9-1-1992

Implementation of back-propagation neural networks with MatLab

Jamshid Nazari
Purdue University School of Electrical Engineering

Okan K. Ersoy
Purdue University School of Electrical Engineering

Implementation of Back-Propagation Neural Networks with Matlab

Jamshid Nazari
Okan K. Ersoy

TR-EE 92-39
September 1992

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907-1285

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The artificial neural network back propagation algorithm is implemented in **Matlab** language. This implementation is compared with several other software packages. **The** effect of reducing the number of iterations in the performance of the algorithm is studied. The speed of the back propagation program, ***mbackprop***, written in **Matlab** language is compared with the speed of several other back propagation programs which are written in the C language. The speed of the **Matlab** program mbackpmp **is** also compared with the C program ***quickprop*** which is a variant of the back propagation algorithm. It is shown that the **Matlab** program mbackpmp is about 4.5 to 7 times faster than the C programs.

1. BACK PROPAGATION ALGORITHM USING MATLAB

This chapter explains the software package, mbackprop, which is written in Matlab language. The package implements the Back Propagation (BP) algorithm [RIIW86], which is an artificial neural network algorithm.

There are other software packages which implement the back propagation algorithm. For example the Aspirin/MIGRAINES Software Tools [Lei91] is intended to be used to investigate different neural network paradigms. There is also NASA NETS [Baf89] which is a neural network simulator. It provides a system for a variety of neural network configurations which uses generalized delta back propagation learning method. There are also books which have implementation of BP algorithm in C language for example, see [ED90].

Many of these software packages are huge, they need to be compiled and sometimes difficult to understand. Modification of these codes requires understanding the massive amount of source code and additional low level programming. The mbackprop on the other hand is easy to use and very fast. With the graphical capability of the Matlab the network parameters can be graphed to see what is going on inside any specific network. Additions and modifications to the mbackprop package are easier and further research in the area of neural network can be facilitated.

1.1 What is Matlab?

Matlab is a commercial software developed by MathWorks Inc. It is an interactive software package for scientific and engineering numeric computation [Inc90]. Matlab has several basic routines which do matrix arithmetics, plotting etc.

1.2 Why Use Matlab?

Matlab is already in use in many institutions. It is used in research in academia and industry. Prototype solutions are usually obtained faster in Matlab than solving a problem from a programming language.

Matlab is fast, because the core routines in Matlab are fine tuned for different computer architectures. Following test was made to compare the speed between Matlab and a program written in C. Since the back propagation algorithm involves nnatrix manipulations the test chosen was matrix multiply. As the next section shows, Matlab[1] was about 2.5 times faster than a C program both doing a matrix multiply.

Speed Comparison of Matrix Multiply in Matlab and C

A program in C was written to multiply two matrices containing double precision numbers. The result of the multiplication is assigned into a third matrix. Each matrix contained 500 rows and 500 columns. A Matlab M file was written to do the same multiply as C program did. Only the segment of the code which does the multiplication is timed. The test was run on an IPC-SparcStation computer, the result is shown in Table 1.1. As the table shows Matlab is faster than the C program by more than a factor of two.

Table 1.1 Speed comparison of matrix multiply in Matlab and a C program. Matlab runs 2.5 times faster than the C program.

Method	Execution Time 500 x 500 Multiply
C Program	277 seconds
Matlab	110 seconds

[1]The version of Matlab we used was 3.5i.

1.3 Back Propagation Algorithm

The generalized delta rule [RHW86], also known as back propagation algorithm is explained here briefly for feed forward Neural Network (NN). The explanation here is intended to give an outline of the process involved in back propagation algorithm.

The NN explained here contains three layers. These are input, hidden, and output layers. During the training phase, the training data is fed into to the input layer. The data is propagated to the hidden layer and then to the output layer. This is called the ***forward pass*** of the back propagation algorithm. In ***forward pass,*** each node in hidden layer gets input from all the nodes from input layer, which are multiplied with appropriate weights and then summed. The output of the hidden node is the non-linear transformation of the this resulting sum. Similarly each node in output layer gets input from all the nodes from hidden layer, which are multiplied with appropriate weights and then summed. The output of this node is the non-linear transformation of the resulting sum.

The output values of the output layer are compared with the ***target output values.*** The ***target output values*** are those that we attempt to teach our network. The error between actual output values and target output values is calculated and propagated back toward hidden layer. This is called the ***backward pass*** of the back propagation algorithm. The error is used to update the connection strengths between nodes, i.e. weight matrices between input-hidden layers and hidden-output layers are updated.

During the testing phase, no learning takes place i.e., weight matrices are not changed. Each test vector is fed into the input layer. The feed forward of the testing data is similar to the feed forward of the training data.

1.4 Mbackprop Program

The ***mbackprop*** program is written in Matlab language. The program implements the back propagation algorithm [RHW86]. The algorithms used in the ***mbackprop*** program involve very few number of iterations. This is one of the reasons why this

program is so fast. In the next section, an example is given to see the effect of reducing number of iterations has on the execution speed of a program. In Section 1.5 execution speed of the *mbackprop* program in Matlab is compared with the execution speed of a back propagation program in C.

