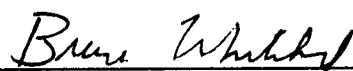


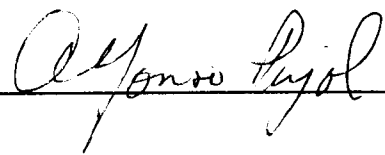
To the Graduate Council:

I am submitting herewith a thesis written by Derek Michael Ealy entitled "Adaptive Learning Rate Techniques for Neural Networks". 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, Major Professor

We have read this thesis
and recommend its acceptance:



DP menta

Accepted for the Council:



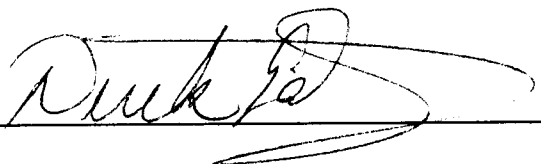
Associate Vice Chancellor and
Dean of the Graduate School

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 this 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

8/16/93

**ADAPTIVE LEARNING RATE TECHNIQUES FOR NEURAL
NETWORKS**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Derek Michael Ealy

December 1993

Copyright © Derek Michael Ealy, 1993

All rights reserved

DEDICATION

This thesis is dedicated to my mother and father
to whom I owe *everything*, and will never be able to
thank enough for all of their years of love and support.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Bruce Whitehead for all of the help and encouragement he gave me during the past two years. I would also like to thank Dr. Alfonso Pujol for his very thoughtful insights into the wording of this thesis. I am indebted to Dr. Dinesh Mehta for explaining the intricacies of preparing a thesis and its defense. I owe much thanks and appreciation to Jackie Benzwi for standing by me through the final years of my education. She has been most patient and helpful no matter how often it seemed that my progress toward the completion of my graduate career seemed to be taking longer than we would have liked. And finally I say thank you to Robert "Roo" Mauck for his help whenever the mathematics got to be a little too much for me.

Abstract

Neural networks that use the Least Mean Squared learning rule or the Generalized Delta Rule require the proper selection of a learning rate parameter to assure good convergence while being trained. This thesis discusses algorithms that modify the learning rate as the network is being trained, while still allowing for good convergence. Radial basis function networks and backpropagation networks were used for the development and testing of these adaptive algorithms. Research shows that modifying the learning rate for gradient descent techniques based upon the history of the normalized error can eliminate the need for the guesswork required to select a good static learning rate. Additionally, it was found that for a given number of training epochs, an adaptive learning rate algorithm can improve a neural network's convergence towards the global minimum of its error surface when compared to a static learning rate algorithm.

Symbol Table

η	eta - the middle to output layer learning rate
ρ	rho - the input to middle layer learning rate
σ	sigma - the standard deviation of the desired output
p	the number of training vectors
q	the number of nodes in input layer (input dimensions)
r	the number of nodes in middle layer
s	the number of nodes in output layer (output dimensions)
$\vec{x}$	input training vector
x_k	the k th member of $\vec{x}$
$\vec{v}_j$	weight vector to middle layer node j
$\Delta \vec{v}_j$	input to middle weight vector delta (amount of change)
V	matrix of input to middle layer weight vectors
y_j^{in}	input to the j th node of the middle layer
$\vec{y}^{out}$	middle layer output vector
y_j^{out}	the j th member of $\vec{y}^{out}$
ϵ_{yj}	the error attributed to j th node of the middle layer
$\vec{w}_j$	weight vector for middle node j connecting to the output layer
$\vec{w}_i$	weight vector for output node i connecting to the middle layer
$\Delta \vec{w}$	middle to output weight vector delta (amount of change)

W	output layer weight vector matrix
$\vec{z}^{des}$	desired output vector
$\vec{z}^{out}$	computed output vector
$\vec{z}^{err}$	output layer error vector
z_i^{err}	error for output node i
$\bar{\epsilon}_z$	error between middle and output layer in backprop networks
ϵ_{zi}	error for node i of the output layer in backprop networks
$\bar{c}_m$	radial basis function center vector
$\bar{\omega}_m$	width of RBF m
ΔErr	change of error from one epoch to the next

List of Figures

Figure 2.1.1:	A MSE surface with both local and global minima.	5
Figure 2.1.2:	When the learning rate is too small the network is unable to jump out of local minima.	6
Figure 2.1.3:	When the learning rate is too large the network's error oscillates above the global minimum.	7
Figure 2.1.4:	When the learning rate is of the appropriate magnitude the network converges toward the global minimum.	8
Figure 3.1.1:	Radial Basis Function centered on a one dimensional point.	10
Figure 3.1.2:	An example of how RBFs are used to approximate the curve of a function.	13
Figure 3.2.1:	The sigmoidal response function.	15
Figure 3.2.2:	The derivative of the sigmoidal response function.	17
Figure 5.2.1:	Normalized error at each epoch for a BP network with the learning rate held at 0.8.	28
Figure 5.2.2:	An OCR bitmap.	30
Figure 6.2.1:	Mackey-Glass chaotic time series, 500 steps with delay $\tau = 17$	33
Figure 6.2.2:	A RBF network's normalized error vs. epoch for a static learning rate of 0.8.	34
Figure 6.2.3:	A RBF network's normalized error vs. epoch for a static learning rate of 0.3.	35
Figure 6.2.4:	A RBF network's normalized error vs. epoch with a static learning rate of 0.0005.	36
Figure 6.2.5:	A RBF network's normalized error vs. epoch when using the linear decay learning rate.	36

Figure 6.2.6: A RBF network's normalized error vs. epoch when using the moving target algorithm.	37
Figure 6.3.1: Normalized error for a BP network with a static learning rate of 0.3.	39
Figure 6.3.2: Normalized error for BP network using the linear decay algorithm.	40

List of Abbreviations

ALRA	Adaptive Learning Rate Algorithm
RBF	Radial Basis Function
LMS	Least Mean Square
BP	Backpropagation
PE	processing element (node)
MSE	mean squared error
RMS	root mean squared
GDR	Generalized Delta Rule
NE	normalized error
OCR	optical character recognition

Table of Contents

Chapter	Page
I Introduction	1
1.1 Neural Networks	1
II Problem Analysis	4
2.1 Choosing a Good Learning Rate	4
III Neural Network Background and Literature Review	9
3.1 Radial Basis Function Networks	9
3.2 Backpropagation Networks	13
3.3 Literature Review	18
IV Gradient Descent Learning Techniques	20
4.1 Least Mean Squared Learning Rule	20
4.2 Generalized Delta Rule	21
V The Adaptive Learning Rate Algorithm	24
5.1 Applying ALRA I to RBF Networks	24
5.2 Applying ALRA II to Backpropagation Networks	28
VI Results and Conclusion	32
6.1 Computer Systems Overview	32
6.2 Radial Basis Function Network Results	33
6.3 Backpropagation Results	38
6.4 Conclusion	40

List of References	43
VITA	46

CHAPTER I

Introduction

1.1 Neural Networks

Neural networks are generally thought of as information processing systems that are modeled on the biological mechanics found in the brain that control an animal's organs and thought processes. Artificial neural networks are similar to nature's creations in that both are systems of single processing elements (PEs) that are interconnected and operate in parallel while offering distributed computation and data storage. Neural networks theoretically operate in parallel, but their implementation is typically on serial computers. Among the first to research artificial neural systems were McCulloch and Pitts, who, with their knowledge of neurology, modeled neural networks in the early 1940's without the benefit of computers. Their work was based upon neurological research done in the late 19th and early 20th centuries [1].

There are two basic components that make up neural networks. The first is the processing element, sometimes called a neuron in biological terms, or a node in artificial neural system terminology. The second component is the interconnection between a PE and all of the PEs in the next layer. In biological terms this is called a synapse, while in neural networks it is simply called a connection. A PE takes an input, processes it and outputs a value. This value is passed along the connections from that particular node to all nodes in the next layer of the network. Each connection has a weight associated with it. This weight modifies the output of the previous node before passing the value on to the next node. This research concentrates on two types of feedforward networks: radial

basis function (RBF) networks and backpropagation (BP) networks. In these types of neural networks, learning is accomplished by modifying the connection weights such that the output of the whole network closely approximates the desired output of the network for a given input. This procedure is called supervised learning. Biological neural systems are not trained in this manner. They are able to learn from their given inputs without knowing what the correct response should be. This procedure is called unsupervised learning. There are many other types of neural network architectures in use, but they are beyond the scope of this study. See [1] and [2] for introductions to neural networks.

Neural networks that employ supervised learning techniques based upon gradient descent algorithms require that a suitable value be chosen for the learning rate. It is difficult to make a guess as to what would be a proper learning rate until there have been several trial and error runs of the network with different learning rate values. The ability to analyze the performance of a network for a given training data set and learning rate requires a certain degree of expertise that comes with the experience of using neural networks to solve different types of problems. An algorithm that would automate the analysis of the network, and pick a learning rate that is appropriate for any given time during the network's training phase, could be beneficial to many of those who spend time searching for a "good" learning rate.

This research presents algorithms that modify the learning rate while the network is being trained. The goal of this research is to eliminate the preliminary training runs of the network that are usually required to nail down a static learning rate that strikes a good balance between the neural network's speed of convergence and accuracy. Also, this thesis

will analyze whether networks trained with the adaptive learning rate algorithm (ALRA) can learn faster than networks with well chosen static learning rates.

Chapter II analyzes the problems with gradient descent learning methods that this research is concerned with. Chapter III presents a literature review and background information on radial basis function networks, and backpropagation networks. Chapter IV analyzes the gradient descent learning techniques that ALRA was applied to for this research. Chapter V details the adaptive learning rate algorithm. Chapter VI presents the results and conclusion obtained from this research in applying ALRA to the two neural network architectures described in Chapter III.

CHAPTER II

Problem Analysis

This chapter discusses some of the problems that are inherent to gradient descent learning techniques such as the least mean squared learning rule and the generalized delta rule. It examines the consequences of choosing a poor learning rate, and explains how a neural network behaves when such a choice is made.

2.1 Choosing a Good Learning Rate

Choosing an appropriate learning rate is important because it will determine the level of accuracy the network can converge to in a reasonable amount of training time. Most neural network implementations use a learning rate that does not change while the network is being trained. Choosing a fixed learning rate is difficult because there are no hard and fast rules for determining one that would put a network on the road to rapid convergence. A suitable learning rate varies depending upon the type of problem being solved and upon the initial state of the network. For example, the learning rate for a backprop network learning the XOR problem need to be around 0.8, while for the double-spiral problem [3], the learning rate should be on the order of 0.002 [4].

The learning rate is typically arrived at after several test runs of the network with different learning rates. The user observes how the network behaves with each one and then begins the process of zeroing in on a choice that allows for good convergence within a reasonable period of time. Pondering this trial and error search for the learning rate suggests that an algorithm that can adjust the learning rate during the training phase

would save many unneeded runs of the network and potentially reduce the training time of the network. An adaptive learning rate algorithm could reduce the training time of the network by allowing the network to take very large steps at the beginning of the learning period via a large learning rate. To keep the network converging towards an optimal configuration the algorithm would then reduce the learning rate as needed, thus avoiding oscillations that occur when the learning rate is too large.

