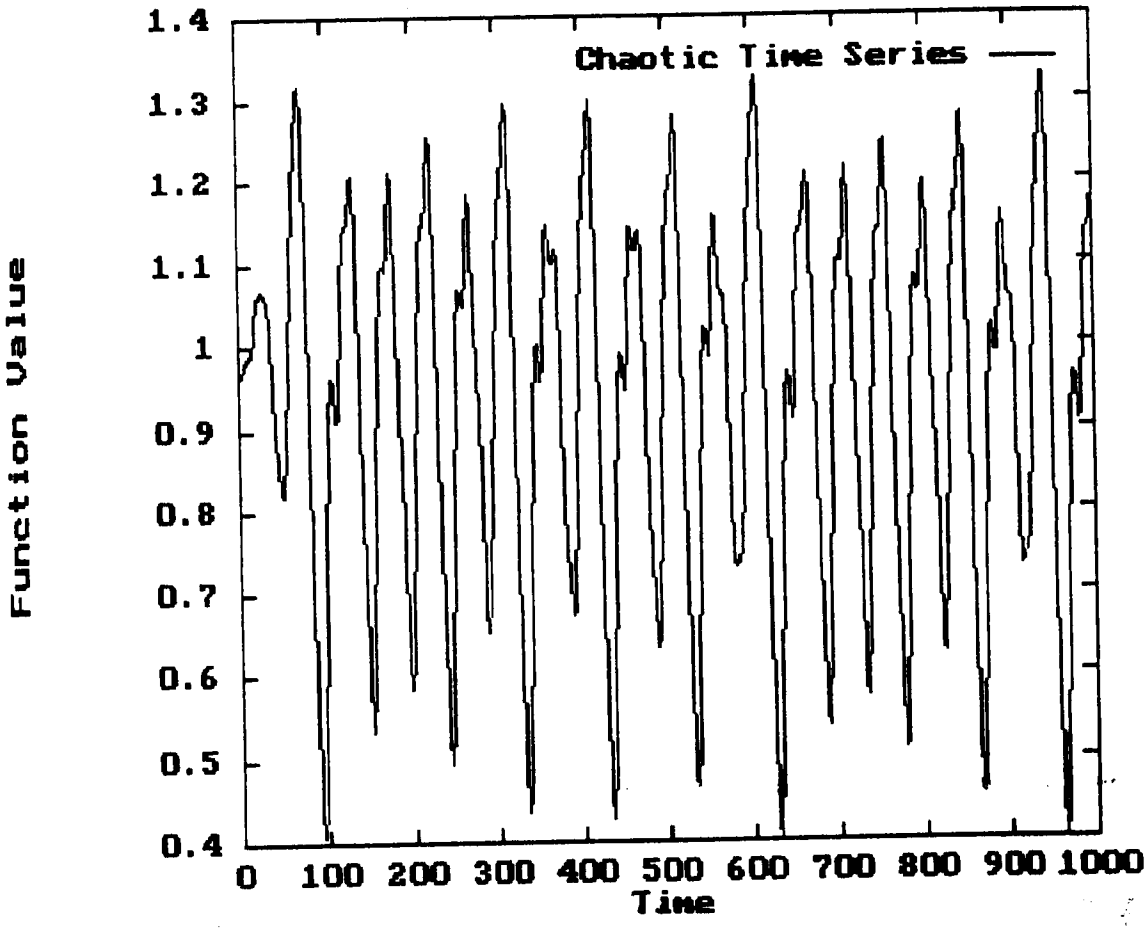


Figure 8. Chaotic Time Series Posed to ENNDE

to perform no better than chance at predicting this type of time series beyond its characteristic time [4].

Following Moody and Darken, a training set was created containing 200 data points. Each data point consisted of 4 real-valued inputs sampled from the time series followed by a single desired output which represents the value of the function 85 time-steps later. Mathematically, the data points can be described as 4 real-valued inputs defined as (x[t], x[t - D], x[t - 2D], x[t - 3D]) and one real-valued output x[t + I] where D = 6 and I = 85. Note that I is greater than the characteristic time of the series. Generally stated, this is the problem of predicting future values of a unknown function based upon discrete samples collected prior to the prediction. Figure 9 shows a comparison of BP and ENNDE as they reduce the error in their ability to solve this problem. Once again, the delay before the first ENNDE value represents the time required to evaluate the first generation of the genetic algorithm. It is interesting to note that the successive generations exhibited better performance as the wider, more computationally expensive RBF's are replaced with