Reducing Number of Iterations Increases Execution Speed

There are several ways to write a program to accomplish a given task. The approach or algorithm a person might take will have a great effect on the execution speed of a program. Here, a ***class identification problem*** is stated and then two solutions are presented. Statement of the problem is, given a matrix A, find the class to which each column of the matrix A belongs.

Each column of the matrix A is a vector x which we want to find to which class this vector belongs. To do this, for each of these vectors x, we want to find the distances between the vector x and m other vectors. These m vectors are the desired vectors representing $class_1$ through $class_m$. The minimum of the m distances, comes from a vector representing $class_j$. The number j is the answer to the column vector x. So the desired output is a row vector B indicating to which class each of the vectors in A belongs. Content of the matrix A is changing, so we need to calculate the row vector B more than once.

Two solutions are now presented for the above problem. The first solution will be algorithm 1 and the second solution will be algorithm 2. Both algorithms will need as input argument the following variables:

- variable "A" which contains the matrix A
- variable "Classes" which contains vectors representing $class_1$ through $class_m$
- variable "nClasses" which contains the number of classes m.

The output of the both algorithms is variable "B" which will contain the class number of each column of the variable "A".

Figure 1.1 shows several of the variables used in algorithm 1. Here the variable "A^n

is made of columns $x_1, x_2, \ldots, x_n$. Variable 'Classes'' is made of columns $c_1, c_2, \ldots, c_m$ which represents $class_1$ through class,. Variable 'dist'' is a column vector of size m which will hold the distance of a vector x in A to each of the m classes in variable "Classes". The algorithm 1 is the following:

- for each x_i in A where $i = 1, \ldots, n$
  - $dist(j)$ = Square Euclidean Distance(x;, c_j) where j = 1, . . . , m
  - $B(i)$ = k where $dist(k) = \min($ dist)

Figure 1.2 shows several of the variables used in algorithm 2. Here the variable 'A' is also made of columns $x_1, x_2, \ldots, x_n$ but we will view it as one block. Variable 'Classes' is made of m column blocks $C_1, \ldots, C_m$, where m is the number of classes. Each block C_j is the same size as block A. The block C_j contains n equal columns where n is the number of columns in A. Each column in block C_j is c_j which represents $class_j$. Variable "dist' is made of m block columns which will hold the distance of block A to each of the m blocks in variable "Classes". The algorithm 2 is the following:

- $dist(j,:)$ = Square Euclidean Distance(A, C_j) where $j = 1, \ldots, m$. $dist(j,:)$ refers to row j of dist matrix.
- $B(i) = k$ where $dist(k,i) = \min($ $dist(:,i)$). $dist(:,i)$ refers to column i of dist matrix.

Tables 1.2 and 1.3 show the two solutions for the class identification problem using algorithms 1 and 2. Note that these solutions are written in Matlab language. The algorithm 1 used in Table 1.2 is straight forward. As shown in the next section, the algorithm 1 contains much more iterations than algorithm 2. This causing the algorithm 1 to run slower than the algorithm 2 of Table 1.3.

Speed Comparison of Algorithm 1 and Algorithm 2

The above algorithms were used to solve the class identification problem, where the number of classes was 8. The size of the variables used in algorithms 1 and 2

Table 1.2 Algorithm 1 is a straight forward method which solves the class identification problem.

```
1) function B=algorithm1( A, Classes, nClasses )
2) % Each column of the variable "Classes" represents a class
3) [ nRow, nCol ] = size( A );
4) B = zeros( 1, nCol );                    % Preallocate memory
5) for i = 1 : nCol,
6)     x = A( :, i );
7)     for j = 1 : nClasses,
8)         dist( j ) = sum( ( x - Classes( :, j ) ) .^ 2 );
9)     end
10)    [ v, B(i) ] = min( dist );
11) end
```

are shown in Table 1.4. Note that the amount of memory used by algorithm 2 (1922 Kbytes) is much greater than the memory used in algorithm 1 (212 Kbytes). However, as shown below, algorithm 2 is much faster than algorithm 1. The speed of execution is related to the number of iterations in the algorithm.

The number of iterations for algorithm 1 is much greater than the number of iterations for algorithm 2. In this example, the statement number 8 in algorithm 1 gets executed 24,000 (nCol x nClasses = 3000 $\times$ 8) times. Where in algorithm 2 either statement number 11 or 15 gets executed only 8 (nClasses = 8) times. Since Matlab is an interpretive language algorithm 1 is much slower than algorithm 2. Table 1.5 shows that algorithm 2 runs about 23 times faster than algorithm 1. The test was performed on an IPC-SparcStation computer. In the next section the speed of *mbackprop* program, written in Matlab, is compared to the speed of a C program both implementing the back propagation algorithm [RHW86].