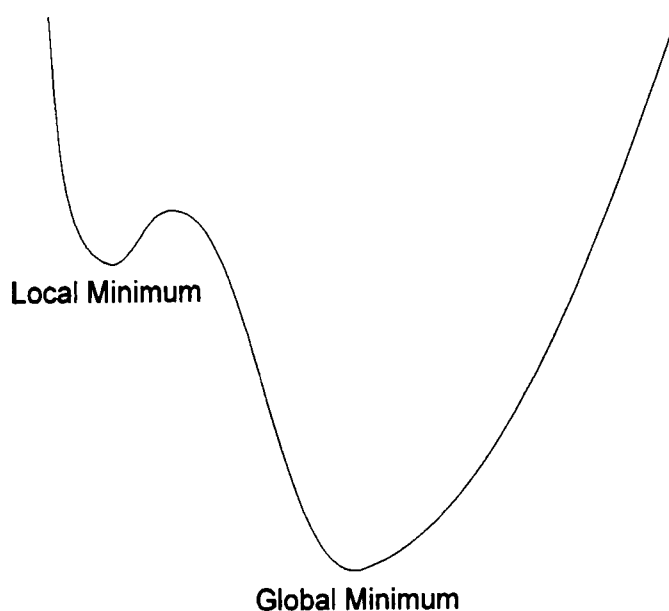


Figure 2.1.1: A MSE surface with both local and global minima.

It is easy to visualize the problems of gradient descent methods if we examine a simplified two dimensional graph of a mean squared error (MSE) surface. Figure 2.1.1 illustrates an error surface that has a local and global minimum. Figure 2.1.2 shows how the MSE of a neural network might change in the presence of a local minimum when the learning rate is too small. In order for a neural network to converge to the global minimum, a fixed learning rate must be large enough so that it does not oscillate within

the local minimum and small enough so that the MSE of the network can reach a low point within the global minimum. In the case of Figure 2.1.2 the MSE eventually is confined within the local minimum and will never be able to recover, since the gradient descent algorithms move the MSE down the slope of the error surface. In this instance the learning rate is too small. If the learning rate were large enough the change of the MSE would have jumped over the local minimum as in Figure 2.1.3.

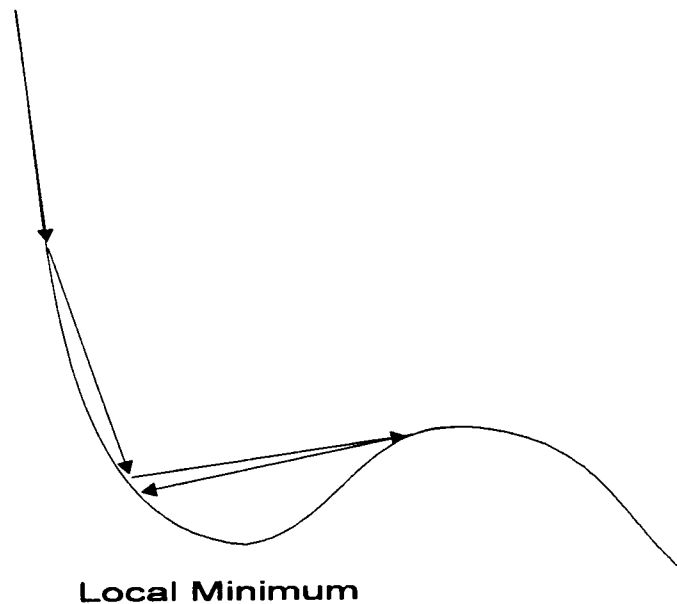


Figure 2.1.2: When the learning rate is too small the network is unable to jump out of local minima.

At the same time Figure 2.1.3 illustrates the behavior of the MSE when the learning rate is too large for the given problem. In this example the first reduction of the MSE goes past the local minimum and subsequently goes past the global minimum and up the other side of the error surface. With such a large learning rate the MSE continues

to oscillate well above the global minimum, resulting in a neural network that might not ever train to within some reasonable error level.

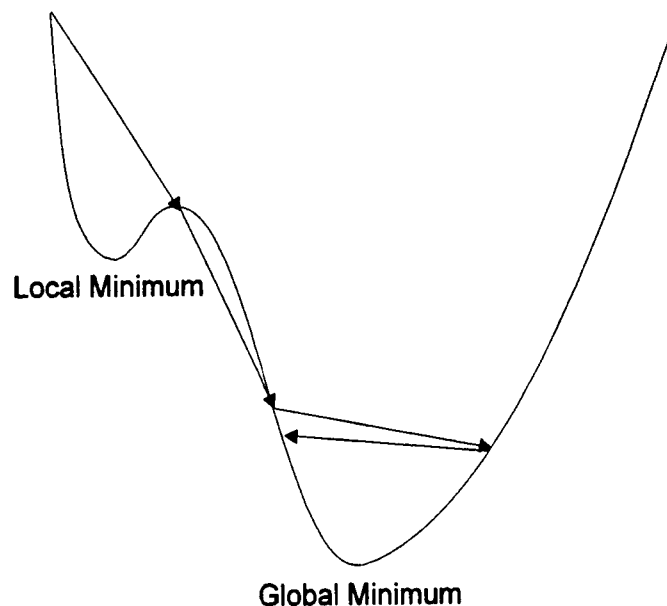


Figure 2.1.3: When the learning rate is too large the network's error oscillates above the global minimum.

Figure 2.1.4 is an example of the reduction of the MSE with a well chosen learning rate. The step size of the MSE reductions is large enough to get past the local minimum, and small enough so that it does not jump beyond the global minimum and continue to oscillate over it. These are the difficulties ALRA was designed to overcome. The idea behind ALRA is as follows. Start the network with a large learning rate, e.g. 0.8. While the network is training keep track of the progress the delta rule makes in reducing the MSE. If, at the completion of an epoch (one pass through the training data set) the MSE error goes up, then the previous jump was too large, i.e. it passed over the global minimum. In this case, reduce the learning rate and reset the weights of the network to

their values prior to the last epoch. Conceptually, ALRA is moving the MSE back to the point before the last epoch and reducing how far it will jump the next time. ALRA will continue to back up and reduce the learning rate until the MSE decreases and avoid jumping past the global minimum.

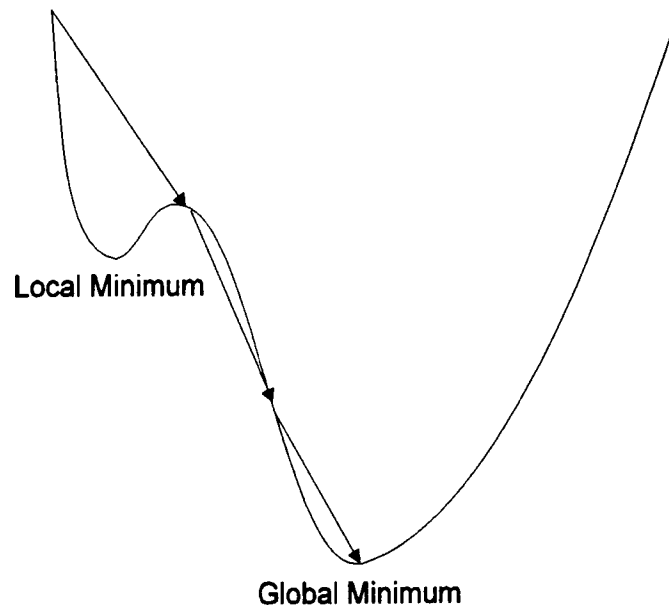


Figure 2.1.4: When the learning rate is of the appropriate magnitude the network converges toward the global minimum.

CHAPTER III

Neural Network Background and Literature Review

This chapter gives a general overview of the RBF and BP neural networks, and also surveys literature relevant to speeding network convergence by modifying the learning rate during the training phase.

3.1 Radial Basis Function Networks

Neural networks modeled after the multilayer perceptron [5] require activity in every PE for each input of the training data set. For large networks or large training data sets this can take up a significant amount of processing time. A network architecture which only requires activity in the PEs that have detectable activation for a given input would reduce the amount of time required to train the network. The Radial Basis Function neural network is such an architecture, which has a 3 layer topology, with the middle layer made up of several locally tuned nodes. The output layer is connected to the middle layer via a standard weighting mechanism and trained with the Least Mean Squared (LMS) learning rule. Among the first to apply the RBF technique to neural networks were Broomhead and Lowe [6] and Moody and Darken [7].

Each PE in the middle layer has a response function which is radially symmetric and produces its highest output when the input is closest to its center, while decreasing monotonically as the distance of the input from the center of the function increases [8]. Figure 3.1.1 is a graph of a RBF centered at point A. If the input to this RBF is either less than or greater than A, the RBF will output a value less than its maximum of 1 that

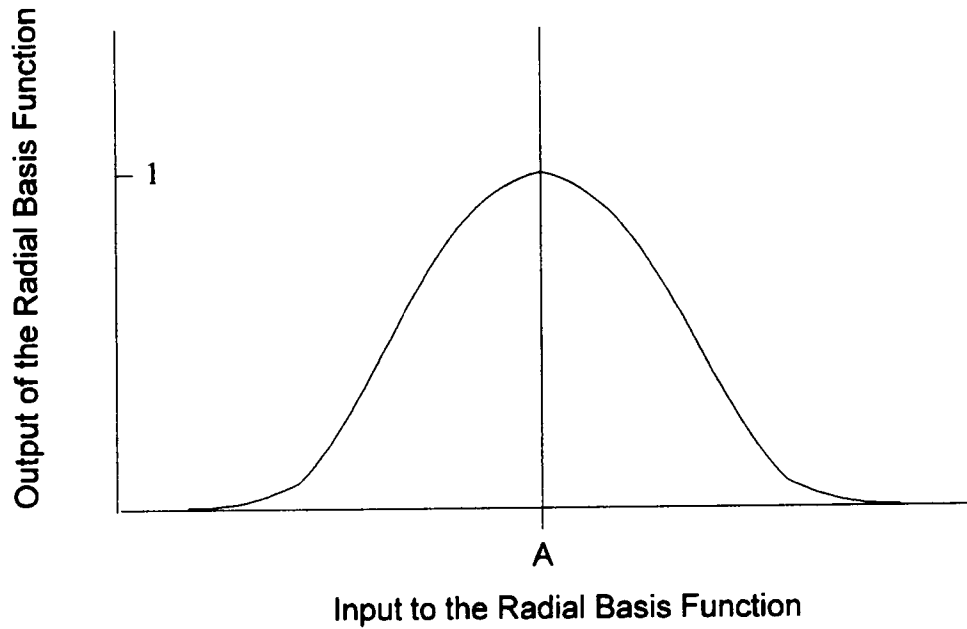


Figure 3.1.1: Radial Basis Function centered on a one dimensional point.

falls off with distance from the center A. For multi-dimensional inputs the center of the RBF would have the same number of dimensions. A RBF graph centered on a two dimensional point would be in the shape of a bell. In multi-dimensional instances the difference between the input and the RBF center is their Euclidian distance from one another. The response function generally chosen for RBF networks is:

$$R(\vec{x}) = \exp \left[- \frac{\|\vec{x} - \vec{c}_m\|^2}{(\tilde{\omega}_m)^2} \right] \quad (3.1.1)$$

where $\vec{c}_m$ is the center and $\tilde{\omega}_m$ is the width of the radial basis function m .