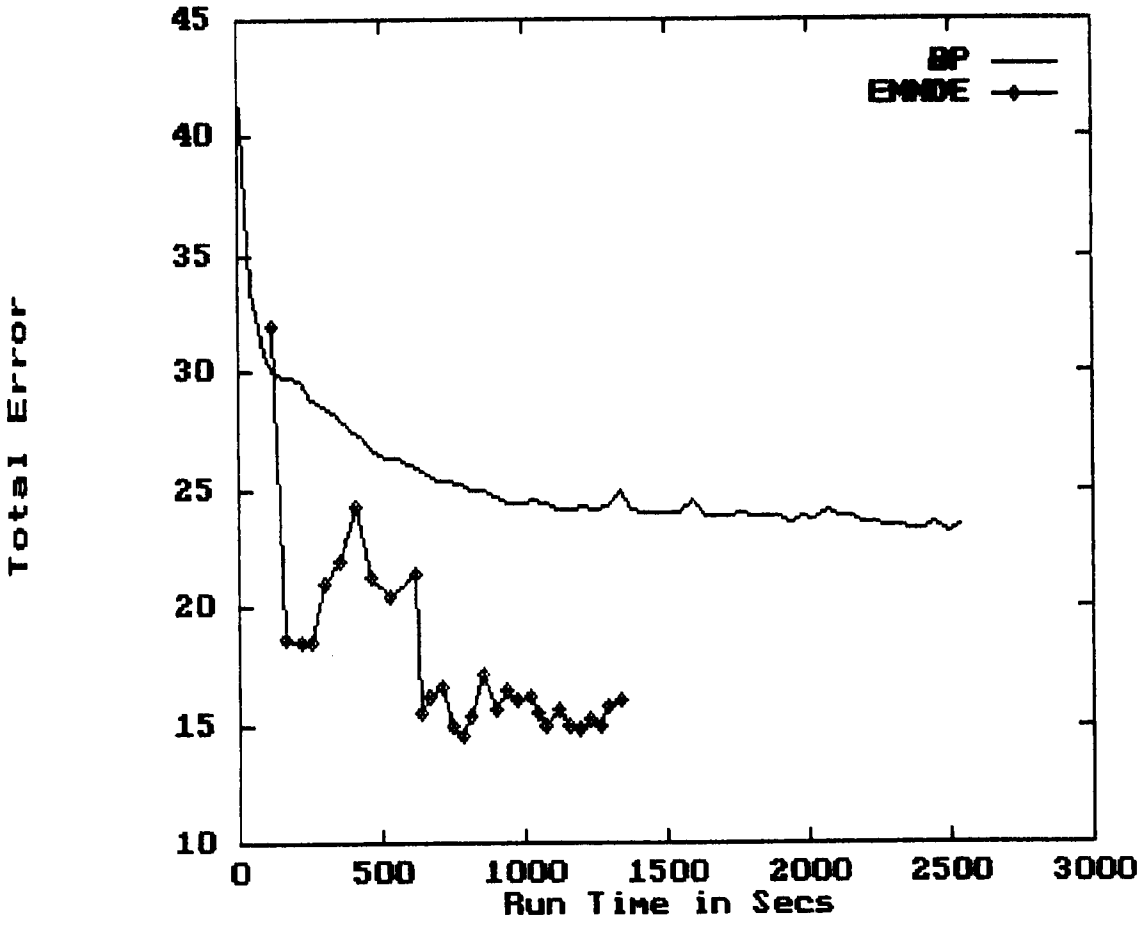


Figure 9. Comparison of ENNDE and BP as They Reduce Prediction Error Over Time for Chaotic Time Series.

localized PE's. As shown by Figure 9, ENNDE converges faster than back propagation even though it must deal with errors resulting from searching the network configuration space. Figure 10 shows the average and maximum fitness of the population for the ENNDE system over 30 generations. Analysis of this plot shows that ENNDE found good maximum fitness values early in the search process. The rest of the time was spent raising the average fitness to this level. Usually, this type of behavior indicates that good solutions are plentiful in the configuration space. In this case, it may mean that Gaussian RBFN's are simply a good technique for predicting this chaotic time-series. ENNDE eventually converged on a configuration of 38 Gaussian elements, a width representing .002 of the function range, and a learning rate of .375.

F. Conclusions

This thesis has described a hybrid learning technique which combines short-term supervised learning and long-term evolutionary learning. This technique has been implemented in a system called the Evolutionary Neural Network Design Engineer (ENNDE).

