

PENALTY METHODS TO REDUCE OVERFITTING  
IN ARTIFICIAL NEURAL NETWORKS

By

ZHONG XIANG LUO

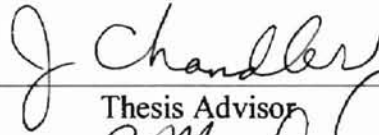
Bachelor of Engineering  
Gezhouba Hydroelectric Engineering  
University  
Yichang, China  
1982

Master of Engineering  
Wuhan Hydraulic and Electric  
Engineering University  
Wuhan, China  
1989

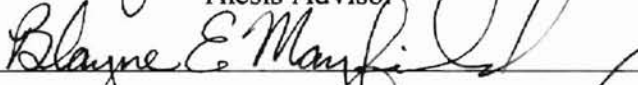
Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
In partial fulfillment of  
The requirements for  
The Degree of  
MASTER OF SCIENCE  
December, 1998

PENALTY METHODS TO REDUCE OVERFITTING  
IN ARTIFICIAL NEURAL NETWORKS

Thesis Approval:

  
\_\_\_\_\_

Thesis Advisor

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

Dean of the Graduate College

## ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. John P. Chandler, for his invaluable guidance and encouragement throughout this research. Thanks are also due my other graduate committee members, Dr. B. E. Mayfield, and Dr. G. E. Hedrick, for their time and valuable assistance and cooperation. I also want to thank the faculty and staff in Computer Science Department at Oklahoma State University for their support.

More over, I would like to express my special thanks and love to my mother, Ying Tong, my uncles, Shu Chen Luo, You Chen Luo, my old sisters Yue Luo, Yuewo Luo, for their support and understanding throughout my life.

Finally, I would like to give my special thanks to my wife, Dr. Ming Yu, my old daughter, Brooke T. Luo, and my new born baby Rynel Luo, for their love, patience, and support during this research. My wife's encouragement and faith always provided me strength and comfort. My old daughter's love always makes me happy and helped me get through all difficulties. My new born baby brings me excitement and joys.

## TABLE OF CONTENTS

Chapter	Page
<b>I. INTRODUCTION</b> .....	1
<b>II. METHODS FOR REDUCING OVERFITTING</b> .....	5
Overfitting and Generalization in Artificial Neural Networks .....	5
Methods of Reducing Overfitting .....	7
Penalty Mechanism and Algorithm.....	7
Penalty Terms as a Method of Reducing Overfitting .....	13
<b>III. NEURAL NETWORK ARCHTECTURE AND LEARNING GORITHMS</b> .....	18
Architectures of Feedforward Artificial Neural Networks.....	18
Activation Function.....	21
Weights in the Neural Network .....	21
Optimization Algorithm .....	23
Forward Computations .....	26
Back-propagation Computation .....	27
Algorithms for the Penalty Method and Improved Penalty Method.....	32
<b>VI. METHODS TESTED AND IMPLEMENTATION</b> .....	37
Neural Network Architecture Design.....	37
Discussion of Test Results.....	40
<b>V. CONCLUSION</b> .....	45
<b>REFERENCES</b> .....	47
<b>APPENDIX A</b> .....	50
Testing Tables.....	50
<b>APPENDIX B</b> .....	65
Computer Programs.....	65



## LIST OF TABLES

Table	Page
4_1 Overview of method tested .....	39
4_2 Performance Comparison of Different Method for Network with 7 hidden nodes .....	44
4_3 Performance Comparison of Different Method for Network with 8 hidden nodes .....	44
4_4 Performance Comparison of Different Method for Network with 10 hidden nodes .....	44
A_1 Performance of Training and Generalization(RMS) Method A with 7 hidden nodes and $\lambda = 0.01$ .....	50
A_2 Performance of Training and Generalization(RMS) Method B with 7 hidden nodes and $\lambda = 0.01$ .....	50
A_3 Performance of Training and Generalization(RMS) Method B with 7 hidden nodes and $\lambda = 0.0001$ .....	51
A_4 Performance of Training and Generalization(RMS) Method C with 7 hidden nodes and $\lambda = 0.0001$ .....	51
A_5 Performance of Training and Generalization(RMS) Method C with 7 hidden nodes and $\lambda = 0.01$ .....	51
A_6 Performance of Training and Generalization(RMS)	

