Department of Electrical Engineering
College of Engineering and Applied Sciences
State University of New York at Stony Brook
Stony Brook, New York 11794-2350

A Comparative Study of Basic Backpropagation and Backpropagation Through Time Algorithms

by

K. Wendy Tang and Hung-Jung Chen

Technical Report # 700

November, 1994

A Comparative Study of Basic Backpropagation and Backpropagation Through Time Algorithms

K. Wendy Tang and Hung-Jung Chen

Department of Electrical Engineering
SUNY at Stony Brook, Stony Brook, NY 11794-2350.

ABSTRACT

A neural network's ability to utilize large amounts of sensory information and its parallel processing, learning and generalization capabilities have made it an attractive computation model for problems intractable on traditional computing. In this report, we review two neural network schemes, the basic backpropagation, and the backpropagation through time algorithms. We incorporated these schemes in a two hidden layers neural network to learn the sine function. There are ten hidden nodes in each layer one input, and one output nodes. We found that, in general, the backpropagation through time algorithm is more robust and has a faster convergence rate. However, with a constant learning rate, the steepest decent method is, indeed, slow. To improve the rate of convergence, we adopted the Delta-Bar-Delta rule on the two algorithms. Our result indicates that the backpropagation through time algorithm has a superior performance than its basic counterpart. [1]

1 Introduction

Neural computing or *neural networks* have received much attention and have become an increasingly important computational model in recent years. Inspired by the structure of the brain, these networks are *massively parallel systems* that rely on dense interconnection of simple processing elements, or neurons. Each neuron has n inputs either from an external source or from other neurons. The weighted sum of these n values is transmitted to a non-linear function, the *transfer function*, to produce the output of a neuron. A diagram of the mathematical model of a neuron is shown in Figure 1 [1].

Neural networks offer several advantages over conventional computing architectures [2]. Calculations are carried out in parallel yielding speed advantages and programming is done

[1]The authors acknowledge and appreciate discussions with and contributions from Paul Werbos. This research was supported by the National Science Foundation under Grant No. ECS-9407363.

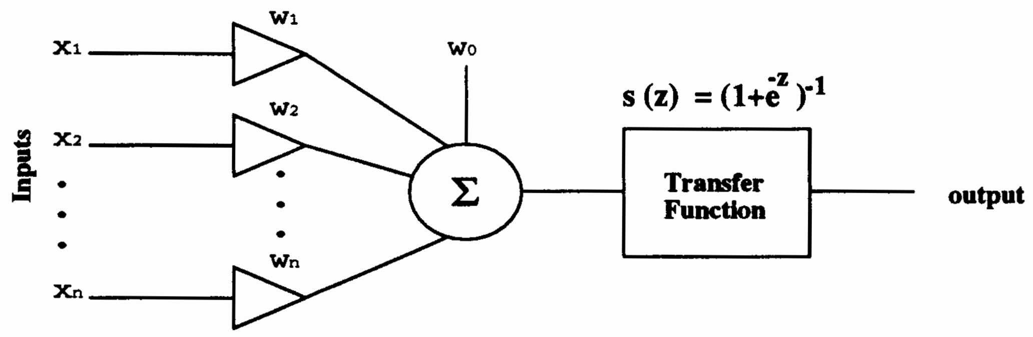

Figure 1: *A Mathematical Model of a Neuron.*

by training through examples. These networks are characterized by their *learning* and *generalization* capabilities and can be deployed as "black boxes" that map inputs to outputs with no explicit rules or analytic function [3]. The neural network "learns" the system model by training through a set of desired input-output patterns. They are particularly useful for problems intractable with traditional computational method, such as pattern recognition and classification, prediction, diagnosis, and process control. Specific applications include speech synthesis and recognition, image processing and analysis, sonar and seismic signal classification, and adaptive control [3].

The most dominant form of neural networks is the multi-layer backpropagation network (Figure 2) [4, 5, 6]. It is a hierarchical design consisting of fully interconnected layers of neurons [7]. The weights associated with each neuron are updated by taking the gradient of an error function with respect to the weights and performing a gradient search of the weight space [8]. Errors are propagated backwards through the network, hence the name ***back-propagation.*** Despite its popularity, the main drawback of the basic backpropagation algorithm is its slow convergence rate. Various efforts are made to increase the rate of convergence, e.g., the Delta-Bar-Delta Rule [9]. In this research, we focus on this multi-layer backpropagation network. We discuss and compare the basic algorithm with a more sophisticated version, the ***backpropagation through time*** algorithm proposed by Werbos [10]. We found that by combining the backpropagation through time algorithm with the Delta-Bar-Delta rule, the neural network provides more robust and faster learning.

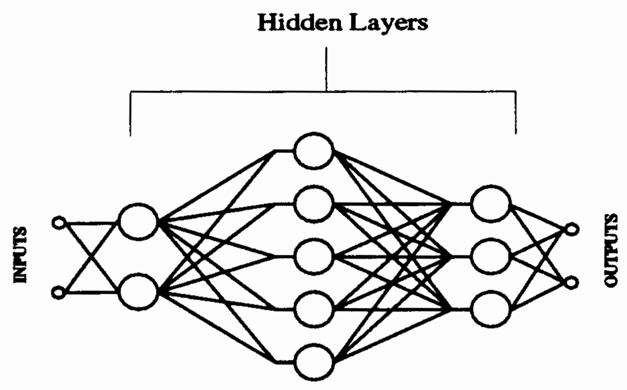

Figure 2: *A Multi-Layer Backpropagation Network.*

This technical report is organized as follows: in Section 2, we review the basic backpropagation algorithm. Section 3 is a summary of the backpropagation through time algorithm. The Delta-Bar-Delta rule are presented in Section 4. Various results and discussions are included in Section 5. Finally, in Section 6, we present a summary and conclusions.

2 Basic Backpropagation Algorithm

For expository convenience, we assume there are m inputs, n outputs, H hidden nodes, and T training samples. The training inputs are presented to the network as $X_i(t), \quad i = 1,\ldots,m, \quad t = 1,\ldots,T$ and the corresponding desired outputs are $Y_i(t), \quad i = 1,\ldots,n, \quad t = 1,\ldots,T$. The network equations are made up of the feedforward and feedback components. The detailed of the algorithm can be found in [10]. For the reader's convenience, they are also summarized here.

Feedforward Equations

$$\begin{array}{rcll} x_i(t) & = & X_i(t) & 1 \le i \le m \\ net_i(t) & = & \sum_{j=1}^{i-1} W_{ij} x_j(t) & m < i \le m+H+n \\ x_j(t) & = & s(net_j(t)) & m < j \le m+H+n \\ \hat{Y}_i(t) & = & x_{m+H+i}(t) & 1 \le i \le n \end{array} \qquad (1)$$

The error of the network is obtained by comparing the actual and the desired outputs.

$$E = \sum_{t=1}^{T} E(t) = \sum_{t=1}^{T}\sum_{i=1}^{n} 0.5[\hat{Y}_i(t) - Y_i(t)]^2 \tag{2}$$

where $\hat{Y}_i(t)$ is the output of the neural network and $Y_i(t)$ is the desired outputs. This error is feedback to the network. The error gradient F_W_{ij} with respect to each weight, W_{ij} is calculated with the feedback equations:

Feedback Equations

$$\begin{array}{lll}
F\_\hat{Y}_i(t) & = \frac{\partial E}{\partial \hat{Y}_i(t)} = \hat{Y}_i(t) - Y_i(t) & i = 1, \ldots, n \\
F\_x_i(t) & = F\_\hat{Y}_{i-m-H}(t) & \\
 & + \sum_{j=i+1}^{m+H+n} W_{ji} * F\_net_j(t) & i = m+H+n, \ldots, m+1 \\
F\_net_i(t) & = s'(net_i) * F\_x_i(t) & i = m+H+n, \ldots, m+1 \\
F\_W_{ij} & = \sum_{t=1}^{T} F\_net_i(t) * x_j(t) & i, j = 1, \ldots, m+H+n
\end{array} \tag{3}$$

where $s(z)$ is the sigmoidal transfer function and $s'(z)$ is the derivative of $s(z)$. Also,

$$s(z) = 1/(1+e^{-z})$$
$$s'(z) = s(z) * (1 - s(z))$$

Furthermore, we assume the network is multi-layer and there is no connections for nodes on the same layer. That is, $W_{ij} = 0$, if nodes i and j are in the same layer.