Table 1.3 Algorithm 2 is another way to solve the class identification problem. It is :faster than Algorithm 1.

```
1) function B=algorithm2( A, Classes, nClasses )
2) % Each column of the variable "Classes" represents all of the
3) % "nClasses" classes. If there are 8 classes and each class is
4) % represented by 8 numbers, then the number of rows of "Classes" is
5) % equal to 64. The number of columns in "Classes" is equal to number
6) % of columns in A.
7) [ nRow, nCol ] = size( A );
8) dist = zeros( nClasses, nCol );        % Preallocate memory
9) if nRow == 1
10)    for j = 1 : nClasses,
11)       dist( j, : ) = ( A - Classes( j, : ) ) .^ 2;
12)    end
13) else
14)    for j = 1 : nClasses,
15)       dist( j, : ) = sum( ( A - Classes( ( ( j - 1 ) * nRow + 1 ) : ( j * nRow ), : ) ) .^ 2 );
16)    end
17) end
18) [ v, B ] = min( dist );
```

1.5 Matlab Backprop Speed vs. C Backprop Speed

The back propagation program in **Matlab,** mbackprop, is compared with two other C back propagation programs fbackprop[2] and dbackprop. The mbackprop is also compared with the C program quickprop **[Fah88].** The quickprop program is a modification of a back propagation program which has similar feed forward and backward routines but in update weight routine, all the weights are updated as a function of each weight's current slope, previous slope, and the size of the last jump. However, if the variable "ModeSwitchThreshold" in the quickprop program is set to a big number then all the weight updates are based on normal gradient descent method **i.e.** same as in regular back propagation algorithm.

The program fbackprop is similar to the program dbackprop. The only difference is that the calculations in ***fbackprop*** are in floats (single precision), where the calculations

[3]**The** *fbackprop* program has been used in some of our research at Purdue University for last two years.

Table 1.4 Size of the variables in algorithms 1 and 2 is shown here. The amount of memory used in algorithm 1 is 212 Kbytes where algorithm 2 uses 1922 Kbytes.

Variable Name	Size of Variable in Algorithm 1	Size of Variable in Algorithm 2	#Bytes Alg 1	#Bytes Alg 2
A	8 x 3000	8 x 3000	192,000	192,000
B	1 x 3000	1 x 3000	24,000	24,000
Classes	8 x 8	64 x 3000	512	1,536,000
dist	8 x 1	8 x 3000	64	192,000
i	1 x 1		8	
j	1 x 1	1 x 1	8	8
nClasses	1 x 1	1 x 1	8	8
nCol	1 x 1	1 x 1	8	8
nRow	1 x 1	1 x 1	8	8
v	1 x 1	1 x 3000	8	24,000
x	8 x 1		64	

in *dbackprop* are in doubles (double precision). All the calculations in Matlab program *mbackprop* are in doubles. The calculations in the *quickprop* program are in floats.

In the *fbackprop* and the *dbackprop* programs, weights get updated after every input/output vector pair. Where the weights in *quickprop* and *mbackprop* programs get updated after a complete sweep of the training data. As shown below the *mbackprop* program is faster than all the three C programs.

The neural network, used in our benchmark tests, had 64 input nodes, 16 hidden nodes, and 8 output nodes. The training data contained 1600 input and output vector pairs. Each input vector was 64 numbers and each output vector was 8 numbers. The training time for 100 sweeps over the training data is measured for the above programs using an IPC-SparcStation Computer. The results are shown in Table 1.6. As the table shows, the *mbackprop* runs 7.0 times faster than the C program *dbackprop*[3]. The *mbackprop* runs 4.5 times faster than the C program *quickprop*.

The training time for the C programs *fbackprop*[4] and *dbackprop* is also measured in two other computers[5]. One of the computers was a Vax 11/780 and the other

[3]The training time for *fbackprop*, *dbockpmp*, and *quickpmp* are measured for 10 iterations. To get the training time for 100 iterations, the measured numbers are multiplied by 10.

[4]The training time for *fbackprop* and *dbackpmp* are measured for 1 iteration. To get the training time for 100 iterations, the measured numbers are multiplied by 100.

[5]We did not have Matlab running in these computers

Table **1.5** Speed of algorithm **1** is compared to the speed of algorithm 2. Algorithm **2** runs about **23** times faster than algorithm 1.

Algorithm	Execution Time
1	51.22 seconds
2	2.26 seconds

Table **1.6** Speed of the Matlab program ***mbackprop*** is compared to the speed of C programs ***fbackprop, dbackprop*** and ***quickprop.*** The training time for 100 sweeps over the training data is measured. The Matlab program ***mbackprop*** runs 7.0 times faster than the C program ***dbackprop.*** The ***mbackprop*** also runs **4.5** times faster than the ***quickprop*** program.

Program	Execution Time
fbackprop	3969 seconds
dbackprop	3792 seconds
mbackprop	536 seconds
quickprop	2407 seconds

computer was a Zenith **386/33** running SCO unix with math coprocessor. Table 1.7 shows the training time for **100** iterations over the training data. The time taken for the ***fbackprop*** program was less than the ***dbackprop*** program in these computers, however the ***fbackprop*** time of the IPC-SparcStation computer was longer the ***dbackprop*** time. So depending on different computer architectures floating point single precision calculations are faster than double precision calculations or vice versa.