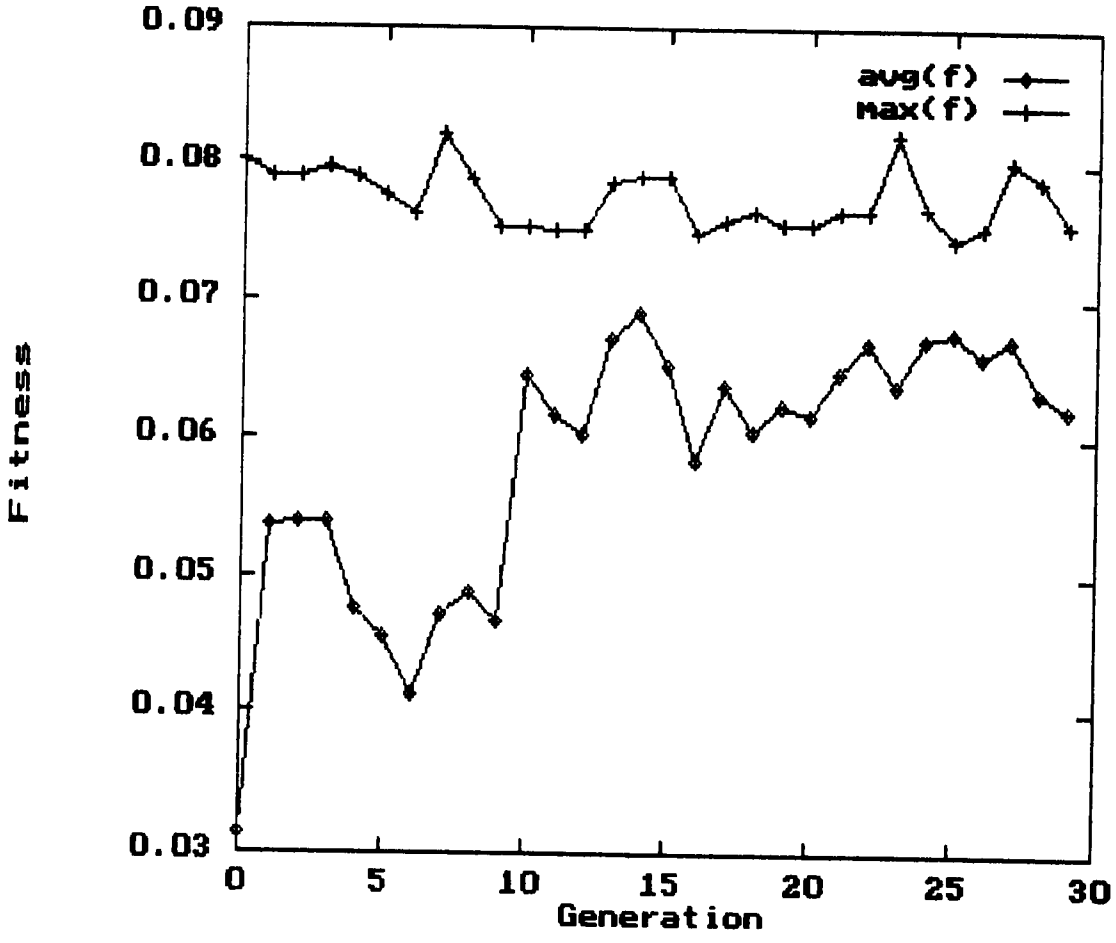


Figure 10. Average vs. Maximum Fitness of ENNDE as it Solves Chaotic Time Series Prediction Problem

ENNDE uses a genetic algorithm to tailor neural network designs for solving specific mapping problems. The neural network model being designed is a radial basis function network (RBFN). The genetic algorithm finds values of the RBFN design parameters which yield the networks most capable of learning some specific problem. Simulation results show that this technique is applicable to real-valued function approximation as well as prediction tasks. Simulation results also show that this technique demonstrates performance which compares well to back propagation. Further, for selected problems, ENNDE has demonstrated performance which is superior to back propagation.

LIST OF REFERENCES

LIST OF REFERENCES

[1] W.B. Dress, "Electronic Life and Synthetic Intelligent Systems," Computers and Mathematics with Applications, v. 20, no. 4-6, pp. 303-312, 1990.

[2] Hugo de Garis, "Building Artificial Nervous Systems Using Genetically Programmed Neural Netowrk Modules," Machine Learning: Proceedings of the Seventh International Conference, ed. Bruce Porter and Raymond Mooney, University of Texas, pp. 132-138, 1990.

[3] J.D. Schaffer, "Using Genetic Search to Exploit the Emergent Behavior of Neural Networks," Physica D, v. 42, no. 1-3, pp. 244-248, 1990.