The width of the RBFs can be determined by various "P nearest neighbor techniques" [7]. These techniques allow the widths of the RBFs to vary such that the

RBFs overlap one another, resulting in a smooth and contiguous interpolation of the regions they represent in the input space [7]. The RBF network used for this research did not attempt to modify each RBF width individually. Instead, an algorithm was used to calculate the average distance between each RBF and its nearest neighbor. This distance was used as the width of all RBFs in the network. The width of the RBF is important because it is now generally accepted that the generalization performance of the network is affected by a trade-off between the number of training examples in the data set, and the capacity of the learning system [9]. In the context of the local learning algorithms, the effective number of training examples is constrained by the width of the RBFs [10]. This is because as the width of the RBF becomes narrower, the number of training examples that are close enough to its center to evoke a significant response is reduced, resulting in increased performance relative to the number of training examples.

The purpose of the RBF network is to speed the learning process by using PEs that will only respond to input data that is relevant. In order to accomplish this, the RBF centers must be placed in the vicinity (in terms of Euclidian distance) of relevant data. This is usually accomplished via self organizing clustering methods. Moody and Darken suggested the standard k -means clustering algorithm [11] [12], or a Kohonen self-organizing feature map neural network [13]. For this research the following merge-clustering algorithm was used:

1. Find the two vectors whose euclidian distance from one another is the smallest of the data set.
2. Average the two vectors together to determine the midpoint between them.

3. Remove the two vectors from the data set, and replace them with the vector calculated in step 2.
4. Repeat steps 1-3 until the desired number of cluster centers is reached.

These clustering methods pre-process the input vectors into a high dimensional space, aiming to exploit the fact that a classification problem cast in high dimensional space is more likely to be linearly separable than would be the case in a low dimensional space [14]. An example of this would be to place a RBF function at each input vector (this is usually computationally infeasible), resulting in a "perfect" network. Simply stated, the RBF nodes should be placed at points about which the input vectors cluster, thereby allowing them to respond appropriately (only to those inputs that are near them). A RBF node that is located away from any of the input vectors is useless, while multiple RBF nodes placed within the same locality is wasteful.

Once the RBF centers have been appropriately placed in the input space it is the job of the LMS learning rule (detailed in chapter 4.1) to modify the weights connecting the middle layer to the output layer. This has the effect of modifying the amplitude of the RBFs so that the network is able to successfully map the input data to the desired output level (see Figure 3.1.2).

Using RBF networks is difficult because there are several aspects of the network's configuration that require the user to guess or experiment: choosing the number of RBFs, picking a good clustering algorithm, determining the width that each RBF should be, and picking a learning rate that will allow good convergence and a reasonable training period. This research concentrates on developing an algorithm that alleviates the need to choose a good learning rate for the LMS rule.

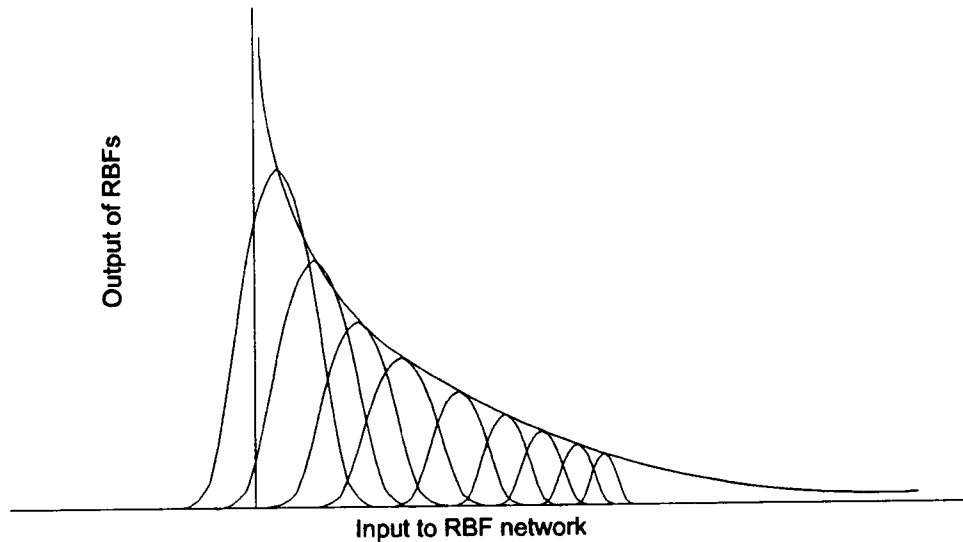


Figure 3.1.2: An example of how RBFs are used to approximate the curve of a function.

3.2 Backpropagation Networks

Backpropagation neural networks are an outgrowth of the multilayer perceptron which is based on Widrow's early work [5]. The BP network is typically a three (or more) layer feedforward architecture that employs a different thresholding function, and most importantly an extension of the LMS learning rule, called the Generalized Delta Rule (GDR). As pointed out by Minsky and Papert [15], the standard multilayer perceptron is unable to solve problems that are not linearly separable. The GDR enables the multilayer feedforward architecture to handle such problems. According to Simpson, backpropagation was discovered and rediscovered by several different people over the past three decades:

The first to apply gradient descent methods to training multilayered perceptrons was Amari [16], who introduced a single hidden layer PE to perform nonlinear classification. Bryson and Ho [17] developed an algorithm similar to backpropagation for nonlinear adaptive control. Werbos [18] discovered backpropagation and several variants. Parker [19] also rediscovered backpropagation calling it -- algorithm learning logic.

However, backpropagation did not get noticed until Rumelhart, Hinton and Williams [20] began using backpropagation in their research. [21]

What actually makes a neural network a BP network is not the way that the PEs are laid out and connected to one another, but the method by which the network learns.

The input to the BP network passes through the first layer to the middle layer via the weighted connections. The weight of each connection is multiplied by the output of the PE that is passing a value through it. Each PE in both the middle layer(s) and the output layer has a connection to all of the PEs in the preceding layer. The group of connections bringing all of the outputs from the previous layer to a PE is called the weight vector. The input for each PE in the middle layer can be expressed as the dot product of the input vector and the weight vector. The following is an equation that calculates the input to a middle layer PE y_j^n :

$$y_j^n = \vec{x} \cdot \vec{v}_j \quad (3.2.1)$$

where $\vec{x}$ and $\vec{v}_j$ respectively are the input vector and the middle layer weight vector.

$$f(x) = \frac{1}{(1 + e^{-x})} \quad (3.2.2)$$

The output of the PEs in the middle and output layers is determined by the response function given by Equation 3.2.2. This function is sigmoidal, monotonically increasing, and it asymptotically approaches fixed values as the input approaches positive or negative infinity (see Figure 3.2.1). These characteristics of the response function are

necessary for BP to work (another function that has these qualities could also be used in a BP network). A function such as this is needed for the GDR to work correctly.

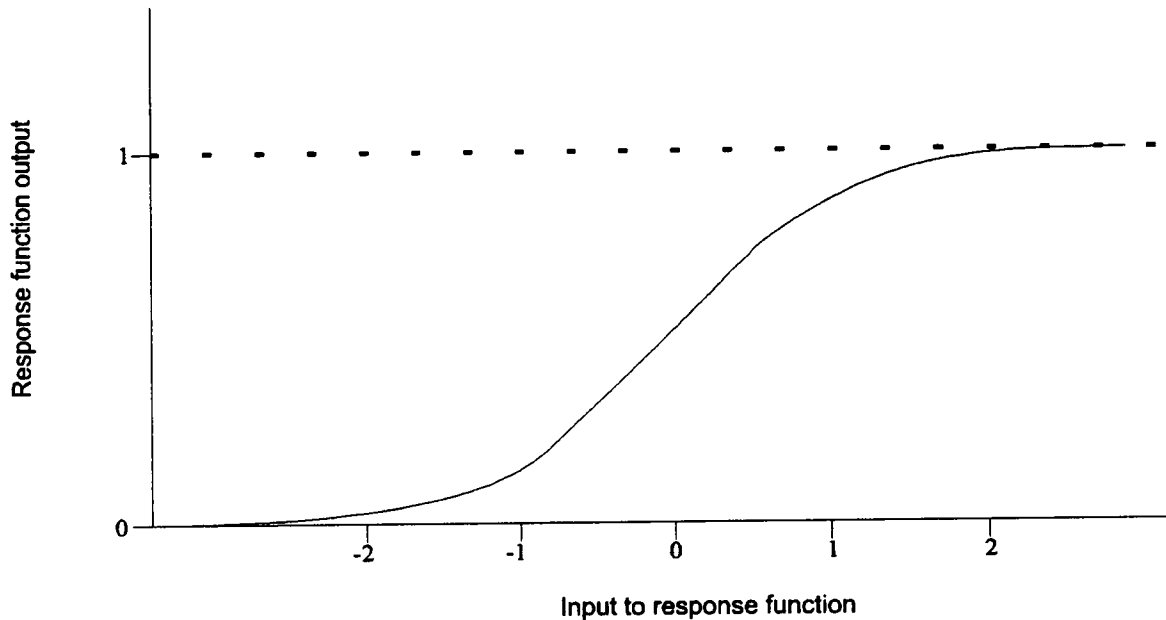


Figure 3.2.1: The sigmoidal response function.

The GDR is what makes the BP network so powerful. It gives the network the ability to determine to what degree the middle layer nodes are responsible for the total error generated by the network. This in turn gives the algorithm the information needed to correctly modify the weights connecting the input layer to the middle layer. With the ability to appropriately adjust nodes in the middle layer, a BP network will be able to solve problems that are not linearly separable. The capability to make such complex distinctions is due to the feature-detection and generalization abilities which are trained into the middle layer [1].

The GDR modifies the weights of both the middle and output layers. For the output layer, the difference between the actual output of a node and its desired output is

multiplied by the derivative of the sigmoidal response function and then again by the learning rate to determine the amount of change to make to the middle to output layer weight vector. The calculation of the output error vector is as follows:

$$\vec{\epsilon}_z = f'(\vec{z}^{out}) (\vec{z}^{des} - \vec{z}^{out}) \quad (3.2.3)$$

Adjusting the weights between the input and middle layers requires a similar approach. The GDR propagates the error back through the connections between the middle and output layers. To determine the amount of error to assign to each PE of the middle layer the following equation is used:

$$\epsilon_{y_j} = f'(y_j^{out})(\vec{w}_j \cdot \vec{\epsilon}_z) \quad (3.2.4)$$

where $\vec{w}_j$ is the weight vector connecting middle layer node j to the output layer and $f'(x)$ is the derivative of the sigmoidal response function.

The derivative of the sigmoidal response function in Equation 3.2.2 is:

$$f'(x) = f(x)(1 - f(x)) \quad (3.2.5)$$

Once the amount of error for a particular middle layer node has been calculated it is multiplied by the learning rate and used to modify the input to middle layer weight vector for that particular node. A detailed explanation of the GDR and its derivation can be found in Rumelhart, Hinton and Williams [19].