As it was shown, the ***mbackprop*** program was fastest among the programs we considered above. However ***mbackprop*** provides an integrated graphic capability that other programs lack.

1.6 Integrated Graphical Capability of the Mbackprop Program

The back propagation program ***mbackprop*** is faster than the C programs considered here. However this is not the only advantage that this program has over others. It provides an integrated graphical capability and an interprative environment that other programs lack.

Table 1.7 Execution speed of the ***fbackprop***, a single precision back propagation program, is compared to the ***dbackprop*** a double precision back propagation program. The training time for 100 sweeps over the training data is measured on three computers. In the IPC-SparcStation computer double precision program was faster than the single precision program. In Vax 11/780 and 386 computer the single precision program is faster than the double precision program. The execution time of the ***rnbackprop*** is included here for comparison.

Computer	Execution Time for fbackprop	Execution Time for Dbackprop	Execution Time for Mbackprop
IPC-SparcStation	3,969 seconds	3,792 seconds	536 seconds
Vax 11/780	69,923 seconds	73,150 seconds	
Zenith 386/33	21,795 seconds	24,523 seconds	

In the ***mbackprop*** program, the network parameters can be easily viewed during program execution. Training and testing reports can be enabled during training and statistics such as mean square error, percent correct, etc. can be collected in report intervals specified by the user.

Figure 1.3 shows a sample ***'mean square error' graph.*** Figure 1.4 shows a sample ***'percent correct'*** graph. Percent correct refers to the percentage of the input vectors, in training or testing data, which are correctly classified. Figure 1.5 shows a sample ***'maximum absolute error'*** graph. Figure 1.6 shows a sample ***'percent bits wrong'*** graph. Percent bits wrong refers to the percentage of the output nodes which were off more than some threshold. Figure 1.7 shows a sample ***'compact graph'*** graph. Compact graph is a graph which contains the above 4 graphs in one graph.

Other graphs can be easily added to the ***mbackprop*** package. In the next section vie list several other capabilities of the ***mbackprop*** program.

1.7 Other Capabilities of the *Mbackprop* Package

So far we have shown that ***mbackprop*** package is fast and contains several standard graphical capabilities. Several of the ***mbackprop*** capabilities are:

- allows a user to specify a weight file for initial weights to start training.

- can generate random initial weights for training and allows the user to save these initial weights to be used later.
- if the training gets started in wrong initial weights, the program is easily interrupted and different set of initial weights is used.
- the result from training a network can be saved and recalled at a later time.
- allows further training from where it was last left off.

1.8 Summary and Conclusions

The ***mbackprop*** program is written in **Matlab** language. This program implements the back propagation algorithm [RHW86]. Since **Matlab** is an interpretive language the number of iterations in the algorithms have been reduced. This reduced number of iterations results in a faster executable program. The ***mbackprop*** program is faster than the C back propagation program ***dbackprop*** by a factor of 7.0. It is faster than ***quickprop*** [Fah88] program by a factor of 4.5. The ***mbackprop*** provides other capabilities such as integrated graphics and interpretive environment which **Matlab** offers.

The ***mbackprop*** size is less than a comparative program in C. It is modular and each individual module can be viewed as a software Integrated Chip (IC). Each software IC can be modified as long as the input/output criteria is met. Additions of other software ICs is easy to be incorporated into the ***mbackprop*** package. Further research in the area of neural network can be facilitated.

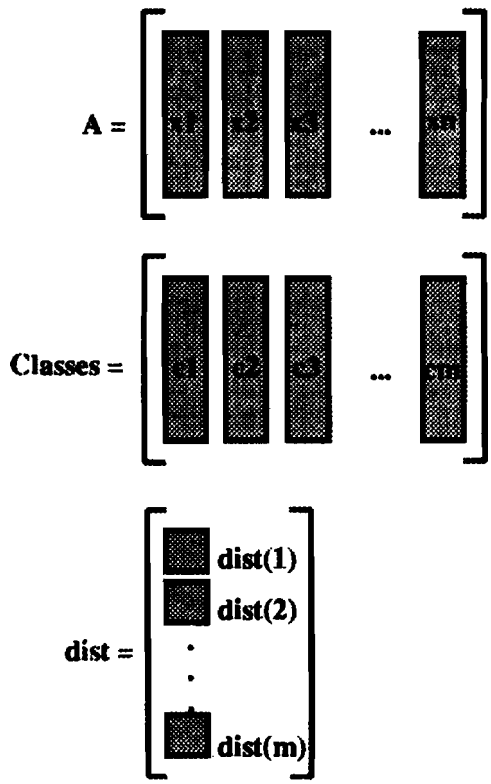

Figure 1.1 Some of the variables used in algorithm 1 is shown here.