Once F_W_{ij} (the gradient of E with respect to W_{ij}) is calculated, each weight is updated according to:

$$\text{New } W_{ij} = W_{ij} - \alpha * F\_W_{ij} \quad i, j = 1, \ldots, m+H+n \tag{4}$$

where α is a constant called the *learning rate.*

3 Backpropagation Through Time Algorithm

Backpropagation through time was first proposed by Werbos [10]. It is basically an extension of the basic backpropagation algorithm but consider also memory from previous time periods. Mathematically, this is implemented through the introduction of a second set of weights W'.

In our version of the backpropagation through time algorithm, we associated a weight W' at each hidden and output node. A more general version that includes a W' for each connection can be found in [10]. In our version, the second equation of the feedforward equations (Eq 1) is replaced by:

$$net_i(t) = \sum_{j=1}^{i-1} W_{ij} x_j(t) + W_i' x_i(t-1), \quad m < i \leq m + H + n \tag{5}$$

And the second equation of the feedback equations (Equation 3) is replaced by:

$$\begin{aligned} F\_x_i(t) = F\_\hat{Y}_{i-m-H}(t) + \textstyle\sum_{j=i+1}^{m+H+n} W_{ji} * F\_net_j(t) \quad &+ W_i' * F\_net_i(t+1) \\ &i = m+H+n, \ldots, m+1 \end{aligned} \tag{6}$$

For adaptation of the W',

$$\begin{array}{ll} F\_W_i' & = \sum_{t=1}^{T} F\_net_i(t) * x_i(t) \\ \text{New } W_i' & = W_i' - \beta * F\_W_i' \qquad\qquad i = m+1, \ldots, m+H+n \end{array} \tag{7}$$

where β is the constant learning rate for W'.

4 Delta-Bar-Delta Rule

To improve the convergence speed of the steepest decent method, Jacob proposed the delta-bar-delta algorithm [9]. Basically, the algorithm is a special case of the *Adaptive Learning Rate* (ALR) discussed in [11]. It implements four heuristics:

1. every connection has its own individual learning rate;

2. every learning rate varies over time;

3. when the gradient possesses the same sign for several consecutive time steps, increase the learning rate;

4. when the sign of the gradient alternates for several consecutive time steps, decrease the learning rate.

In other words, every weight of the network is given its own learning rate and that the rates changes with time. According to [9], the learning rate update rule is:

$$\Delta\alpha_{ij}(t) = \begin{cases} \kappa & \text{if } \bar{\delta}_{ij}(t-1)\delta_{ij}(t) > 0 \\ -\phi\alpha_{ij}(t-1) & \text{if } \bar{\delta}_{ij}(t-1)\delta_{ij}(t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where

$$\begin{aligned} \delta_{ij}(t) &= F\_W_{ij} \\ \bar{\delta}_{ij}(t) &= (1-\theta)\delta_{ij}(t) + \theta\bar{\delta}_{ij}(t-1) \\ \alpha_{ij}(t) &= \alpha_{ij}(t-1) + \Delta\alpha_{ij}(t) \end{aligned}$$

In these equations, $\delta_{ij}(t)$ is the partial derivative of the error with respect to W_{ij} at time t and $\bar{\delta}_{ij}(t)$ is an exponential average of the current and past derivatives with θ as the base and time as the exponent[9]. If the current derivative of a weight and the exponential average of the weight's previous derivatives possess the same sign, the learning rate for that weight is incremented by a constant κ. If the current derivative of a weight and the exponential average of the weight's previous derivatives possess opposite signs, the learning rate for the weight is decremented by a proportion ϕ of its current value [9].

As will be discussed in the next section, we found the best result comes from a combination of the backpropagation through time algorithm with the delta-bar-delta rule. In this case, $\beta_i(t)$, the learning rate for W_i' also changes with time. More specifically,

$$\Delta\beta_i(t) = \begin{cases} \kappa & \text{if } \bar{\gamma}_i(t-1)\gamma_i(t) > 0 \\ -\phi\beta_i(t-1) & \text{if } \bar{\gamma}_i(t-1)\gamma_i(t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

where

$$\begin{aligned} \gamma_i(t) &= F\_W_i' \\ \bar{\gamma}_i(t) &= (1-\theta)\gamma_i(t) + \theta\bar{\gamma}_i(t-1) \\ \beta_i(t) &= \beta_i(t-1) + \Delta\beta_i(t) \end{aligned}$$

5 Comparison of Results

To compare the performance of the basic and backpropagation through time algorithms, we use a neural network with two hidden layers and ten nodes in each layer to learn the sine function for 100 sampling points over a period of 2π. Figure 3 shows the comparison of the basic backpropagation (labeled as "Basic") and the backpropagation through time (labeled as "B-Time") algorithms. In this example, the learning rate $\alpha = 0.8$ for the two algorithms and different learning rates β for W' are used. In these cases, the initial weight W is randomly distributed between ± 0.1 and weight W' is fixed at zero until iteration 1000.

Although from Figure 3, we can clearly observe that the algorithms are converging, the learning rate is very slow and the error after $10,000$ iterations is still large. In an effort to increase the learning rate and decrease the error, we have made various attempts. First, we use some "smart" value as the initial weights. From these results, we concluded that the learning rates for the weights need to be adjusted. We then investigate the effect of delta-bar-delta rule. We found that if the constant, κ is adjusted according to the error, the convergence is more stable. We call the resultant algorithm, the *modified* delta-bar-delta rule. Finally, we discovered that the best performance can be obtained by combining the modified delta-bar-delta rule with some "smart" value of initial weight. The following subsections summarize our findings. The parameters used in all the figures are included in Table 1.

5.1 Smart Initial Weights

One obvious approach is to start with an intelligent guess of the initial weight for gradient decent. Instead of using random weights between (± 0.1), we use the resultant weight of the basic backpropagation for learning 20 samples after 10,000 iterations as the initial weight for both the basic and backpropagation through time algorithms for learning 100 samples. For the W' weight in the backpropagation through time algorithm, we set $W' = 0.0$ and start learning of W' only after a certain number of iterations, $BacIter$. By delaying the learning of the W' weight, we effectively allow the network to achieve a stable state before the introduction of the backpropagation through time algorithm. This technique is also discussed in [10].