	Method D with 7 hidden nodes and $\lambda = 0.0001$ .....	52
A_7	Performance of Training and Generalization(RMS)	
	Method D with 7 hidden nodes and $\lambda = 0.01$ .....	52
A_8	Performance of Training and Generalization(RMS)	
	Method E with 7 hidden nodes and $\lambda = 0.01$ .....	52
A_9	Performance of Training and Generalization(RMS)	
	Method E with 7 hidden nodes and $\lambda = 0.0001$ .....	53
A_10	Performance of Training and Generalization(RMS)	
	Method A with 8 hidden nodes .....	53
A_11	Performance of Training and Generalization(RMS)	
	Method B with 8 hidden nodes and $\lambda = 0.01$ .....	53
A_12	Performance of Training and Generalization(RMS)	
	Method B with 8 hidden nodes and $\lambda = 0.0001$ .....	54
A_13	Performance of Training and Generalization(RMS)	
	Method C with 8 hidden nodes and $\lambda = 0.0001$ .....	54
A_14	Performance of Training and Generalization(RMS)	
	Method C with 8 hidden nodes and $\lambda = 0.01$ .....	54
A_15	Performance of Training and Generalization(RMS)	
	Method D with 8 hidden nodes and $\lambda = 0.01$ .....	55
A_16	Performance of Training and Generalization(RMS)	
	Method D with 8 hidden nodes and $\lambda = 0.0001$ .....	55
A_17	Performance of Training and Generalization(RMS)	

	Method E with 8 hidden nodes and $\lambda = 0.001$ .....	55
A_17	Performance of Training and Generalization(RMS)	
	Method E with 8 hidden nodes and $\lambda = 0.01$ .....	56
A_18	Performance of Training and Generalization(RMS)	
	Method A with 10 hidden nodes .....	56
A_19	Performance of Training and Generalization(RMS)	
	Method B with 10 hidden nodes and $\lambda = 0.01$ .....	57
A_20	Performance of Training and Generalization(RMS)	
	Method B with 10 hidden nodes and $\lambda = 0.0001$ .....	57
A_21	Performance of Training and Generalization(RMS)	
	Method C with 10 hidden nodes and $\lambda = 0.00001$ .....	57
A_22	Performance of Training and Generalization(RMS)	
	Method C with 10 hidden nodes and $\lambda = 0.01$ .....	58
A_23	Performance of Training and Generalization(RMS)	
	Method D with 10 hidden nodes and $\lambda = 0.0001$ .....	58
A_24	Performance of Training and Generalization(RMS)	
	Method D with 10 hidden nodes and $\lambda = 0.01$ .....	58
A_25	Performance of Training and Generalization(RMS)	
	Method E with 10 hidden nodes and $\lambda = 0.01$ .....	59
A_26	Performance of Training and Generalization(RMS)	
	Method E with 10 hidden nodes and $\lambda = 0.00001$ .....	59
A-27	Performance of Training and Generalization (RMS)	

	Method F with 7 hidden nodes and different $\lambda$ (0.008, 0.006, 0.004 0.002, 0.001, 0.0006, 0.0001, 0.00006, 0.00004, 0.00001) .....	60
A_28	Performance of Training and Generalization (RMS) Method G with 7 hidden nodes and $\lambda = 0.001$ .....	60
A_29	Performance of Training and Generalization (RMS) Method G with 7 hidden nodes and $\lambda = 0.0001$ .....	60
A_30	Performance of Training and Generalization (RMS) Method G with 7 hidden nodes and $\lambda = 0.0002$ .....	61
A_31	Performance of Training and Generalization (RMS) Method G with 8 hidden nodes and $\lambda = 0.0002$ .....	61
A_32	Performance of Training and Generalization (RMS) Method G with 8 hidden nodes and $\lambda = 0.0001$ .....	62
A_33	Performance of Training and Generalization (RMS) Method G with 10 hidden nodes and $\lambda = 0.0001$ .....	62
A_34	Performance of Training and Generalization (RMS) Method G with 10 hidden nodes and $\lambda = 0.00001$ .....	62
A_35	The Training Data Set .....	63
A_36	The Validation Data Set .....	64

## LIST OF FIGURES

Table		Page
2.1	The Relationship Between Training Error and Testing Error .....	6
3.1	A Three Layer Feed forward Network .....	19
3.2	The Sigmoid Function .....	22
3.3	The Hyperbolic Function .....	22

## Chapter I

### INTRODUCTION

Artificial neural networks are computational models of the human brain. In contrast with conventional single-processor computers, the brain has a multiprocessor architecture that is highly interconnected. This architecture can be described as parallel distributed processing. Parallel distributed processing has many advantages over single-processor models for many difficult computer science problems. It allows problems that were once very difficult to solve on a computer to be attacked with relative ease.