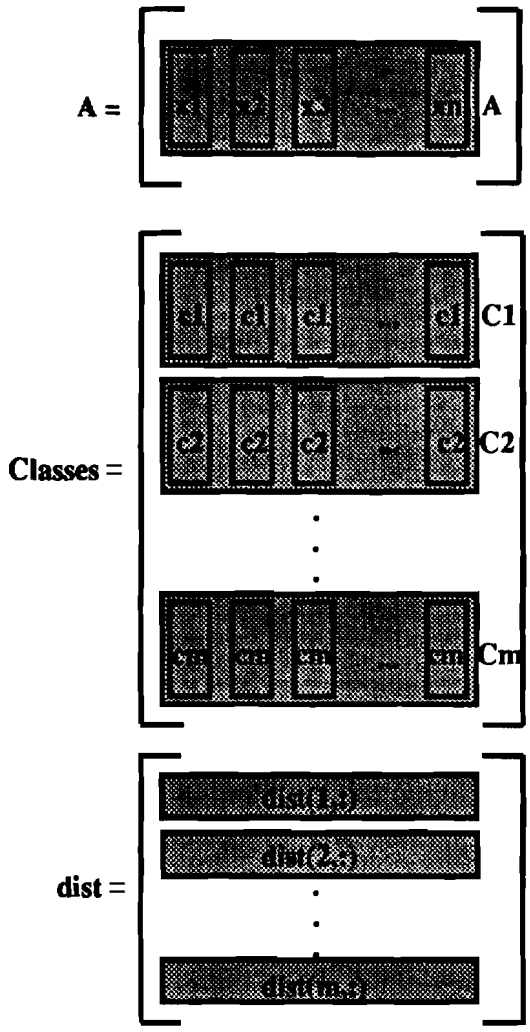

Figure 1.2 Some of the variables used in algorithm 2 is shown here.

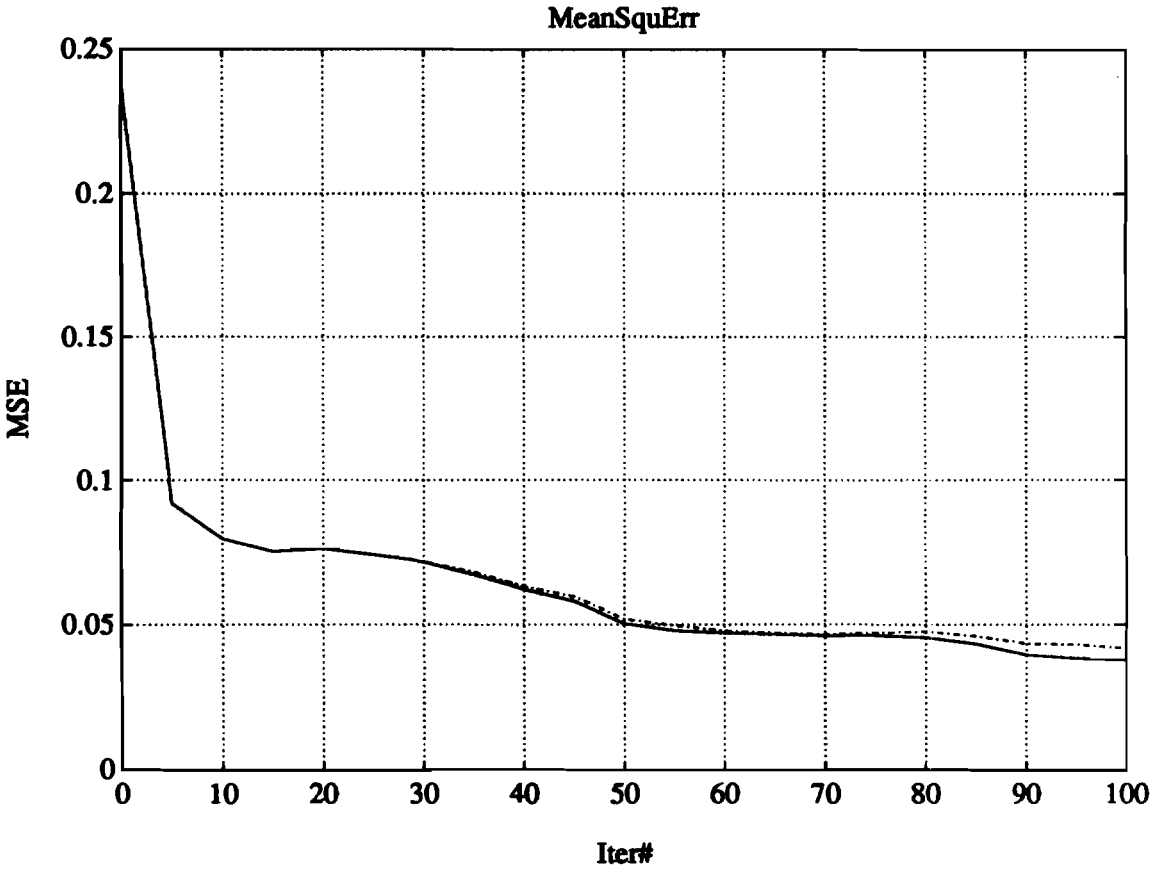

Figure 1.3 A sample 'mean square *error*' graph, generated by the mbackprop program, is shown here. The solid line is the training mean square error and the dashed line is the testing mean square error.