Figure 4 shows the result of using the two algorithms to learn 100 samples with $\alpha = 0.8$ and $\beta = 0.2$. The result is not encouraging. Both networks fluctuate over the 10,000 iterations. We attribute such fluctuation to the large learning rate. Our next attempt is

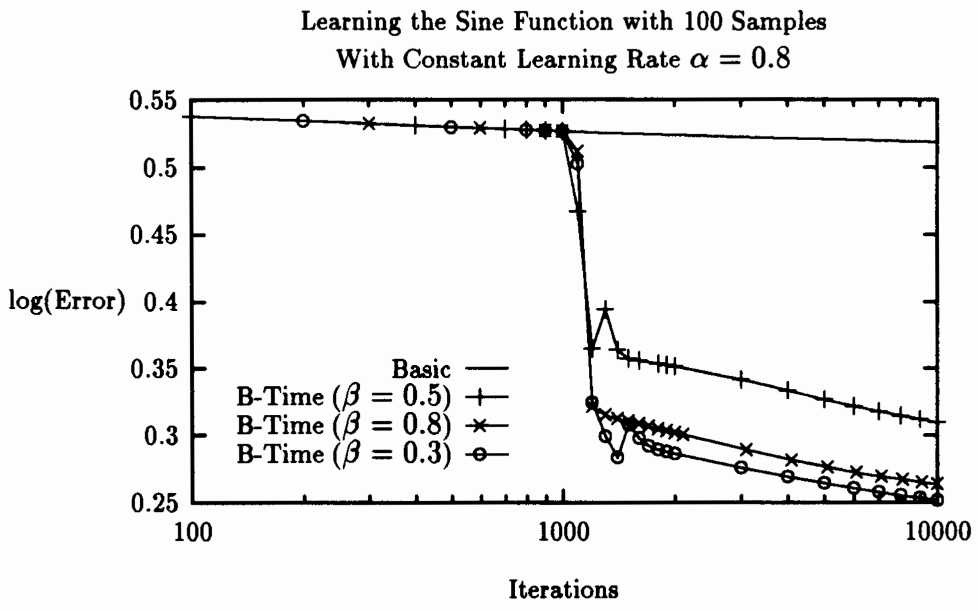

Figure 3: *Constant Learning Rate Comparison.*

to use a smaller learning rate, α. Figure 5 shows the result of learning 100 samples over $100,000$ iterations with $\alpha = 0.2$. The learning rate for W' in the backpropagation through-time algorithm is $\beta = 0.01, 0.03, 0.05$. From this figure, we observe that, indeed, the error is much smaller than those in Figure 3. Furthermore, in all cases, the backpropagation through time algorithm learns faster and has a smaller error than the basic algorithm and the convergence rate increases with β. But when β is large ($\beta = 0.05$), the error fluctuates again. This observation agrees with intuition. When the learning rate (or step size) for W' is big, the nature of the steepest decent method implies that adjustments of the weights are large and thus stability problem arises.

Hence from Figure 5, it is clear to us that for a given learning rate α, the backpropagation through time algorithm converges much faster than the basic algorithm. Nevertheless, the strength of the backpropagation through time algorithm can best be utilized if, somehow, the learning rate is adjusted. Therefore, we adapted Jacob's delta-bar-delta rule in weight adjustment.

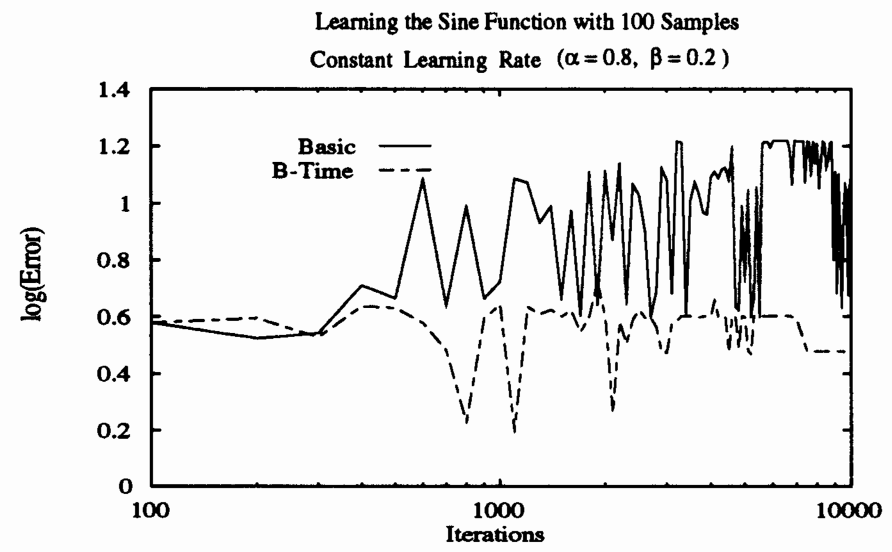

Figure 4: *Constant Learning Rate Comparison with Smart Initial Weights.*

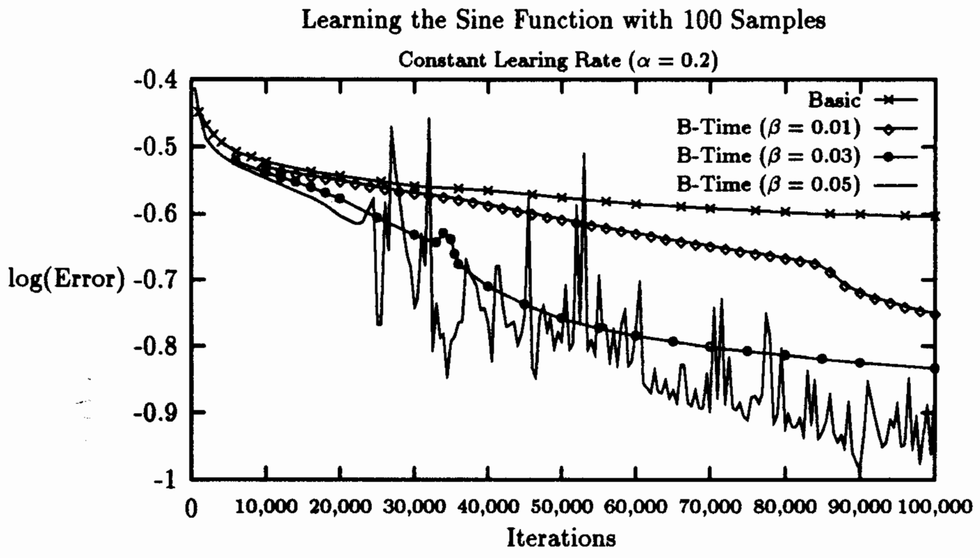

Figure 5: *Constant Learning Rate Comparison with Smart Initial Weights II.*

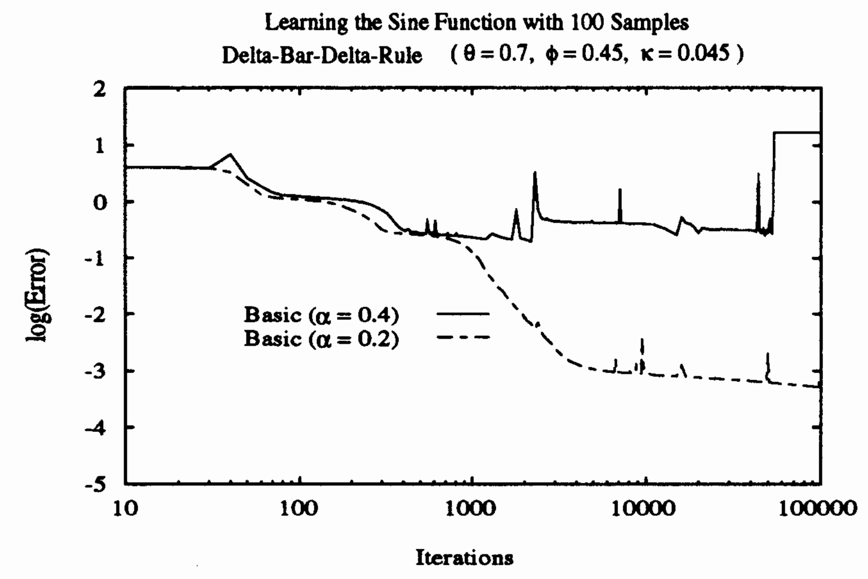

Figure 6: *Constant Delta-Bar-Delta Rule for Basic Backpropagation.*

5.2 Delta-Bar-Delta Rule

Figures 6 and 7 show the direct implementation of the delta-bar-delta rule with constants ($\theta = 0.7, \phi = 0.45, \kappa = 0.045$) to the basic backpropagation and backpropagation through time algorithms. We observed that the basic algorithm exhibits minor fluctuation for $\alpha = 0.2$ and more serious fluctuation for $\alpha = 0.4$. We believe that the κ value is too big. And when the initial α value is large ($\alpha = 0.4$), the error is less stable than that of $\alpha = 0.2$. The backpropagation through time algorithm, on the other hand, exhibits the most severe fluctuation. We attribute this behavior to the fact that both the W and W' learning rate are adjusted with the delta-bar-delta rule constants. When κ is large, both learning rates are increased too much and thus instability arises. To demonstrate this point, we implemented the delta-bar-delta rule for smaller κ values, $\kappa = 0.009, 0.0009, 0.00009$. Figures 8 and 9 show these results. Indeed, for small κ values ($\kappa = 9.0e-4, 9.0e-5$), stability is no longer a problem. A large κ value tends to give better performance (smaller error) in the early stage (say, $< 1,000$ iterations) but is more prone to stability problem in the later stage. Based on this observation, we proposed a *modified* delta-bar-delta rule, in which the κ value changes with different ranges of errors.