Neural networks can be trained to develop operational capabilities to respond to an information environment. Supervised learning and unsupervised learning are the two main learning regimes used in neural network training.

A supervised learning algorithm adjusts the strengths or weights of the inter-neuron connections according to the difference between the desired and actual network outputs corresponding to a given input. Thus, supervised learning requires a teacher or supervisor to provide desired or target output signals. Examples of supervised learning algorithms include the delta rule [1], the generalized delta rule or backpropagation algorithm [2] and the LVQ algorithm [3].

Unsupervised learning algorithms do not require the desired outputs to be known. During training, only input patterns are presented to the neural network that automatically adapt the weights of its connections to cluster the input patterns into groups with similar features. Examples of unsupervised learning algorithms include the Kohonen [3] and Carpenter-Grossberg Adaptive Resonance Theory (ART) [4] competitive learning algorithms.

Neural Networks have been used in many fields including economics, transportation, defense, electronics, manufacturing, medicine, robotics, speech and telecommunications [1].

The Multi-layer Perceptron (MLP) will be discussed in this research. MLPs are perhaps the best-known type of feedforward networks. One of the interesting properties of a feedforward neural network is its capability of learning, i.e., a feedforward neural network can adjust its behavior using information from the environment. When a feedforward neural network is used to solve a problem, it is trained by a set of input-output sample data. Based on this data set, the network, when properly trained, will not only try to reproduce the sample set correctly, but also to generalize from the training examples to the entire problem domain.

A learning algorithm is applied a set of training data, then it is applied to make predictions on new data points. The goal is to maximize its predictive accuracy on the new data points. If it is trained too hard to find the very best fit to the training data, there is a risk that the data noise will be fitted by memorizing various peculiarities of the training data rather than finding a general predictive rule. For continuous domains, or large discrete ones, it is impossible to provide samples of every possible input. For a

large network, if the system simply memorizes the training patterns, it may do quite well during the training process but it may give spurious and misleading outputs if the input is slightly different from the sample inputs. This phenomenon is called overfitting. Overfitting is thought to happen when the network has more degrees of freedom than the number of the training samples. Obviously, a network can obtain a good generalization only when the number of parameters is less than the number of data points in the training set. Unfortunately, it is difficult to find the smallest neural network size that can learn the training data best.

Many techniques for reducing overfitting have been developed. The penalty-term method is one of the most popular methods. The basic approach used in a penalty-term method is adding penalty terms to the usual error function in order to constrain the search and cause weights to decay differentially. By modifying the cost function, the backpropagation will drive unnecessary weights close to zero and, in effect, remove them during training. Even if the weights are not actually removed, the network acts like a smaller system.

This thesis focuses on the possibilities of reducing the overfitting by using the penalty-term method in artificial neural networks. Many penalty terms have been developed to reduce overfitting. Some of them are complicated; some of them include a user-dependent constant factor. Each penalty term has different advantages and disadvantages. The question remains of whether there is a penalty term or a combination of penalty terms that can produce superior results and, if there is, what the penalty term could be. This research will compare and summarize different penalty terms through their performance. An improved penalty term method will be proposed in this research.



It is expected that the improved penalty method will improve the generalization performance of neural networks significantly. The paper is organized as follows:

In Chapter I, a general introduction to neural networks and the problems of interest is given.

In Chapter II, a review of different algorithms for reducing overfitting, especially the penalty term methods, will be conducted.

Chapter III will explain the architecture of the neural network that will be discussed in this research, and the application of optimization theory in the algorithm. A new penalty term method will be developed to reduce overfitting in this chapter.

In Chapter IV, an overview of methods that will be tested is given. The regular learning algorithm without a penalty term, the penalty method with different penalty terms, and the improved penalty method will be tested. All the methods and different penalty terms tested will be compared with each other through their generalization performance in the research.