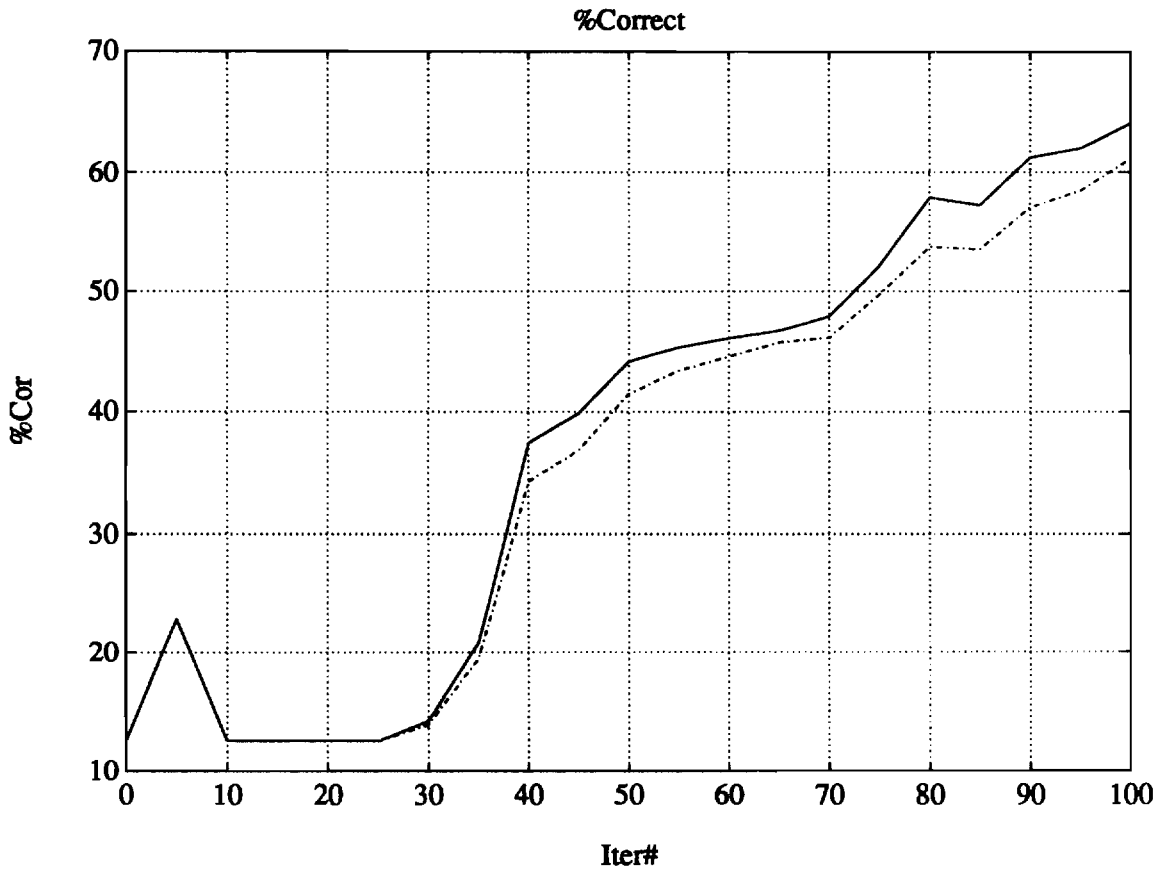

Figure 1.4 A sample 'percent *correct'* graph, generated by the mbnckprop program, is shown here. The solid line is training percent correct and the dashed line is the testing percent correct.