The sigmoidal response function's importance in BP is that its derivative approaches zero for large positive and negative values, while its maximum occurs when the input is close to zero (see Figure 3.2.2). These characteristics prevent too much blame

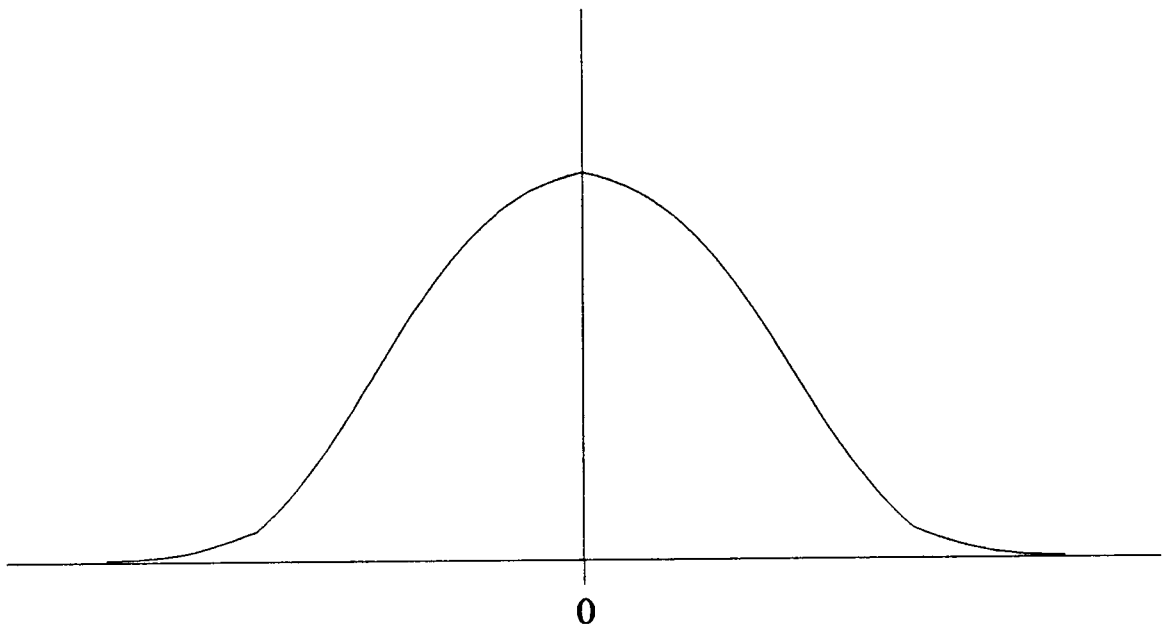


Figure 3.2.2: The derivative of the sigmoidal response function.

for the output error from being assigned to a middle layer node. When the output of a middle layer node is either 0 or 1, the PE is in a stable state (see Figure 3.2.1). If the output of a middle layer node is in the transitional phase of the sigmoid (around 0.5) the PE is in an unstable state. The middle layer nodes should be in a stable state once the network is trained, so when a PE is unstable the weights to that middle layer node should be changed. The derivative of the response function will cause the greatest change in the weight vector whenever the output of the response function is in this middle range. In Figure 3.2.2 the maximum output of the derivative of the response function will occur when the input to the response function is near zero. This illustrates how the derivative function will indeed cause the greatest change in the input to middle layer weight vector when the PE is in an unstable state. There is a trade-off in this ability to properly modify the middle layer: since the weight changes are proportional to the derivative, a weight

change may occur more slowly than desired for PEs which yield either small or large responses, which can cause difficulties in training the network [1].

There are two main areas of difficulty in constructing a BP network. The first is choosing how many PEs will make up the middle layer(s). It is important to make a good choice because too many PEs in the middle layer will cause the network to memorize the training patterns. Memorizing the patterns causes the network to lose its ability to generalize the input data. When this happens the network is not able to respond correctly to data that it was not trained with. Too few PEs in the middle layer will prevent the network from performing well in any respect. The second problem with BP networks is choosing a good value for the learning rate. If the learning rate is too small the network will get stuck in a local minimum, and if it is too large the network will not converge to a good error level, due to the oscillation of the MSE.

3.3 Literature Review

The remainder of this research is aimed at developing an algorithm that can adjust the learning rate adaptively while the network is being trained, thereby removing the guesswork from choosing a good learning rate value. There are several others who have worked on ways to improve the training time of neural networks through modification of the learning rate. Jacobs [22] developed the delta-bar-delta heuristic, in which a learning rate is assigned to each weight in the network. The learning rates vary over time based on the sign changes of the network's error. Gluck, Glauthier, and Sutton [23] extended the delta-bar-delta technique, by adding a "cue-salience or attentional model" to the LMS

learning rule. Giving the network the ability to modify the learning rates when it determines whether an input is relevant or not. This is determined by the correlation of successive weight changes. Janakiraman and Honavar [24] developed an heuristic for adapting the learning rate by computing the error three times for each epoch, with the learning rate at values of x , $x/2$, and $2x$, and choosing the one that produced the smallest error. Fahlman [25] developed the Quickprop algorithm which modifies the derivative of the sigmoid response function, and stops the training of PEs which have a quadratic error of less than 0.01. Cater [26] developed an heuristic which doubles the learning rate for the training vector which caused the largest error in the previous epoch, while keeping a smaller learning rate for the other training vectors. The smaller learning rate is reduced by one half whenever the total network error increases by more than 1%. Salomon [4] dynamically adapted the learning rate by computing the error for a smaller learning rate and a larger learning rate at the end of each epoch, and choosing the one that generated the lower error sum.

CHAPTER IV

Gradient Descent Learning Techniques

This chapter presents the algorithms for the least mean squared learning rule and the generalized delta rule which are employed in RBF and BP networks respectively. For conciseness the mathematics are presented in vector notation.

4.1 Least Mean Squared Learning Rule

The Least Mean Squared (LMS) learning rule was developed by Widrow and Hoff, and first published in 1960 [5]. This rule is used to adjust the weights in the RBF networks described above. The difference between the actual and desired output of each PE in the output layer is used in conjunction with the slope of the MSE surface to calculate by how much to change the weights of that PE, and in which direction to change them. Here is the algorithm as it applies to the training phase of a RBF network, from Simpson [20]:

1. Initialize the weights between the middle and output layers of the network. For network architectures with more than two layers the weights should be non-zero and randomized (so the network can differentiate the hidden PEs). The weights were initialized to zero for the RBF network used in this research (because RBF networks are trained like two layer perceptrons).
2. For each training pair $\vec{x}$ and $\vec{z}^{des}$ (training vector and desired output vector) do:
 - a. Pass $\vec{x}$ through the Gaussian functions making up the middle layer of the network. The output of the middle nodes $\vec{y}^{out}$ is then multiplied by $\vec{W}$, the matrix of weight vectors. The result is the output vector $\vec{z}^{out}$.

$$\vec{z}^{out} = W \cdot \vec{y}^{out} \quad (4.1.1)$$

- b. Calculate the error between $\vec{z}^{out}$ and $\vec{z}^{des}$

$$\vec{z}^{err} = \vec{z}^{out} - \vec{z}^{des} \quad (4.1.2)$$

- c. Adjust the weights of each PE in the output layer

$$\Delta \vec{w}_i = \eta (z_i^{err}) \vec{y}^{out} \quad (4.1.3)$$

$$\vec{w}_i(t+1) = \vec{w}_i(t) + \Delta \vec{w}_i \quad (4.1.4)$$

the learning rate η , is a positive number, where $0 < \eta < 1$. The LMS rule adjusts the weights by doing gradient descent on the n-dimensional MSE surface, this is done by calculating the gradient of the error with respect to the weights. $\vec{w}_i(t)$ is the weight vector at time t .

3. Repeat step 2 until z_i^{err} for each output node is either equal to 0 or small enough to satisfy the user's need.

4.2 Generalized Delta Rule

BP networks require a more complex algorithm than the LMS learning rule. A BP network is made up of three or more layers. Each layer between the input and output layers has a set of weight vectors connecting it to the previous layer. The basic LMS learning rule has no provision for modifying the weight vectors of the internal layers. The Generalized Delta Rule (GDR) has such a provision. The LMS learning rule is also called the Delta Rule, but that appellation is avoided in order to reduce confusion with the GDR. Here is Simpson's version of the GDR algorithm for BP networks in vector notation [20]:

1. Assign random values in the range of $[-x, +x]$ to all weight vectors $\vec{v}$ between the first and second layers, to all weight vectors $\vec{w}$ between the middle and output layers, and to each PE threshold in the middle and output layers.
2. For each training pair $\vec{x}$ and $\vec{z}^{des}$ do:
 - a. Transfer $\vec{x}$ through the first layer and V , the matrix of input to middle layer weight vectors, and calculate the activations for the PEs in the middle layer using this equation:

$$\vec{y}^{out} = f(V \cdot \vec{x}) \quad (4.2.1)$$

where $f()$ is the sigmoid threshold function $f(x) = (1 + e^{-x})^{-1}$.

- b. Pass the middle layer output vector $\vec{y}^{out}$, through the weight vector matrix W to the output layer and calculate output vector $\vec{z}^{out}$ using the following equation:

$$\vec{z}^{out} = f(W \cdot \vec{y}^{out}) \quad (4.2.2)$$

- c. Compute the effective error for each output node (this is the difference between the computed output and desired output for each PE multiplied by the derivative of the response function) as shown below:

$$\vec{\epsilon}_z = f'(\vec{z}^{out}) (\vec{z}^{des} - \vec{z}^{out}) \quad (4.2.3)$$

$$f'(x) = f(x)(1 - f(x)) \quad (4.2.4)$$

where $f'(x)$ is the derivative of the sigmoid response function (see Equation 3.2.2).

- d. Compute the error that each middle layer node y_j^{out} is responsible for by filtering the output error $\vec{\epsilon}_z$ backwards through the output layer weight vector $\vec{w}_j$ and multiplying the result by the derivative of the sigmoid transfer function acting on the activation of this middle layer node that produced the output error vector:

$$\epsilon_{yj} = f'(y_j^{out})(\vec{w}_j \cdot \vec{\epsilon}_z) \quad (4.2.5)$$

- e. Adjust the weight vectors connecting the middle and output layers:

$$\Delta \vec{w}_i = \eta(\epsilon_{zi}) \vec{y}^{out} \quad (4.2.6)$$

$$\vec{w}_i(t+1) = \vec{w}_i(t) + \Delta \vec{w}_i \quad (4.2.7)$$

where $\Delta \vec{w}_i$ is the amount the weight vector has to change by, and η is the learning rate.

- f. Adjust the weight vectors connecting the input and middle layers:

$$\Delta \vec{v}_j = \rho(\epsilon_{yj}) \vec{x} \quad (4.2.8)$$

$$\vec{v}_j(t+1) = \vec{v}_j(t) + \Delta \vec{v}_j \quad (4.2.9)$$

where $\Delta \vec{v}_j$ is the amount the input to middle layer weight vector has to change by, and ρ is the learning rate between the input and middle layers.

3. Repeat step 2 until z_i^{err} for each output node is either equal to 0 or small enough to satisfy the user's need.

CHAPTER V

The Adaptive Learning Rate Algorithm

This chapter presents a detailed description of two versions of ALRA. Version I applies ALRA to the LMS learning rule which was used in the RBF network examples in this research. Version II applies ALRA to the GDR used in the BP network for this research.

5.1 Applying ALRA I to RBF Networks

As stated above, the LMS learning rule adjusts the steps taken along the MSE surface according to the magnitude of the error (difference) separating the desired and computed output of the network for a set of training examples. The smaller the error the smaller the size of the step along the MSE surface. The proportion of the step size in relation to the error is determined by the learning rate η , and the difficulty lies in choosing what the learning rate should be. ALRA is designed to avoid the guesswork in choosing a good learning rate for a given problem. It manages the selection of the learning rate by comparing the MSE at the end of each training epoch with the MSE generated by the previous training epoch. If the MSE is no longer decreasing or if it remains unchanged, the learning rate is reduced. The following is a detailed description of how to apply the ALRA to radial basis function networks.