Finally, the test results will be placed in Appendix A and the source program that is used in the implementation of the penalty term method and the improved penalty term method will be placed in Appendix B.

## **Chapter II**

### **METHODS FOR REDUCING OVERFITTING**

#### **Overfitting and Generalization in Artificial Neural Networks**

Mathematically, the objective of learning in the neural network is to infer a function from a given sample data set. Learning algorithms are designed essentially to search for a function that best fits the given data in a space of functions. After learning, the neural network is applied on the new data set. If it is trained too hard to find the best fit to the training data, there is a risk that we will fit the noise in to the data by memorizing various peculiarities of the training data rather than finding a general predictive rule [5]. When a network is trained, the weights are modified in order to decrease errors on the training data set. If the network is tested on a new set of data, the errors on the test data set tend to decrease in step with the training error as the network tries to generalize from the training data set to the underlined function. However if the training data is incomplete, it may contain spurious and misleading regularities due to sampling [6]. Figure 2-1 illustrates this situation schematically.

It is generally agreed that overfitting is closely related to the architecture of the network, i.e., the size of the network. If training starts with too small a network for the problem, good results cannot be obtained. If the network is too large, it may be

vulnerable to overfitting [20]. B. Baum and David Haussler [19] analyzed theoretically the lower and upper bounds on the size of the sample vs. the network size needed to achieve a valid generalization. Subutai Ahmad and Gerald Tesauro [21] analyzed how many training patterns and training cycles are needed for a problem of a given size and difficulty, how to represent the input, and how to choose training examples.

In general, overfitting is related to the degree of freedom of neural networks. The degree of freedom of neural networks includes not only the number of weights but also the potential non-linearity of the network, the architecture and the amount of time and the number of data used during training [22].

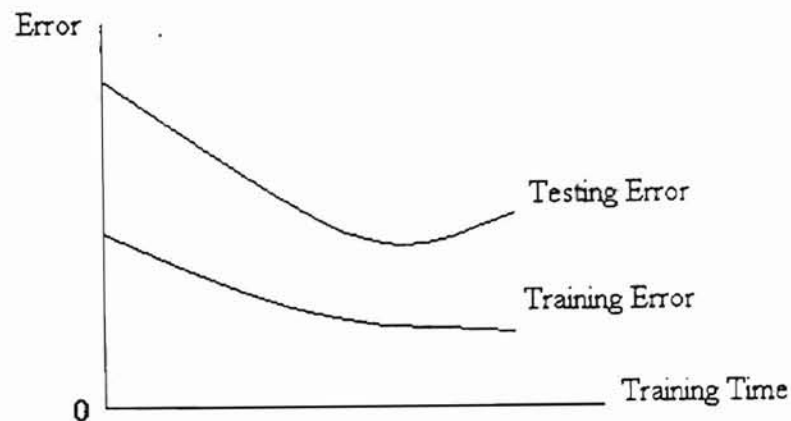


Figure 2\_1 The Relationship Between Training Error and Testing Error

## **Methods of Reducing Overfitting**

There are many methods to reduce overfitting and improve generalization [6] such as pruning methods, stopped training methods and penalty term methods. The pruning method is to train a network that is larger than necessary and then remove parts that are not needed. The large initial size allows the network to learn reasonably quickly with less sensitivity to initial conditions, while the reduced complexity of the trimmed system favors improved generalization. The stopped training method is to estimate the generalization ability during training and stop when it begins to decrease. The simplest method is to divide the data into a training set and a validation set. The training set is used to modify the weights, the validation set is used to estimate the generalization ability, and training is stopped when the error on the validation set begins to rise. The penalty term method is another way to reduce overfitting. The basic approach involves adding penalty terms to the usual error function in order to constrain the search and cause weights to differentially decay.

Actually, stopped training and penalty term methods are two widely used categories. The detailed penalty machines and penalty terms are presented in the following section.

## **Penalty Mechanism and Algorithm**

Penalty Function Methods: Usually, penalty function methods are used in determining a solution of a constrained nonlinear programming problem [10]. Currently, there is not a universally accepted method of dealing with such a problem. A penalty function method is to replace a constrained problem with one that is unconstrained. The

latter problem is then solved using an iterative technique. A general penalty function method, a barrier penalty function method and a quadratic penalty function method are introduced in the following sections.