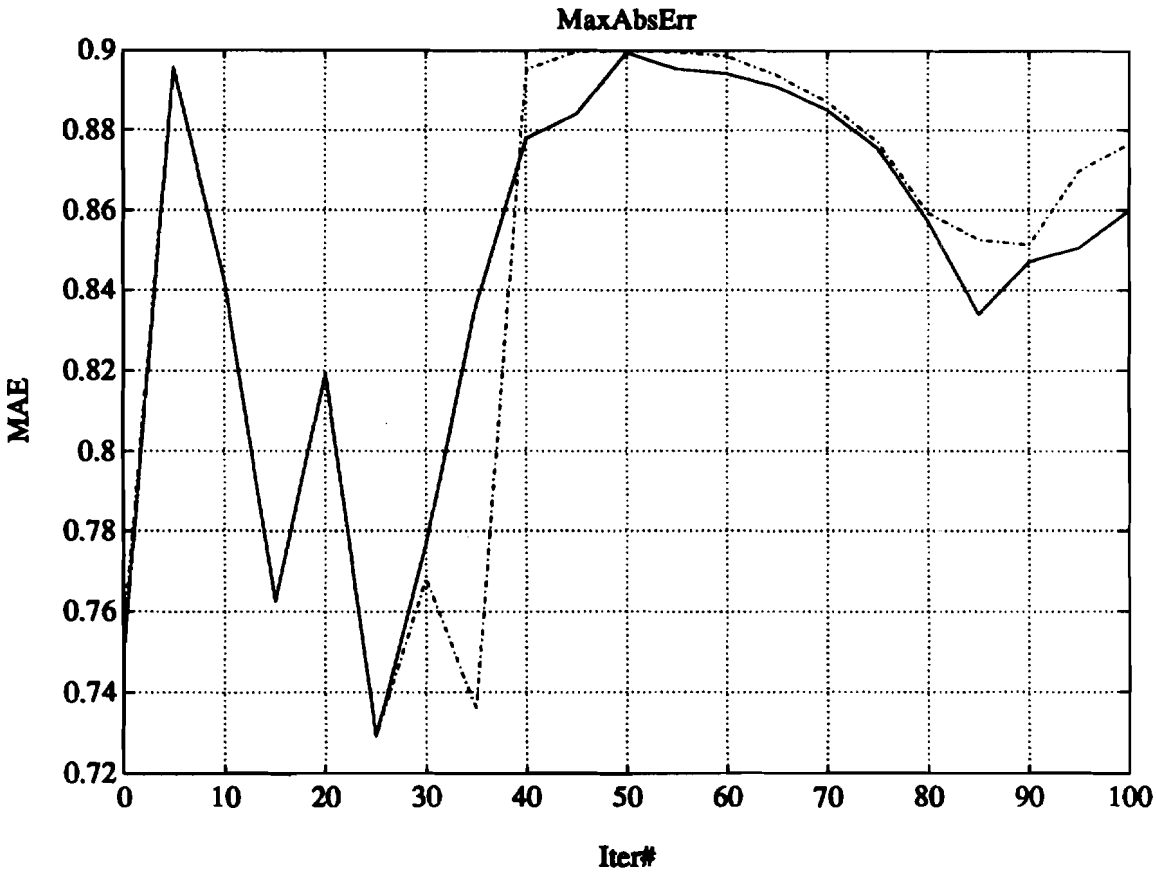

Figure 1.5 A sample *'maximum absolute* error' graph, generated by the *mbackprop* program, is shown here. The solid line is training maximum absolute error and the dashed line is the testing maximum absolute error.

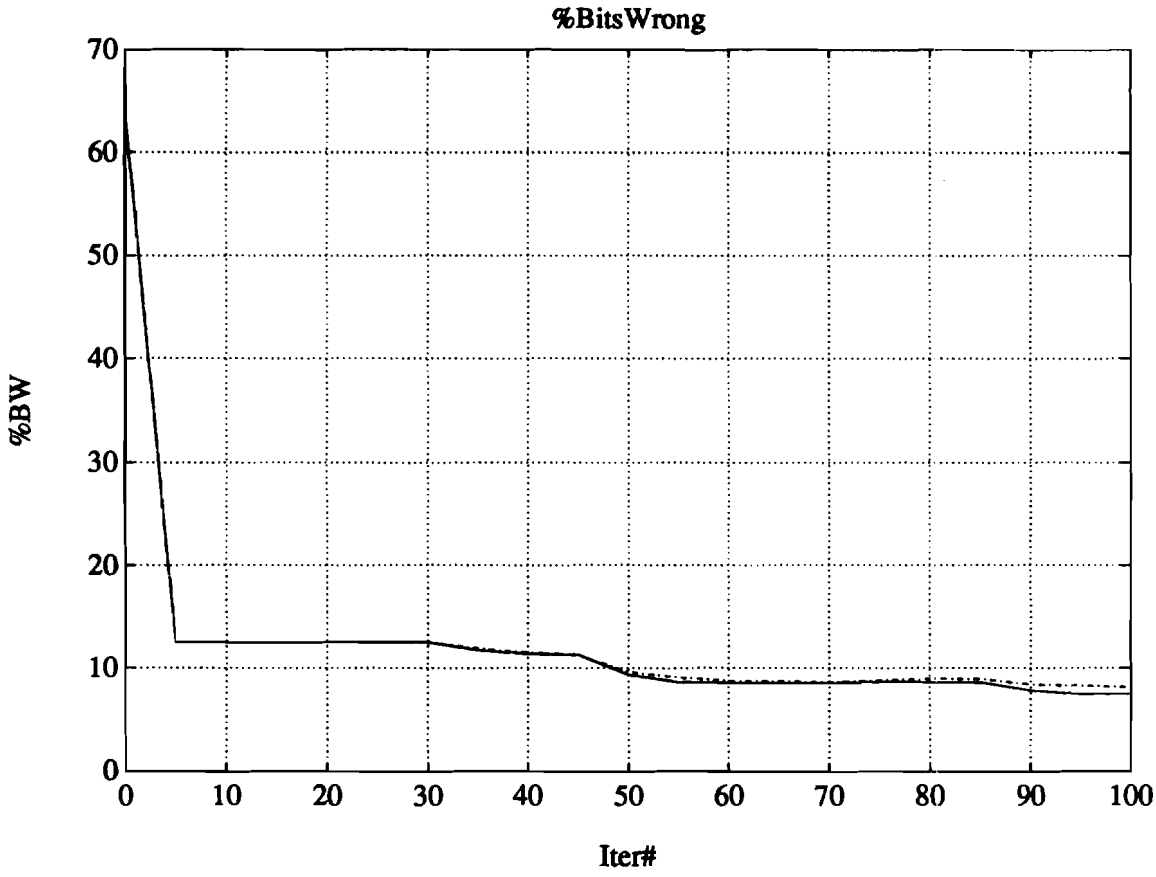

Figure 1.6 A sample 'percent bits wrong' graph, generated by the mbnckprop program, is shown here, The solid line is the training percent bits wrong and the dashed line is the testing percent bits wrong.