Since it is useful to calculate the normalized error of a RBF network at the end of each training epoch, we can use this as a measure of the current state of network on the MSE surface relative to its position on the MSE surface at the end of the previous

epoch. During each training iteration, the difference between the desired output and calculated output (the network error) is calculated. The following equation is used to calculate the MSE for a particular epoch:

$$MSE = \frac{\sum_{h=1}^p \sum_{i=1}^s (z_{hi}^{err})^2}{p} \quad (5.1.1)$$

where z_{hi}^{err} is the error generated by the i^{th} node of the output layer, in response to the h^{th} training pattern.

The RBF network's normalized error is often used as a quantitative indicator of the network's state of convergence. The normalized error is shown in the next equation:

$$NE = \frac{\sqrt{MSE}}{\sigma} \quad (5.1.2)$$

where σ is the standard deviation of the output. If the output vector is multi-dimensional then there will be a standard deviation and MSE for each node of the output vector.

At the end of each epoch, the change in normalized error, ΔErr , is calculated by subtracting the current normalized error from the normalized error of the previous epoch. If ΔErr is less than zero then the normalized error is greater than it was at the end of the previous epoch. Initially, whenever the normalized error was on the rise the learning rate was reduced. The reasoning behind this was that the previous change of weights resulted in a jump beyond the global minimum of the MSE surface. Experimentation showed that immediately reducing the learning rate usually did not result in good convergence of the network. The reason for this poor convergence was that the learning rate was decreasing

too fast in relationship to network's approach to the global minimum. This caused the change in weight vectors to be very small, resulting in unusably slow convergence.

In order to solve this problem a feature was introduced that would track rising normalized error trends. This would not modify the learning rate for minor perturbations of a descending normalized error pattern, but would trigger the reduction of the learning rate when there was a noticeable rising trend in the normalized error. Here is how the method was implemented. Before the training of the network began, a trigger value for the rising error count detector is selected. For the chaotic time series problem a value of 3 was used for the rising error trigger. This means that if the normalized error has increased from one epoch to the next for three consecutive epochs then the learning rate is reduced. Experimenting with rising error triggers ranging from 1 to 5 proved 3 to be the best choice. With values below 3 the learning rate became too small too fast, causing the network to converge too slowly. Values greater than 3 slowed the convergence of the network too much by allowing the network to stray too far off course before reducing the learning rate.

One of the characteristics of a RBF network is that for a given static learning rate the network will approach the global minimum, but rarely (if ever) hit it. When the learning rate is too large the normalized error will jump across the global minimum and for all practical purposes will jump back right across to the place where it was at the end of the previous epoch (it actually descends an infinitesimal amount). Conceptually the normalized error is bouncing back and forth between the same two places on the MSE surface (see Figure 2.1.3). Because of this characteristic it is necessary to detect when

ΔErr approaches zero (indicating the MSE is not being reduced anymore) and to then decrease the learning rate.

Initially a linear decay method was used to modify the learning rate, simply reducing it by a fixed factor, usually 3%. After some experimentation it was realized that picking the learning rate reduction factor was going to be problem dependent. In essence, the tricky learning rate selection was replaced with another choice that still involved guesswork but not as much. Preliminary results showed that a small learning rate reduction factor would allow good convergence toward the global minimum in a reasonable amount of time. These results also indicated that network convergence could be speeded up if the learning rate reduction factor varied as the normalized error decreased. The learning rate reduction factor needed to be reduced by a large amount during the beginning of the RBF network's training and then only reduced by a small amount as the network approached the global minimum of the MSE surface. This behavior is similar to the way the network's weight vectors are modified.

My advisor suggested a method that would reduce the learning rate by half the distance between the current learning rate and a moving target. If the learning rate starts out to be much larger than the target then the distance between the two will be relatively large. With each reduction of the learning rate it comes closer to the moving target. Since the learning rate is reduced by half the distance between it and the moving target then it will never hit the target. The moving target is reduced by a factor ranging from 0.1% to 0.5% at the end of each epoch. This creates a learning rate reduction schedule that calls for large reductions at the beginning of the training, and only allows small reductions

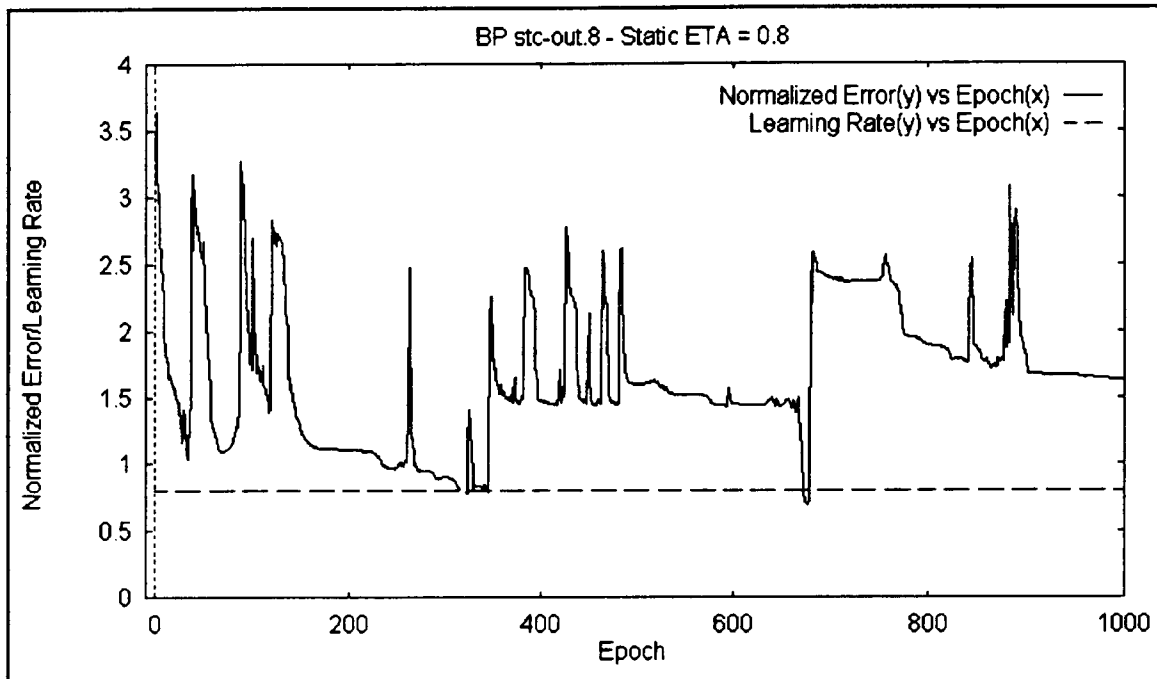


Figure 5.2.1: Normalized error at each epoch for a BP network with the learning rate held at 0.8.

once the learning rate comes into close proximity to the moving target. For example, if the learning rate is initially 0.8, and the moving target is initially 0.1, then if learning rate reduction is called for within the first few epochs the learning rate would be reduced by about 0.4. Eventually the learning rate will become very close to the moving target, and will then be effectively reduced by the rate of the moving target's reduction factor, thereby causing the small learning rate reductions that are desired when the normalized error is approaching the global minimum.

5.2 Applying ALRA II to Backpropagation Networks

ALRA I did not work when applied to backpropagation networks. The reason for this is that the error surface of a BP network is different than that of a RBF network.

When ALRA I is applied to BP networks, the descent of the normalized error is erratic. Figure 5.2.1 is a graph of the normalized error generated by a BP network when the learning rate was held constant, and at a high value (0.8). In it you can see that the normalized error tends to decrease in an orderly fashion until the network takes a bad jump. When the network takes a bad jump, the normalized error soars, and then relatively quickly falls back down, making the graph look spiky. If ALRA I is used, then the learning rate is reduced whenever there is an increase in the normalized error. So, there would be a reduction of the learning rate at the top of each spike. This will not lead to good convergence in a BP network. In Figure 5.2.1, the learning rate is held constant, so, when there is a spike, the network can relatively quickly reduce the normalized error down to a level near the normalized error before the spike. This is because it was the weight vectors' relationships to one another that brought the normalized error to where it was before the spike. The change in the weights that caused the spike was not a large change, but the effect of that change on the normalized error was large. Since the relationship between the weight vectors was good and has not changed very much, and the learning rate has not changed, it does not take the network very long to return to a normalized error close to what it was before the spike. Since ALRA I would reduce the learning rate at the top of each spike, then the network would have a hard time getting its normalized error back to its pre-spike levels.

In order to solve this problem with BP networks, ALRA I had to be modified. The main difference between ALRA I and ALRA II is that when ALRA II detects a rising error trend it must return the network to the state it was in before the rising error trend

began. Then it will reduce the learning rate, resulting in a smaller jump. In effect, the network has realized that it made a mistake, and will go back to try the same jump again, but with a slightly different target in mind. ALRA II will also increase the learning rate when the BP network's normalized error stops decreasing and remains the same (indicating it has reached a flat spot on the MSE surface). The problem that the BP network was applied to did not require the use of the moving target algorithm for reducing the learning rate. However, this does not mean that the moving target algorithm cannot be used with ALRA II and BP networks.

The BP network with ALRA II was trained to solve a simple OCR (optical character recognition) simulation problem. Several 5x7 bitmaps were created for characters of the alphabet (see Figure

5.2.2). The BP network was trained to recognize the characters and to output an 8-bit ASCII code for the character.

The BP code was taken from Maureen Caudill's *Neural Networks Primer Part*

III [27]. Caudill's BP code did not track the normalized error as the RBF

network used for the ALRA I example did. Instead it tracked the RMS (Root Mean Squared) error. Since the normalized error is based on the RMS error (see Equation 5.1.2), then the RMS error can be used in place of the normalized error in ALRA II.

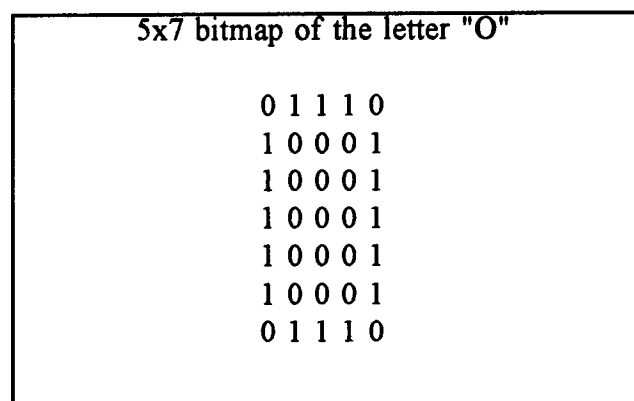


Figure 5.2.2: An OCR bitmap.

ALRA II begins in much the same manner as ALRA I. First ΔErr (error delta - the amount of change in the RMS error from one epoch to the next) is calculated at the end of each epoch. If there is a rising error trend (detected by the same mechanism in ALRA I) then the weights must be restored and the learning rate modified downward. If there is no rising error trend then the weights must be saved, and the descent of the RMS error must be checked to see if the network is in a flat spot of the MSE surface. To check if the network is within a flat spot a slow descent detector was used. The slow descent detector was a mechanism similar to the rising error trend detector in ALRA I. A network will activate the slow descent detector if ΔErr has decreased by only a small amount (less than 1%) from the previous epoch, for several consecutive epochs. For the OCR network the slow descent detector trigger was set to 3 epochs. So if ΔErr decreased by less than 1% for three consecutive epochs the learning rate would be increased. This increase would allow the network to take bigger steps when on a flat spot of the MSE surface. The learning rate is modified in either the positive or negative direction by 3%.