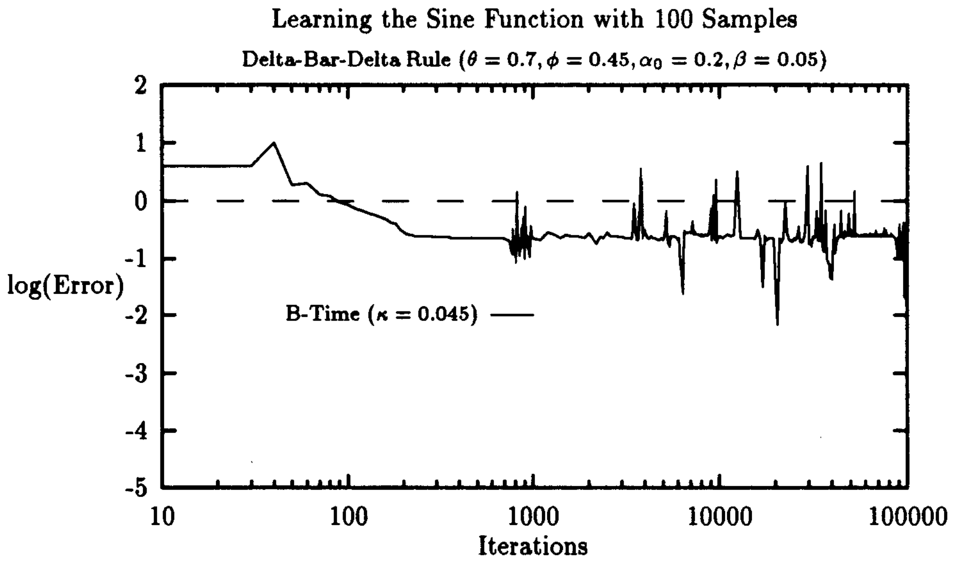

Figure 7: *Constant Delta-Bar-Delta Rule for Backpropagation Through Time.*

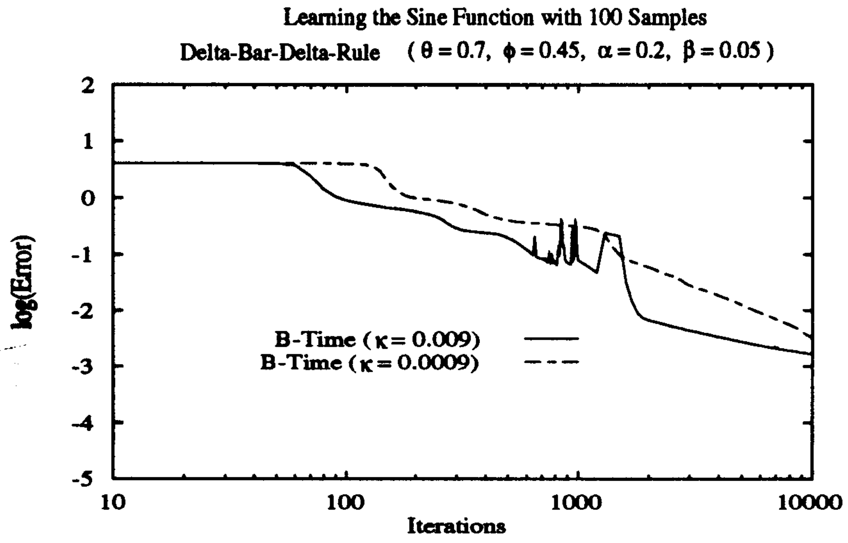

Figure 8: *Constant Delta-Bar-Delta Rule for Backpropagation Through Time.*

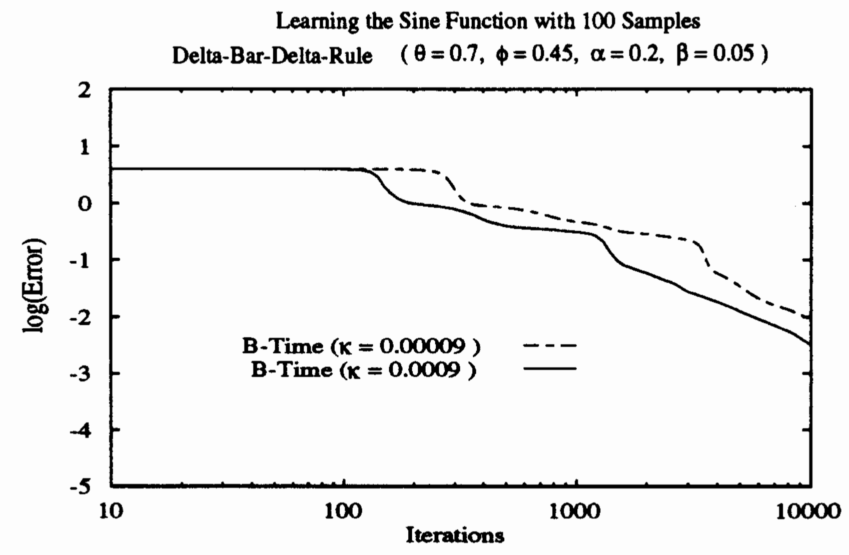

Figure 9: *Constant Delta-Bar-Delta Rule for Backpropagation Through Time II.*

5.3 Modified Delta-Bar-Delta Rule

From previous experimentation, we found that both algorithms converge faster if κ in the delta-bar-delta algorithm changes with the error. In this modified delta-bar-delta rule, we start with an initial κ value and when the error calculated as in Equation 2 is less than ϵ_i, then $\kappa = \kappa_i$, $i = 1, 2, 3, 4$. In some cases, we use $i = 1, 2, 3$ only.

Figure 10 shows the result of learning 100 samples using the basic algorithm. As compared with Figure 6, the modified delta-bar-delta rule provides a much more stable convergence. However, for $\alpha = 0.4$, convergence is still not stable when iteration $> 10,000$. This is due to the fact that the κ value used is still too large. From this figure, we conclude that the proper choice of the κ value is critical to the stability of the algorithm. The task of determining the most appropriate κ value becomes formidable when the initial error is large. We also noted that in this case, the initial weight are randomly assigned between ± 1. If we can use some smart guess of the initial weight, such as the resultant weight of 20 samples, to reduce the initial error, we believe that the algorithm will be less sensitive to the choice of the κ values. We shall explore this direction in the next section.