In penalty function methods, the constrained problem is converted into an unconstrained problem by adding a penalty function,  $p(x)$ , to the objective function  $f(x)$ . The resulting unconstrained objective function has the form  $f(x) + \beta p(x)$ , where  $\beta > 0$ . The function  $p(x)$  imposes a penalty of  $\beta p(x)$  whenever  $x$  does not satisfy the constraints of the original problem. Actually, a sequence  $\{ f(x) + \beta p(x) \}$  of functions are minimized (or maximized). The solution,  $\{x_k\}$ , of the sequence will usually approach the solution of the original problem. Normally, each  $x_k$  is not a feasible solution of the original problem. The process terminates whenever the required accuracy has been obtained, or whenever some solution,  $x_k$ , is generated that is a feasible solution of the original problem. In a penalty function method, an expression involving the constraints is added to the objective function. The expression is selected so that the value of the updated objective function is excessively high (or low) at a point  $x$  where the problem is infeasible.

In general, one penalty function for the problem (2-1) is function (2-2)

$$\begin{aligned} & \text{Minimize } f(x) \\ & \text{subject to} \\ & g_i(x) = b_i \text{ for } i = 1, \dots, l \\ & g_i(x) \leq b_i \text{ for } i = l+1, \dots, m \end{aligned} \tag{2-1}$$

$$p(x) = \sum_{i=1}^l |b_i - g_i(x)|^k + \sum_{i=l+1}^m (\max\{0, g_i(x) - b_i\})^k \quad (2-2)$$

where  $k$  is a natural number. Notice that  $p(x) \geq 0$ . In fact  $p(x) = 0$  if and only if  $x$  is feasible.

Problem (2-1) could be converted into the form

Minimize  $f(x)$

subject to (2-3)

$$h_i(x) = 0 \text{ for } i = 1, \dots, m$$

by adding the square of an unrestricted variable to the left side of each inequality constraint, and then moving each  $b_i$  to the left side of each constraint. A typical penalty function for (2-2) is

$$p(x) = \sum_{i=1}^m |h_i(x)|^k \quad (2-4)$$

where  $k$  is a (usually even) natural number. Again notice that  $p(x) \geq 0$ . The remainder of this section deals with problem (2-3).

Barrier function methods: A Barrier function method is an improved penalty function method. Again a sequence of functions  $\{f(x) + (1/\beta_k)b(x)\}$  is minimized (or maximized) and the sequence of solutions  $\{x_k\}$  normally tends to a solution of the original problem. The difference in barrier function is that the solutions,  $x_k$ , are all

feasible solutions of the original problem. The function  $b(x)$  is called a barrier function because it imposes a penalty near the boundary of the set of feasible solutions of the original problem.

For the problem:

$$\text{Minimize } f(x) \text{ subject to } g_i(x) \leq 0 \text{ for } i = 1, \dots, m \quad (2-5)$$

Notice that problem (2-5) does not contain any equality constraints. Barrier function methods are similar to penalty function methods in that a barrier function is added to the objective function, and the resulting function is minimized. The difference is that the solutions are interior points of  $F$  (rather than points exterior to  $F$ ). The purpose of the barrier function is to prevent the solutions from leaving the interior of  $F$ .

Some common barrier functions for Problem (2-5) are

$$b(x) = -\sum_{i=1}^m \frac{1}{g_i(x)} \quad (2-6)$$

and

$$b(x) = \sum_{i=1}^m \ln|g_i(x)| \quad (2-7)$$

Notice that  $b(x)$  is, in either case, continuous throughout the interior of  $F$ . Moreover,  $b(x) \rightarrow \infty$  as  $x$  approaches the boundary of  $F$  via the interior of  $F$ . Rather than solve (2-5), we intend to solve the following problem:

$$\text{Minimize } f(x) + \frac{1}{\beta} b(x) \text{ subject to each } g_i(x) < 0 \quad (2-8)$$

where  $\beta > 0$ .

The Quadratic penalty function method: Both penalty function and barrier function methods can possess the undesirable property of slow convergence. In [25], the penalty function method is modified using Lagrange multipliers to obtain a more efficient method. The technique is called the method of multipliers and has emerged as an important tool for solving constrained nonlinear programming problems. The quadratic penalty function method is one of these methods. It is briefly introduced as following.