CHAPTER VI

Results and Conclusion

Chapter VI presents a description of how the programs used in this research were developed. It also presents the results of applying ALRA to the neural networks and problems described in the previous chapters.

6.1 Computer Systems Overview

To test the usefulness of ALRA I, the chaotic time series prediction problem was used. This problem was described in [7]. ALRA II was tested on an OCR simulation that was described in [2]. IBM 386 and 486 PC compatible computers were used for all programming and testing. The RBF network was programmed in Borland C++ 3.1 for the Microsoft Windows 3.1 environment. The BP network code was taken from Caudill's examples in [2] and modified. The BP network code was in ANSI C and was compiled with IBM's C Set/2 for the OS/2 2.0 operating system. When research was started on the RBF network, OS/2 2.0 had not yet been shipped (hence the choice of Windows 3.1 as a multitasking environment). OS/2 was used for its multitasking abilities and because C Set/2 would generate 32-bit code that would execute faster than the 16-bit code generated for the MS Windows environment. Since different operating systems and processors were used during the testing phase there will be no references to the amount of CPU time it took to train the neural networks. Instead, the measure used to judge how well each algorithm works is the number of passes through a training data set (epochs) required to reach a particular normalized error level.

6.2 Radial Basis Function Network Results

A description of the chaotic time series problem by Moody and Darken is as follows:

As it is usually formulated, this problem requires finding a real-valued mapping $f: R^n \rightarrow R$ which takes a sequence of n recent samples of a time series and predicts the value of the time series at a future moment. It is assumed that the underlying process which generates the time series is unknown. ... This particular time series results from integrating the Mackey-Glass differential delay equation:

$$\frac{dx[t]}{dt} = -bx[t] + a \frac{x[t-\tau]}{1+x[t-\tau]^{10}} \quad (6.2.1)$$

... The characteristic time of the series, given by the inverse of the mean of the power spectrum, is $t_{char} \approx 50$ The network has four real valued inputs ($x[t]$, $x[t - \Delta]$, $x[t - 2\Delta]$, $x[t - 3\Delta]$) and one real-valued output $x[t + T]$ with $\Delta = 6$ and $T = 85 > t_{char}$ [7].

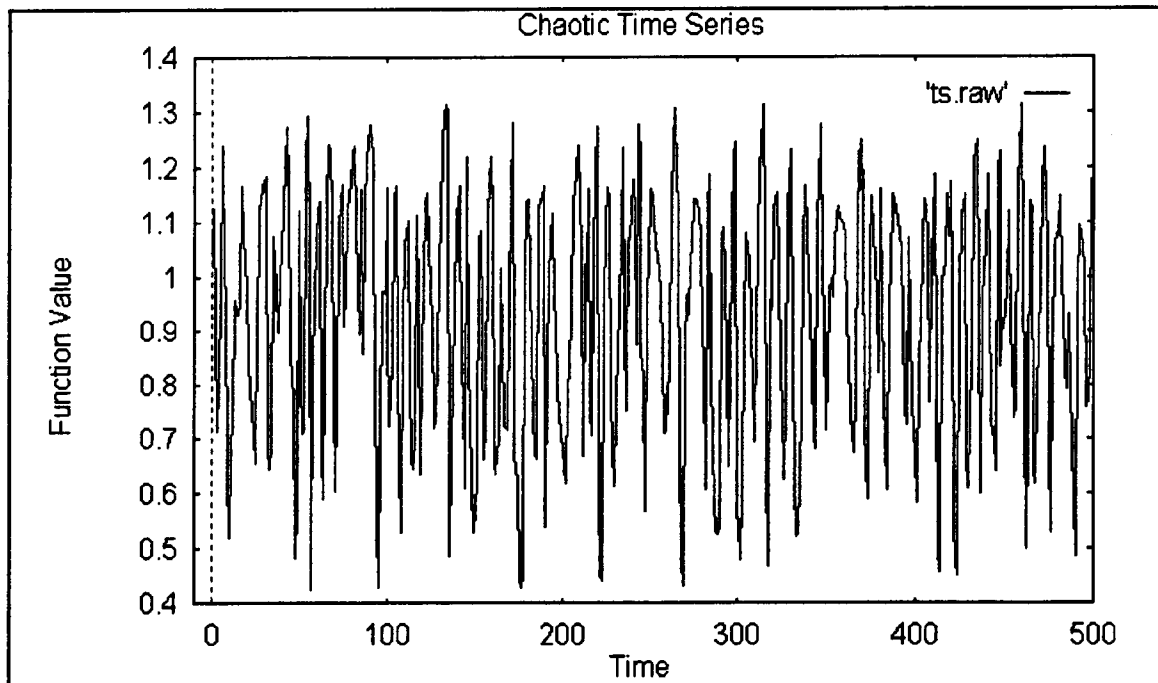


Figure 6.2.1: Mackey-Glass chaotic time series, 500 steps with delay $\tau = 17$.

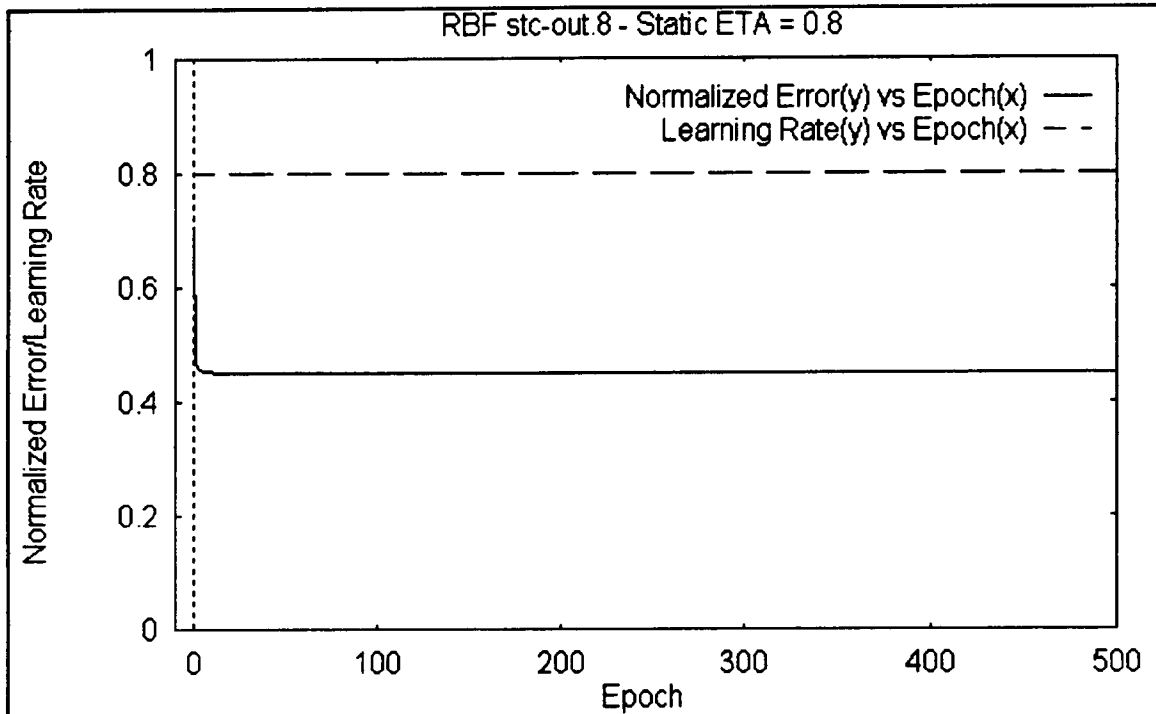


Figure 6.2.2: A RBF network's normalized error vs. epoch for a static learning rate of 0.8.

For this research, there were 500 training examples in the data set used for this problem. The RBF network had a middle layer consisting of 50 gaussian units. The gaussian centers were positioned by the merge clustering algorithm described in chapter 3.1. The width of the gaussians was determined by the nearest neighbor technique to be 0.14. Figure 6.2.2 is a graph of the normalized error decreasing through several training epochs. In this graph the learning rate is held constant at $\eta = 0.8$. This is an example of a bad guess for the learning rate when using a standard RBF network. Since the learning rate is so large the RBF network converges to a normalized error of about 0.47. In Figure 6.2.3 the value of the static learning rate is $\eta = 0.3$. This is a better choice, it allows convergence to a normalized error of about 0.22, substantially reducing the normalized error over the learning rate of 0.8.

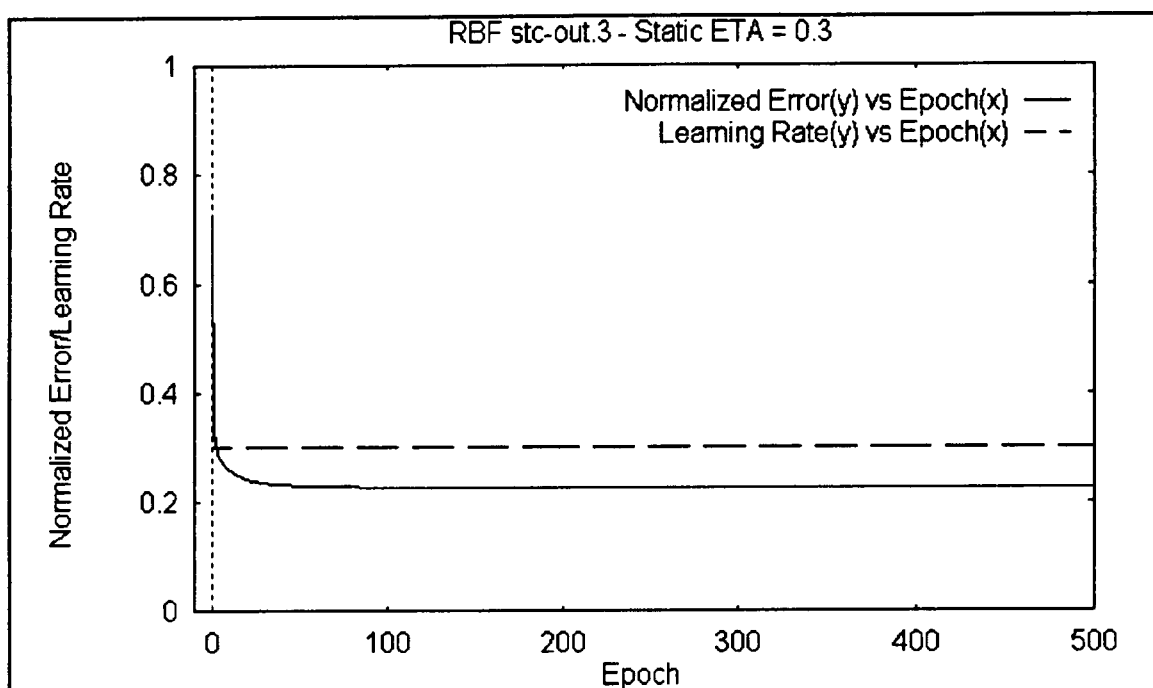


Figure 6.2.3: A RBF network's normalized error vs. epoch for a static learning rate of 0.3.

Figure 6.2.4 is an example of a RBF network that takes a long time to train due to a learning rate that is too small. Notice that the curve of the normalized error plot is shallower in this graph. This is because it takes the network more runs through the training data set to reach a normalized error level comparable to that in Figure 6.2.3.