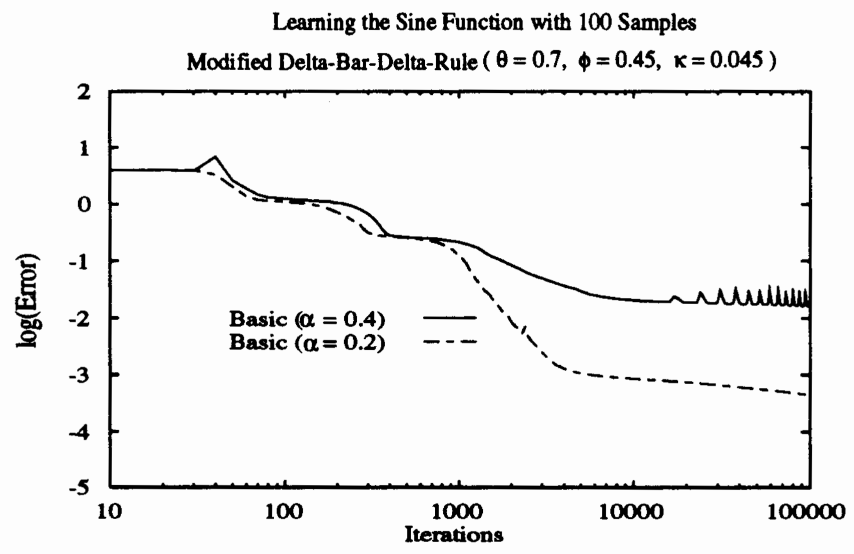

Figure 10: *Modified Delta-Bar-Delta Rule for Basic Backpropagation.*

Figure 11 shows the modified delta-bar-delta rule for backpropagation through time. In this case, we use three epsilon values: $\epsilon_1 = 1e-3, \epsilon_2 = 5e-4, \epsilon_3 = 4.75e-5$. That is, initially $\kappa = 0.045$ and when the error is $< \epsilon_i$, $\kappa = \kappa_i, i = 1,2,3$ ($\kappa_1 = 0.025, \kappa_2 = 0.009, \kappa_3 = 0.007$). Also, initially, $W' = 0.0$ and its adjustment starts at iteration $> BacIter = 1,000$. From the figure, we observe that immediately after 1,000 iteration, the error decreases significantly indicating that the backpropagation through time algorithm helps to reduce the error. However, the error later shots back up between iteration $1,000$ and $2,000$. During this period, the error is in the order of 10^{-2} which is $> \epsilon_1 = 10^{-3}$, therefore, $\kappa = 0.045$. We believe that this relatively large κ value is responsible for the increase in error. In another attempt, we used a different set of ϵ_i values. In particular, $\epsilon_1 = 1e-2, \epsilon_2 = 5e-3, \epsilon_3 = 1e-4, \epsilon_4 = 5e-4$. The κ values remained unchanged, except with the addition of $\kappa_4 = 0.005$. With this new set of ϵ values, during iteration $1,000$ and $2,000$, the error is $< \epsilon_1 = 1e-2$, and thus the κ value is reduced to $\kappa_1 = 0.025$. Not surprisingly, once this smaller κ value is used, the algorithm converges stablely. However, we consider such a trial and error guess of the proper set of ϵ values not practical. In the next section, we shall explore the idea of coupling this modified delta-bar-delta rule scheme with some intelligent guess of the initial weight.

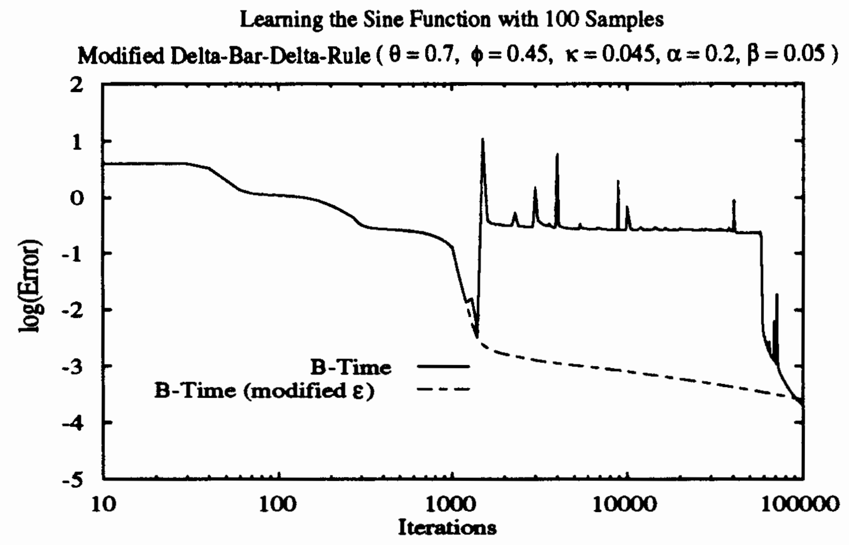

Figure 11: *Modified Delta-Bar-Delta Rule for Backpropagation Through Time.*

5.4 Modified Delta-Bar-Delta Rule with Smart Initial Weight

In this section, we use the resultant weight of learning 20 samples after 10,000 iterations as the initial weight W for learning 100 samples of the sine function. Again, the initial values for $W' = 0.0$ and their adjustment starts after iteration $> BacIter$. Figures 12-15 show these comparisons for learning 20 samples with different α, β values. In these cases, we start the learning of W' after $BacIter = 1,000$ iterations. The backpropagation through time algorithm performs consistently better than the basic algorithm. The resultant weight after learning the 20 samples for 10,000 iterations is used in the learning of 100 samples over a period of 2π. Figures 16 to 23 show results for different α, β values.

From these figures, we confirm that the backpropagation through time algorithm is more robust than its basic counterpart when the modified delta-bar-delta rule is used with some intelligent guess of the initial weights. For example, in Figure 16, the basic algorithm fluctuates within the first 10,000 iterations whereas the through-time algorithm exhibits only minor fluctuation and quickly converges to a smaller error. However, such fluctuation is not a characteristic of either algorithms. In fact, we attribute the fluctuation to individual weight-space. For example, in Figures 17 to 19 and Figures 21 to 23 both algorithms achieve stable convergence. We believe that since previous memory are used in the prediction of current outputs, the backpropagation through time algorithm is able to minimize any possible fluctuation and achieve a faster and more stable convergence. As another example, in Figure 20 both algorithms display fluctuation in earlier iterations, but the through-time algorithm is able to achieve steady convergence in less than $10,000$ iterations.

6 Conclusions

In this report, we reviewed the basic backpropagation, backpropagation through time algorithms, and the delta-bar-delta rule for faster convergence. For comparison purpose, we use the two algorithms to learn the sine function over the period of 2π in a two-hidden-layer neural network. The network consists of one input, one output, and ten hidden nodes in each layer. We found that, in general, the backpropagation through time algorithm is faster and more robust than its basic counterpart. To improve the rate of convergence, we explore different options, including the use of some smart initial weight and delta-bar-delta rule. We found that the best performance is achieved by combining a modified delta-bar-delta rule in which the κ value changes with error and the use of some intelligent guess of the weight space. We showed that by accounting for previous memory, the backpropagation through time algorithm is able to minimize any possible error fluctuation and provides a faster and more robust convergence.

References

[1] S.Y. Kung. *Digital Neural Networks.* Prentice Hall, Inc, Englewood Cliffs, New Jersey, 1993.

[2] Richard L. Lippmann. "An Introduction To Computing With Neural Nets". *IEEE ASSP Magazine*, pages 4–22, April 1987.

[3] Judith E. Dayhoff. *Neural Network Architectures.* Van Nostrand Reinhold, New York, 1990.

[4] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA, November 1974.

[5] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA, 1986.

[6] D. Parker. Learning Logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA, 1985.

[7] Paul J. Werbos. *The Roots of Backpropagation.* John Wiley and Sons, Inc, New York, 1994.

[8] B. Widrow and M.A. Lehr. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation". *Proceedings of the IEEE*, 78(9):1415–1442, September 1990.

[9] Robert A. Jacobs. "Increased Rates of Convergence Through Learning Rate Adaptation". *Neural Networks*, 1:295–307, 1988.