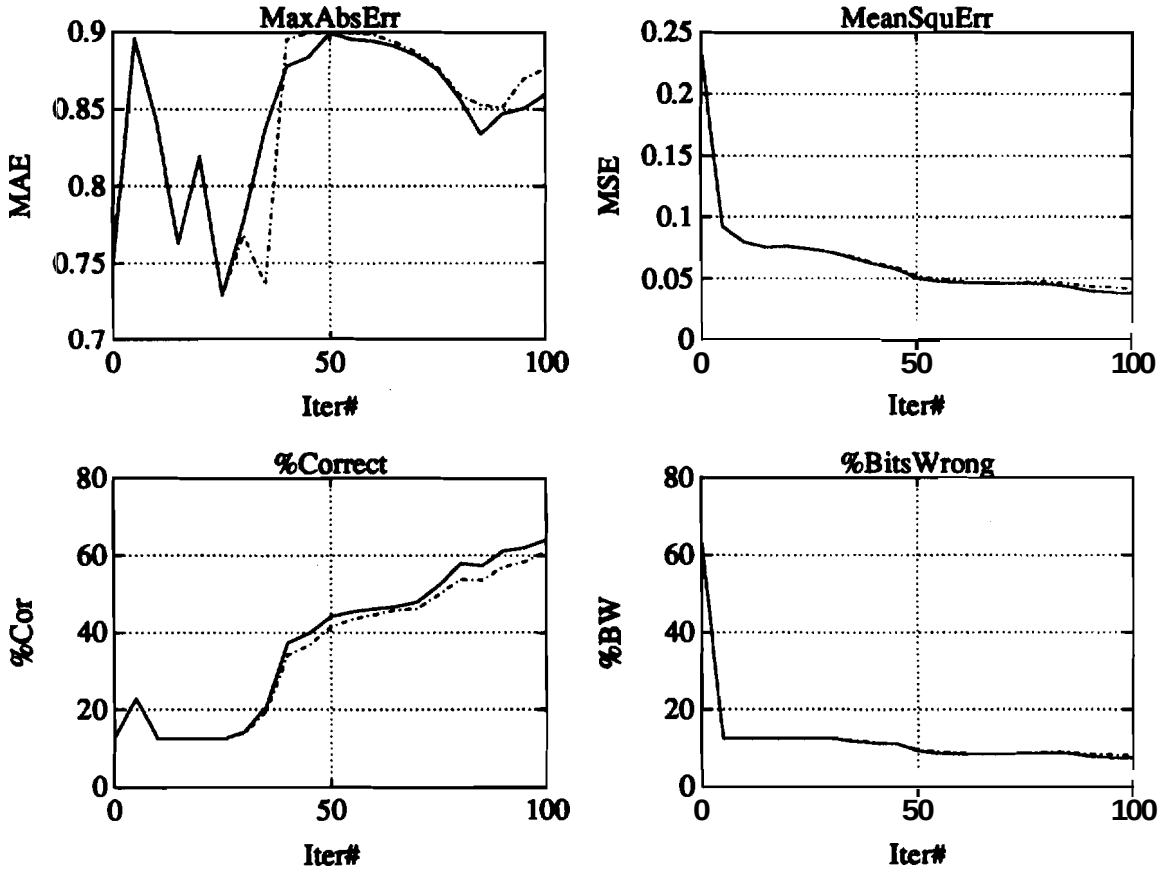

Figure 1.7 A sample *'compact* 'graph, generated *by* the *mbackprop* program, is shown here. The solid lines refers to the training results and the dashed lines refer to the testing results.

BIBLIOGRAPHY

BIBLIOGRAPHY

[Baf89] Paul T. Baffes. NETS Users's Guide, Version 2.0 of NETS. Technical Report JSC-23366, NASA, Software Technology Branch, Lyndon B. Johnson Space Center, September 1989.

[ED90] R. C. Eberhart and R. W. Dobbins. ***Neuml Network PC Tools, A Practical Guide.*** Academic Press, San Diego, California 92101, 1990.

[Fah88] E. Fahlman, Scott. An Empirical Study of Learning Speed in Back-Propagation Networks. Technical Report CMU-CS-88-162, CMU, CMU, September 1988.

[Inc90] The MathWorks Inc. ***PRO-MATLAB for Sun Workstations, User's Guide.*** The MathWorks Inc., January 1990.

[Lei91] Russell R. Leighton. The Aspirin/MIGRAINES Software Tools, User's Manual, Release V5.0. Technical Report MP-91W00050, MITRE Corporation, MITRE Corporation, December 1991.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. ***Learning Internal Representaions by Error Propagation in Rumelhart, D. E. and McClelland, J. L., Parallel Distributed Processing: Explorations in the Microstructure of Cognition.*** MIT Press, Cambridge Massachusette, 1986.