Figure 6.2.5 shows an example of the same network configuration. The difference is that ALRA I is applied to the learning algorithm. This graph depicts how ALRA I will make this network perform when it is using the linear decay method (see Section 5.1) for reducing the learning rate. Notice that it takes longer for the normalized error to fall below 0.2 (about 450 epochs). At 750 epochs the normalized error was about 0.185521. After 6 runs of the network with the following static learning rates: 0.0005, 0.07, 0.15,

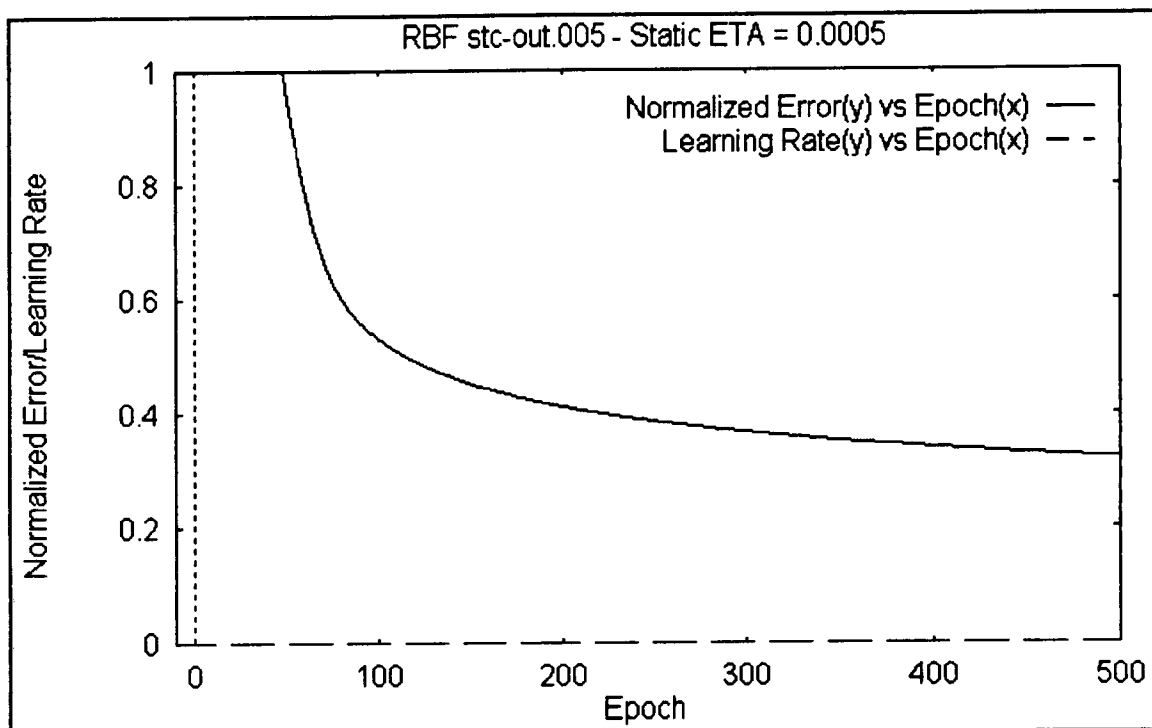


Figure 6.2.4: A RBF network's normalized error vs. epoch with a static learning rate of 0.0005.

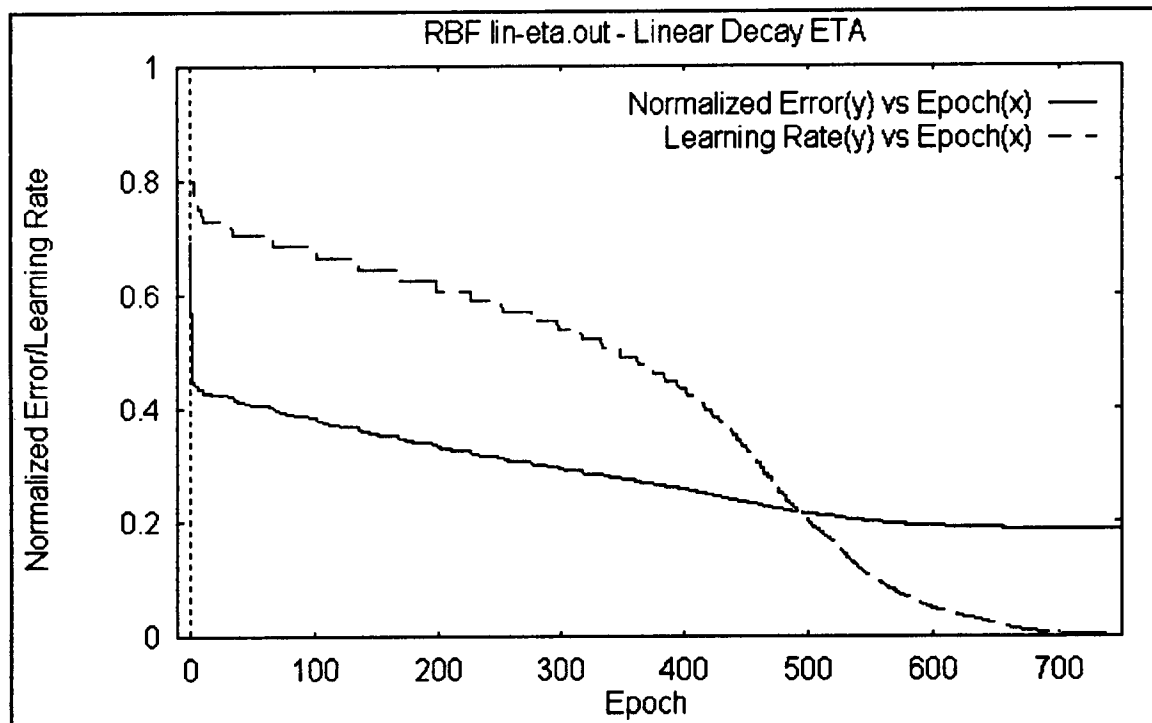


Figure 6.2.5: A RBF network's normalized error vs. epoch when using the linear decay learning rate.

0.3, 0.5 and 0.8. The best normalized error that was achieved was about 0.199, with $\eta = 0.07$. With the learning rate held at 0.0005 the network would probably converge to normalized error of less than 0.185, but it would have taken much longer than 750 epochs (see Figure 6.2.4).

Figure 6.2.6 shows how the same network configuration performs with ALRA I using the moving target mechanism for learning rate reduction. The normalized error was 0.19027 at 500 epochs. Using this method for adjusting the learning rate offers speed of convergence that is comparable to the static learning rate networks, and at the same time offers the convenience of the linear decay eta networks by not requiring trial and error in selecting a good learning rate. At 750 epochs the linear decay network was able to converge to a smaller normalized error, 0.185521. The moving target algorithm, however,

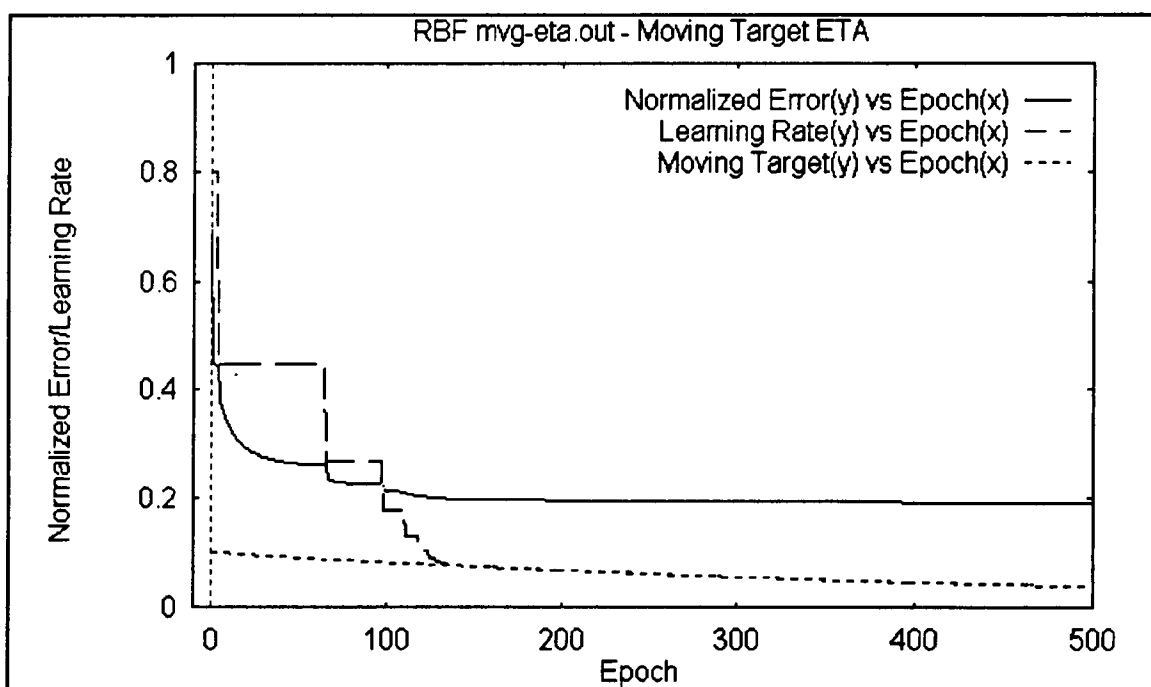


Figure 6.2.6: A RBF network's normalized error vs. epoch when using the moving target algorithm.

allowed the network to converge to a normalized error of about 0.194 in only 200 epochs. Conversely, the linear decay network had a normalized error that was greater than 0.2 at 500 epochs.

The important difference between the two learning rate modification algorithms is that the moving target network's normalized error behaves very similarly to the normalized error of the static learning rate networks. It decreases very rapidly at first and approaches its point of convergence asymptotically in a relatively short period of time. The linear decay network takes a leisurely approach to its point of convergence but can result in a smaller normalized error if given enough time. This is because the method that is used to reduce the learning rate is not as radical as the moving target method. With the moving target algorithm the network's speed of convergence is significantly reduced once the difference between the learning rate and its moving target becomes insignificant.

6.3 Backpropagation Results

The BP network was trained on the OCR simulation data described in Section 5.2. There were 10 character bitmaps, see Figure 5.2.2 for an example. The BP network consisted of 3 layers. The input layer was a 5x7 matrix through which the whole bitmap would be passed. The middle layer had 4 nodes, and the output layer had 8 nodes, one for each bit of the ASCII character to be output.

Figure 5.2.1 shows a run of this BP network while training on the OCR data, with the learning rate held at 0.8. The RMS error fluctuates rather erratically because the learning rate is too large. There are several instances where the RMS error is reduced to

a value less than 1, but the network is not able to continue converging from these good points since it eventually makes a bad jump to some far off place on the MSE surface. This is what prompted the idea behind ALRA II. The concept was to make the network detect when it made a bad jump, and to then try to correct its learning rate, in the hopes of making a better jump from the same position. Figure 6.3.1 illustrates how this BP network converges when a good value for the static learning rate is chosen, in this instance 0.3. The RMS error descends gracefully during the bulk of the training on the OCR data. Around epoch 230 the network makes a bad jump but recovers rather quickly and continues converging.