[10] Paul J. Werbos. "Backpropagation Through Time: What It Does and How to Do It". *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.

[11] Paul J. Werbos. Neurocontrol and Supervised Learning: an Overview and Evaluation. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligent Control*, pages 65–89. Van Nostrand Reinhold, 1992.

	$\theta = 0.7, \kappa_1 = 0.025, \kappa_2 = 0.009, \kappa_3 = 0.007, \kappa_4 = 0.005$										
	Figure Number	# of Samples	α_0	β_0	ϕ	κ	BacIter	ϵ_1	ϵ_2	ϵ_3	ϵ_4
Constant	Fig 3	100	0.8	$0.3, 0.5, 0.8$	-	-	1,000	-	-	-	-
Learning	Fig 4	100	0.8	0.2	-	-	100	-	-	-	-
Rate	Fig 5	100	0.2	$0.01, 0.03, 0.05$	-	-	1,000	-	-	-	-
Constant	Fig 6	100	$0.2, 0.4$	-	0.45	0.045	-	-	-	-	-
Delta	Fig 7	100	0.2	0.05	0.45	0.045	0	-	-	-	-
Bar	Fig 8	100	0.2	0.05	0.45	$9e-3, 9e-4$	0	-	-	-	-
Delta	Fig 9	100	0.2	0.05	0.45	$9e-4, 9e-5$	0	-	-	-	-
	Fig 10	100	0.2	-	0.45	0.045	-	$1e-3$	$5e-4$	$4.75e-5$	-
		100	0.4	-	0.45	0.045	-	1.0	0.5	$4.75e-2$	$5e-3$
Modified	Fig 11	100	0.2	0.05	0.45	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
Delta		100	0.2	0.05	0.45	0.045	1,000	$1e-2$	$5e-3$	$1e-3$	$5e-4$
Bar	Fig 12	20	0.2	0.1	0.45	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
Delta	Fig 13	20	0.4	0.2	0.45	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
	Fig 14	20	0.6	0.3	0.45	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
	Fig 15	20	0.8	0.4	0.45	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
	Fig 16	100	0.2	0.4	0.45	0.045	100	$1e-3$	$5e-4$	$4.75e-5$	-
Delta	Fig 17	100	0.4	0.2	0.45	0.045	100	$1e-3$	$5e-4$	$4.75e-5$	-
Bar	Fig 18	100	0.6	0.3	0.45	0.045	100	$1e-3$	$5e-4$	$4.75e-5$	-
Delta	Fig 19	100	0.8	0.4	0.45	0.045	100	$1e-3$	$5e-4$	$4.75e-5$	-
with	Fig 20	100	0.4	0.4	0.15	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
Smart	Fig 21	100	0.4	0.2	0.6	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
Weight	Fig 22	100	0.4	0.2	0.8	0.045	1,000	$1e-3$	$5e-4$	$4.75e-5$	-
	Fig 23	100	0.2	0.4	0.6	0.045	100	$1e-3$	$5e-4$	$4.75e-5$	-