[4] John Moody and Christian J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units,", Neural Computation, Massachusetts Institute of Technology, 1989.

[5] Alan F. Murray, Dante Del Corso, Lionel Tarassenko, "Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques," IEEE Transactions on Neural Networks, v. 2, no. 2, 1991.

[6] David E. Goldberg, Genetic Algorithms in Search, Optimization, & Machine Learning, Addison-Wesley, New York, 1989.

[7] Maureen Caudill and Charles Butler, Naturally Intelligent Systems, MIT Press, Cambridge, Massachusetts, 1989.

[8] Valentino Braitenberg, VEHICLES Experiments in Synthetic Psychology, Cambridge, Massachusetts, 1987.

[9] J.H. Holland, "Robust algorithms for adaptation set in a general formal framework," Proc. IEEE Symposium on Adaptive Processes - Decision and Control, v. 17, no. 5, pp. 1-5, 1970.

[10] J.H. Holland, "Processing and processors of schemata," Associative Information Processing, American Elsevier, New York, pp. 127-146, 1971.

[11] J.H. Holland, "Schemata and intrinsically parallel adaptation," Proc. NSF Workshop on Learning Systems Theory and its Application, University of Florida, Gaineville, Florida, pp. 43-46, 1973.

[12] David E. Goldberg, "Genetic Algorithms and Walsh Functions and Walsh Functions: Part I, A Gentle Introduction," Complex Systems, v. 3, pp. 129-152, 1989.

[13] David E. Goldberg, "Genetic Algorithms and Walsh Functions and Walsh Functions: Part II, Deception and its Analysis," Complex Systems, v. 3, pp. 153-171, 1989.

[14] W.S. McCullock and W.H. Pitts, "A logical calculus of ideas immanent in nervous activity," Bull. Math. Biophys., v. 5, pp. 115-133, 1943.

[15] D.O. Hebb, Organization of Behavior, Wiley and Son, New York, 1949.

[16] Bruce G. Mathew, "A Comparison of Neural Network and Other Pattern Recognition Approaches to the Diagnosis of Low Back Disorders," Neural Networks, v. 3, pp. 583-591, 1990.

[17] Sukhan Lee and Rhee M. Kil, "A Gaussian Potential Function Network with Hierarchically Self-Organizing Learning," Neural Networks, v. 4, pp. 207-224, 1991.

[18] Maureen Caudill, "Evolutionary Neural Networks," AI Expert, March 1991.

[19] Darrell Whitley and Timothy Starkweather, "Optimizing Small Neural Networks Using a Distributed Genetic Algorithm," Technical Report CS-89-114, Colorado State University, Fort Collins, Colorado.

[20] Borut Maricic and Zoran Nikolov, "GENNET - System for Computer Aided Neural Network Design Using Genetic Algorithms," Proc. International Joint Conference on Neural Networks, v. 1, pp. 102-105, 1990.

[21] J.D Schaffer, "Using Genetic Search to Exploit the Emergent Behavior of Neural Networks," Physica D., v. 42, no. 1-3, pp. 244-248, 1990.

[22] D. Whitley, "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity," Parallel Computing, v 14, no. 3, pp. 347-371, 1990.

[23] Hiroaki Kitano, "Design Neural Networks Using Genetic Algorithms with Graph Generation System," Complex Systems, v. 4, pp. 461-476, 1990.

[24] B. Widrow and S.D. Stearns, Adaptive Signal Processing, Prentice-Hall, New Jersey, 1985.

[25] Patrick K. Simpson, Artificial Neural Systems, Peragon Press, p. 89, 1990. [26] Kurt Hornik, "Approximation Capabilities of Multilayer Feedforward Networks," Neural Networks, v. 4, pp. 251-257, 1991.

[26] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations, MIT Press, 1986.

[27] David E. Goldberg, "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," Complex Systems, v. 4, pp. 415-444, 1990.

VITA

Timothy Daniel Choate was born in Linden, Tennessee on May 28, 1964. He attended elementary school at Linden Elementary and graduated from Perry County High School in May 1982. He entered Middle Tennessee State University in the August of 1982 and graduated in the May 1986 with a Bachelor of Science degree in Computer Science. In May of 1986 he accepted employment with OAO Corporation, and in August of 1986, he began his graduate studies at the University of Tennessee Space Institute at Tullahoma. In August 1991, he received his Master of Science degree in Computer Science.

Mr. Choate is still employed with OAO Corporation and resides in Murfreesboro, Tennessee with his wife Tammy and son Chason.