Figure 6.3.2 shows the RMS error as it decreases when ALRA II is applied to this BP network. For every large increase in the RMS error there is a corresponding decrease in the learning rate. Examine the graph at epochs 35 and 60 to see ALRA II adapting the

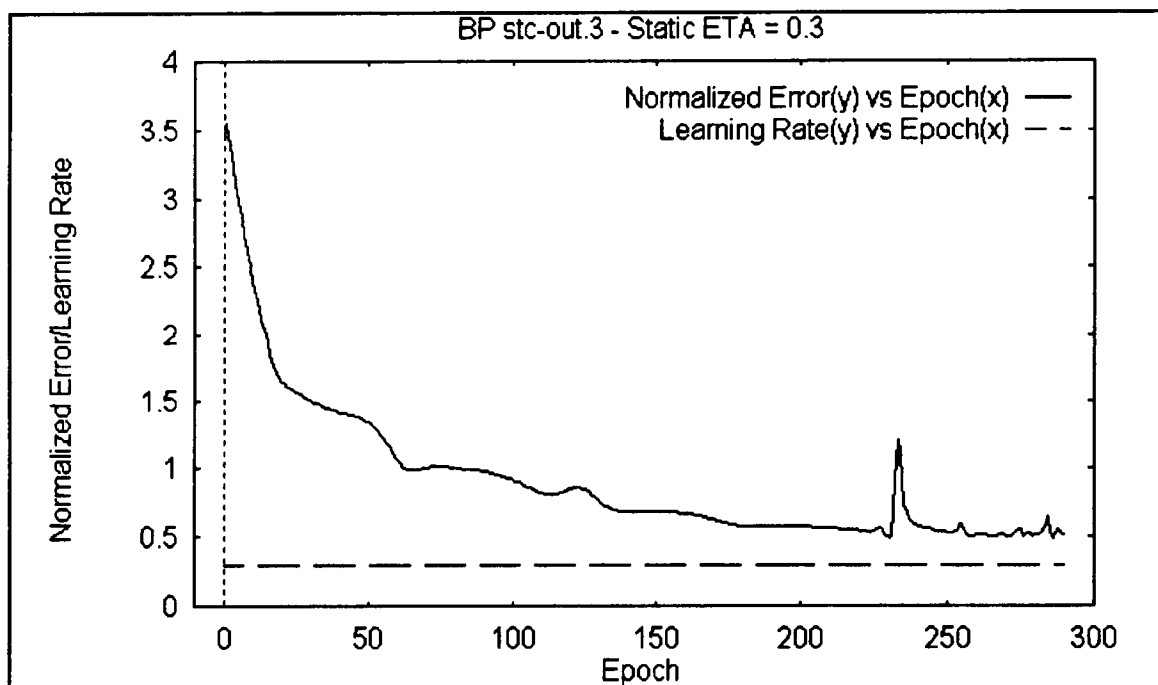


Figure 6.3.1: Normalized error for a BP network with a static learning rate of 0.3.

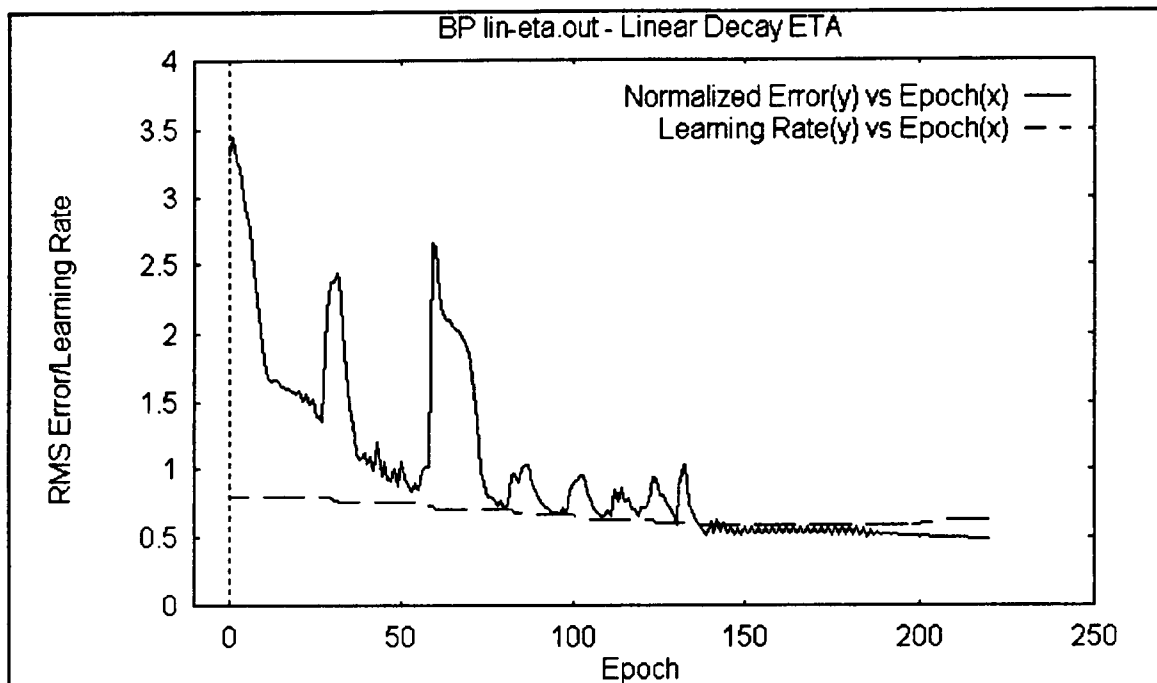


Figure 6.3.2: Normalized error for BP network using the linear decay algorithm.

learning rate. This graph also shows how ALRA II is able to detect when the network has reached a flat spot on the MSE surface. Around epoch 190 the RMS stops oscillating and flattens into a very gradual descent. ALRA II detects this and raises the learning rate in order to speed the network's convergence. At epoch 200 the learning rate is increased and the RMS error continues downward.

6.4 Conclusion

Neural networks that use gradient descent learning techniques require the user to determine a good learning rate for each particular network configuration and its associated problem. Determining a good learning rate usually involves training the network several times with various learning rates, keeping track of which ones offer the best combination of accurate results and reasonable training time. ALRA I and ALRA II were developed

to reduce the amount of trial and error guesswork the user must deal with when training her network. ALRA I offers the user two methods for modifying the learning rate as the radial basis function network is being trained. The linear decay method allows the network to train to a lower normalized error over a longer period of time. The moving target algorithm allows the network to converge to a higher normalized error, but over a shorter period of time. ALRA II offers an adaptive learning rate option to the backpropagation network user, and faster convergence than using the GDR alone. An area of further study is to assign a different learning rate to each weight in the network and use ALRA to modify each one individually. Another area of study is to employ the moving target mechanism in ALRA II for backpropagation networks.

LIST OF REFERENCES

List of References

- [1] Alianna J. Maren, Craig T. Harston and Robert M. Pap. *Handbook of Neural Computing Applications*, San Diego, Academic Press Inc., 1990.
- [2] Maureen Caudill. "Neural Networks Primer Part II", *AI Expert*, pages 55-61, Feb. 1988.
- [3] Kevin J. Lang and Michael J. Witbrock. "Learning to tell two spirals apart", *Proceedings of the 1988 Connectionist Models Summer School*, pages 56-69, San Mateo, CA, Morgan Kaufmann, 1989.
- [4] Ralf Salomon, "Improved convergence rate of back-propagation with dynamic adaptation of the learning rate", *Parallel Problem Solving from Nature*, pages 269-273, Berlin, Springer-Verlag,, 1990.
- [5] B. Widrow and M. Hoff. "Adaptive Switching Circuits", *1960 WESCON Convention Record: Part 4*, pages 96-104, 1960.
- [6] D. S. Broomhead and D. Lowe. "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems*, **2**, pages 321-325, 1988.
- [7] John Moody and Christian J. Darken. "Fast Learning in Networks of Locally-Tuned Processing Units", *Neural Computation*, **1**, pages 281-294, 1989.
- [8] M. T. Musawi, W. Ahmed, K. H. Chan, K. B. Faris and D. M. Hummels. "On the Training of Radial Basis Function Classifiers", *Neural Networks*, **5**, pages 595-603, 1992.
- [9] I. Guyon, V.N. Vapnik, B. E. Boser, L. Bottu and S. A. Solla. "Structural Risk Minimization for Character Recognition", *Advances in Neural Information Processing Systems Part 4*, San Mateo, CA, Morgan Kaufmann, 1992.
- [10] Leon Bottou and Vladimir Vapnik. "Local Learning Algorithms", *Neural Computation*, **4**, pages 888-900, 1992.
- [11] S. P. Lloyd. "Least squares quantization in PCM", *IEEE Transactions on Information Theory*, **IT-28:2**, (Bell Laboratories Internal Technical Report, 1957), 1982.

- [12] J. MacQueen. "Some methods for Classification and analysis of multivariate observations", *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probability*, page 281, Berkeley, U. California Press, 1967.
- [13] T. Kohonen. *Self-organization and Associative Memory*, Berlin, Springer-Verlag, 1988.
- [14] Steve Renals and Richard Rohwer. "Phoneme Classification Experiments Using Radial Basis Functions", *International Joint Conference on Neural Networks*, 1, pages 461-467, New York, IEEE TAB Neural Network Committee, 1989
- [15] M. Minsky and S. Papert. *Perceptrons*, Cambridge, MIT Press, 1969.
- [16] S-I Amari. "A theory of adaptive pattern classifiers", *IEEE Transactions on Electronic Computers*, EC-16, pages 299-307, 1967.
- [17] A. Bryson and Y.C. Ho. *Applied Optimal Control*, New York, Blaisdell, 1969.
- [18] P. Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", Ph.D. Dissertation, Harvard University, 1974.
- [19] D. Parker. "Learning Logic, Invention Report", S81-64, Office of Technology Licensing, Stanford University, 1982.
- [20] D. Rumelhart, G. Hinton, and R. Williams. "Learning Representations by Backpropagating Errors", *Nature*, **323**, pages 533-536, 1986.
- [21] Patrick Simpson. *Artificial Neural Systems*, New York, Pergamon Press, 1990.
- [22] Robert A. Jacobs. "Increased Rates of Convergence Through Learning Rate Adaption", *Neural Networks*, 1, pages 295-307, 1988.
- [23] Mark A. Gluck, Paul T. Glauthier, and Richard S. Sutton. "Adaption of Cue-Specific Learning Rates in Network Models of Human Category Learning", *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society : July 29 to August 1, 1992*, pages 540-545, Hillsdale, Lawrence Erlbaum Assoc., 1992.

- [24] Jayathi Janakiraman and Vasant Honavar. "Adaptive Learning Rate for Increasing Learning Speed in Backpropagation Networks", Department of Computer Science, Iowa State University, 1993.
- [25] Scott E. Fahlman. "Faster-Learning Variations on Back-Propagation: An Empirical Study", *Proceedings of the 1988 Connectionist Models Summer School Part 1*, pages 38-51, San Mateo, CA, Morgan Kaufmann, 1988.
- [26] John P. Cater. "Successfully Using Peak Learning Rates of 10 (and greater) in Back-propagation Networks with the Heuristic Learning Algorithm", *IEEE First International Conference on Neural Networks*, 2, pages 645-651, San Diego, CA, SOS Printing, 1987.
- [27] Maureen Caudill. "Neural Networks Primer Part III", *AI Expert*, pages 53-59, Jun. 1988.

VITA

Derek Michael Ealy was born on June 14, 1963 in Los Angeles, California. He graduated from Providence High School, Burbank, California in 1981. He received his Bachelor of Science degree in Computer Science from California State University, Dominguez Hills in 1991. In August of 1991 he entered the University of Tennessee Space Institute to study neural networks. He received his Masters of Science degree from UTSI in December of 1993. His interests also include artificial intelligence and object-oriented programming.