Table 1: *A Summary of Simulation Parameters.*

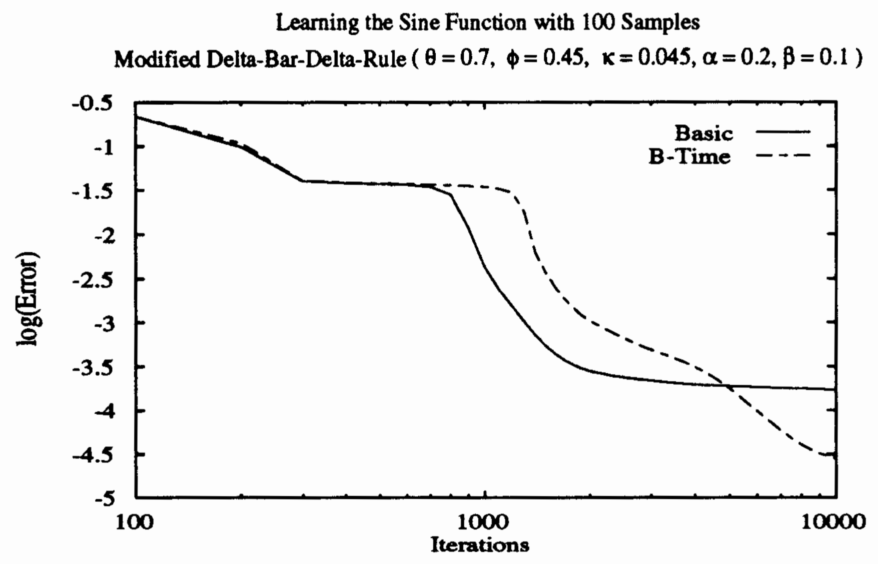

Figure 12: *Learing with Delta-Bar-Delta Rule (20 Samples, $\alpha = 0.2$).*

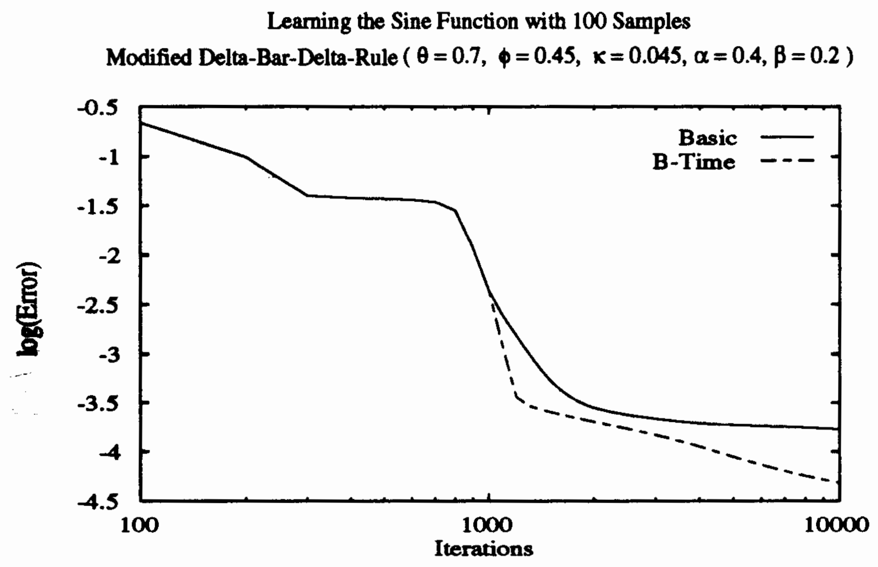

Figure 13: *Learing with Delta-Bar-Delta Rule (20 Samples, $\alpha = 0.4$).*

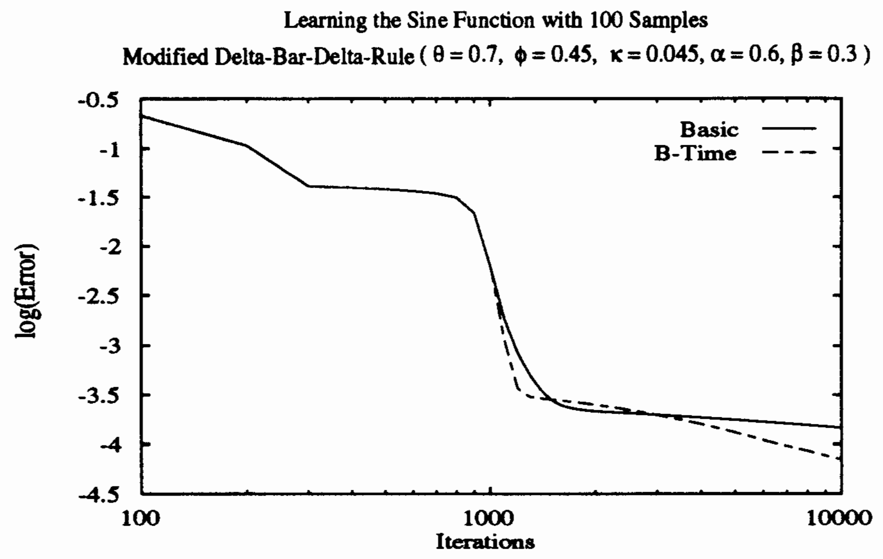

Figure 14: *Learing with Delta-Bar-Delta Rule (20 Samples, $\alpha = 0.6$).*

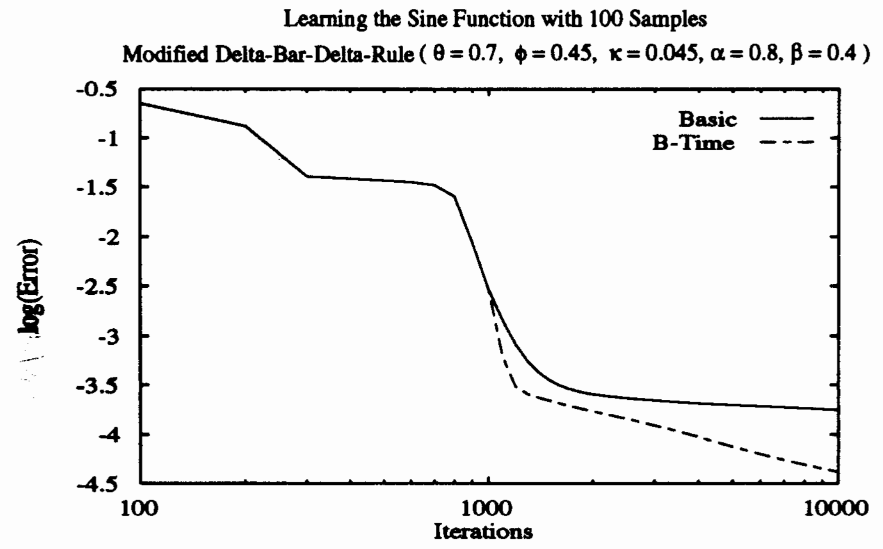

Figure 15: *Learing with Delta-Bar-Delta Rule (20 Samples, $\alpha = 0.8$).*

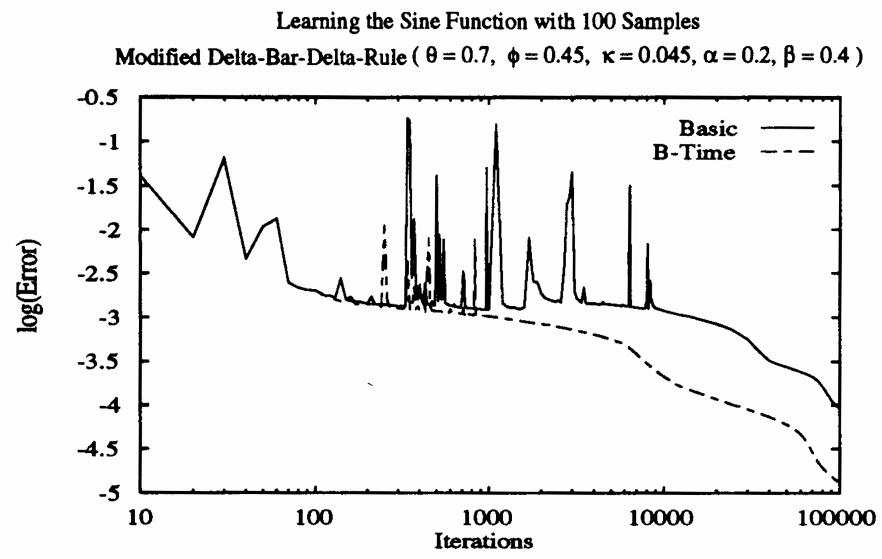

Figure 16: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.45, \alpha = 0.2$).

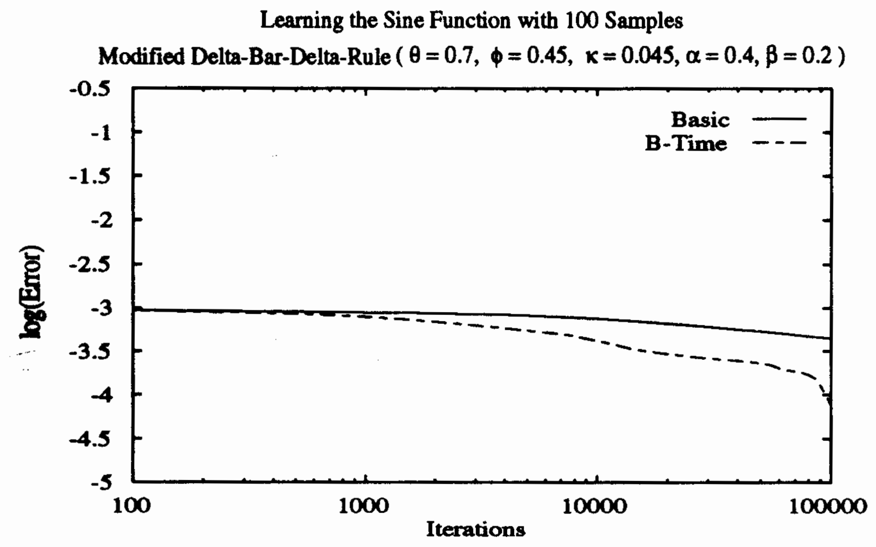

Figure 17: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.45, \alpha = 0.4$).

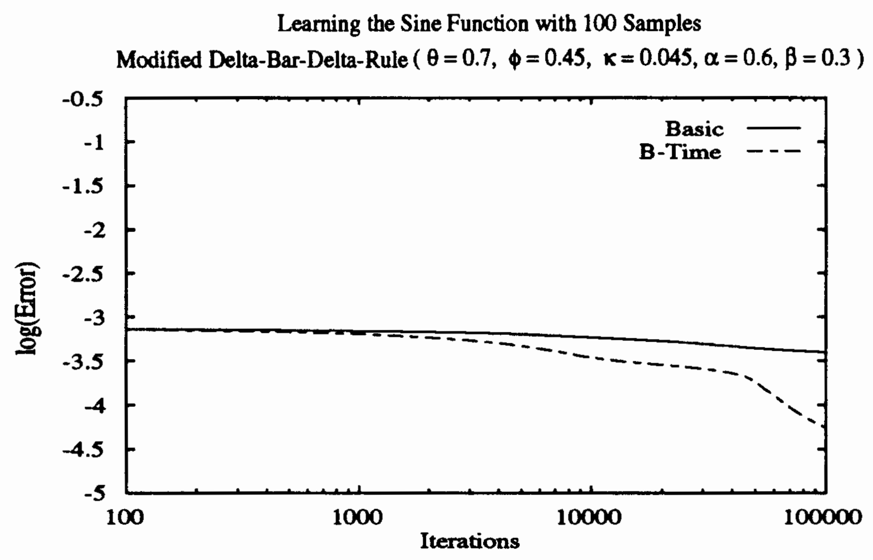

Figure 18: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.45, \alpha = 0.6$).

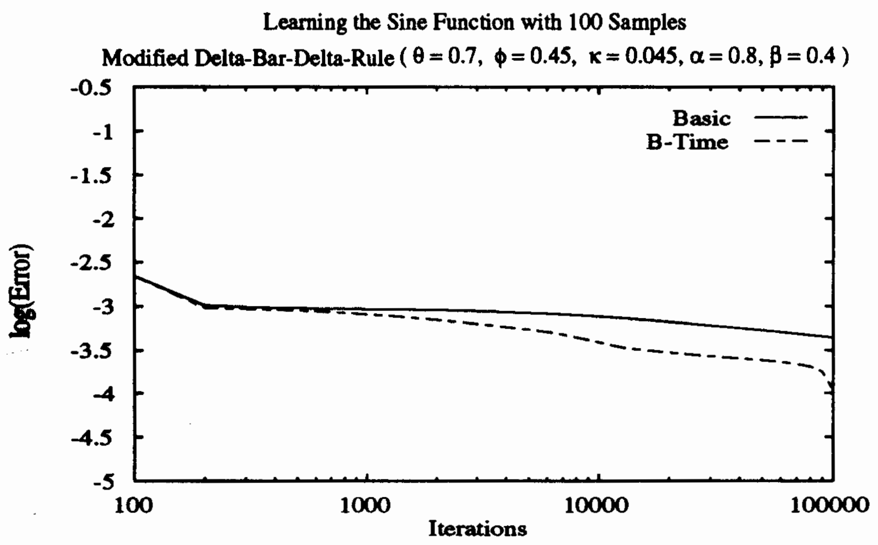

Figure 19: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.45, \alpha = 0.8$).

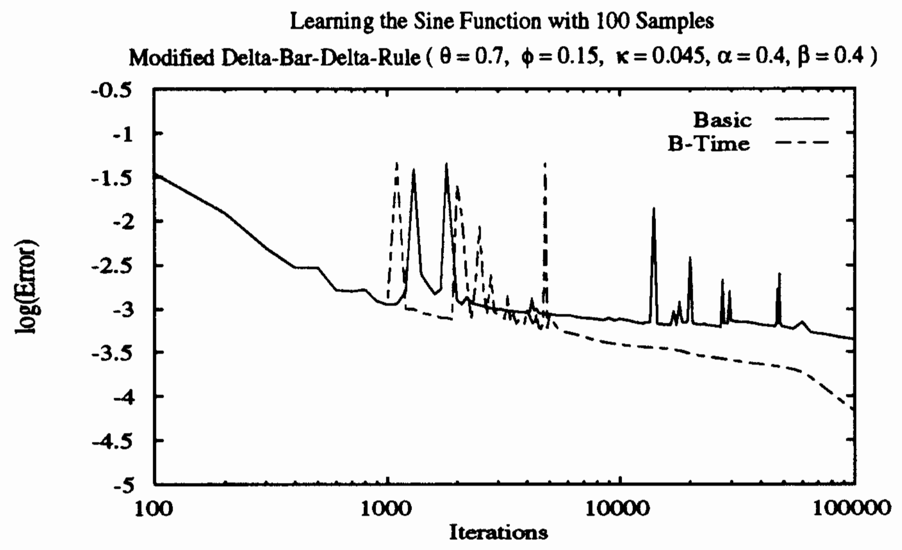

Figure 20: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.15, \alpha = 0.4$).

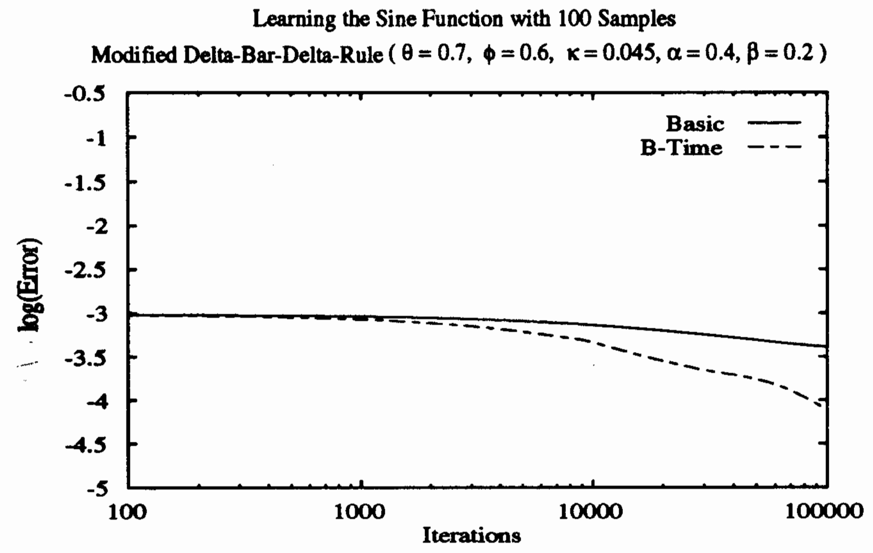

Figure 21: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.6, \alpha = 0.4$).

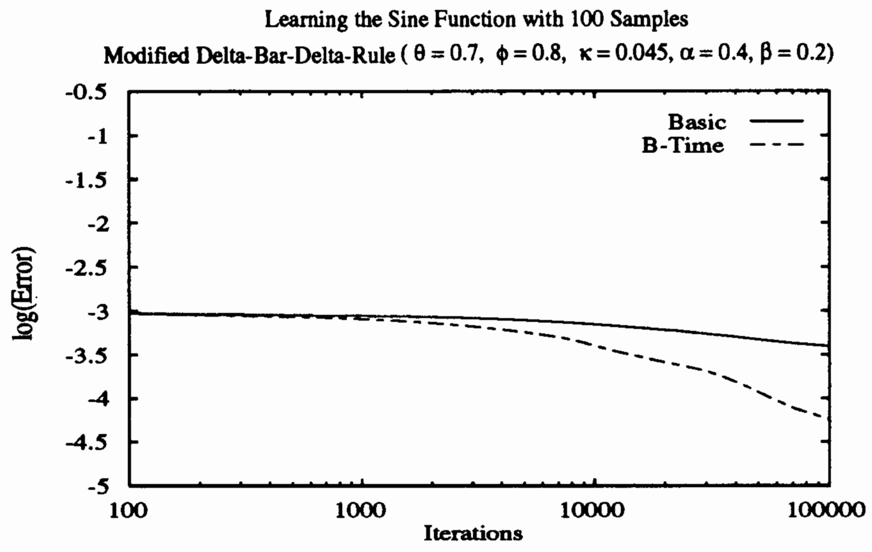

Figure 22: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.8, \alpha = 0.4$).

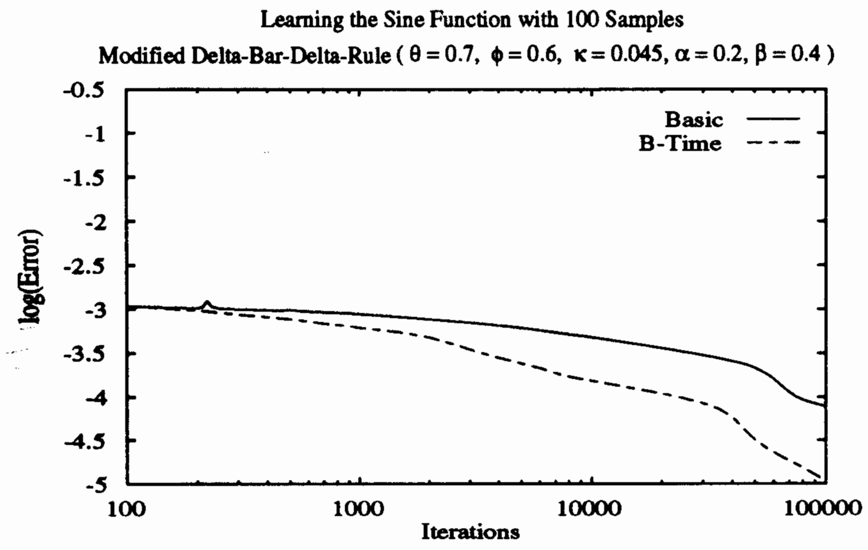

Figure 23: *Learning with Delta-Bar-Delta Rule* ($\phi = 0.6, \alpha = 0.2$).