For the problem

$$\text{Minimize } f(x) \text{ subject to } h_i(x) = 0 \quad i = 1, \dots, m \quad (2-9)$$

where  $f, h_1, \dots, h_m$  are continuously differentiable, assume that the set,  $F$ , of feasible solutions of (2-9) is nonempty. The continuity of the  $h_i$  ensures that  $F$  is closed. As mentioned in [10], the Weierstrass theorem guarantees the existence of a solution,  $x^*$ , of problem (2-9).

In [10], a method for determining  $x^*$  was suggested, Namely, compute vectors  $x^*$  and  $\lambda^*$  that satisfy

$$0 = \frac{\partial L}{\partial x}(x^*, \lambda^*) = \nabla f(x^*)^T + \lambda^{*T} \frac{\partial h}{\partial x}(x^*)$$

and

$$0 = \frac{\partial L}{\partial \lambda}(x^*, \lambda^*) = h(x^*)$$

(2-10)



where  $L(x, \lambda) = f(x) + \lambda^T h(x)$  and  $h(x) = [h_1(x) \dots h_m(x)]^T$ . Unfortunately, the system of equations (2-10) is difficult to solve.

Consider a solution  $x^*$  of (2-9). Let  $\lambda^*$  be the corresponding vector of Lagrange multipliers for which equations (2-10) hold. Notice that whenever  $x \in F$ , then

$$L(x^*, \lambda^*) = f(x^*) \leq f(x) = f(x) + \lambda^{*T} h(x) = L(x, \lambda^*)$$

$$\text{Thus, } \min \{L(x, \lambda^*) : x \in F\} = L(x^*, \lambda^*) \text{ and} \tag{2-11}$$

$$\min \{f(x) : x \in F\} = \min \{L(x, \lambda^*) : x \in F\}$$

This suggests that rather than solve (2-9), we could solve the problem on the right side of (2-11), possibly using a penalty function method. That is

$$\text{Minimize } f(x) + \lambda^{*T} h(x) + \frac{\beta}{2} \sum_{i=1}^m (h_i(x))^2 \tag{2-12}$$

where  $\beta > 0$ . Of course the problem is that  $\lambda^*$  is not known at the onset of the problem. The next result suggests an alternate strategy consisting of solving a sequence of problems of the form

$$\text{Minimize } f(x) + \lambda_k^T h(x) + \frac{\beta}{2} \sum_{i=1}^m (h_i(x))^2 \tag{2-13}$$

where  $\lambda_k \in \mathbb{R}^{m \times 1}$ .

The above discussions concern the penalty function methods and the penalty mechanism. They have some similarity with the penalty term method used in neural network training and can be used to evaluate the penalty terms and penalty mechanism used in neural network training.

To evaluate the different penalty terms developed in neural network training, a summary of different penalty terms is presented in the following.

### **Penalty Term Method of Reducing Overfitting**

#### **A. Weigend et al Penalty Term**

Weigend et al. [11]-[13] suggested the following cost function:

$$\sum_{k \in T} (t_k - o_k)^2 + \lambda \sum_{i \in C} \frac{w_i^2/w_i^2}{1 + w_i^2/w_o^2} \quad (2-14)$$

where C is the set of all connections and T is the set of training patterns. The second term is the penalty term that represents the complexity of the network as a function of the weight magnitudes relative to the constant  $w_o$ . if  $|w_i| \gg w_o$ , then the cost of a weight will approaches  $\lambda$ . If  $|w_i| \ll w_o$ , the cost is close to zero. The value of  $\lambda$  depends on the problem. If it is too small, it won't have any significant effect; if it is too large, all the weights will be driven to zero.

#### **B. Chauvin Penalty Term**

In [14], Chauvin minimize the cost function

$$C = \mu_{\sigma} \sum_j^P \sum_i^O (d_{ij} - o_{ip})^2 + \mu_{en} \sum_j^P \sum_i^H e(o_{ij}^2) \quad (2-15)$$

where  $e$  is a positive monotonic function. The sums are over the set of output units  $O$ , the set of patterns  $P$ , and the set of hidden units  $H$ . The first term is the normal back-propagation error term, the second term measures the average “energy” expended by the hidden units. The parameters  $\mu_{\sigma}$  and  $\mu_{en}$  balance the two terms. The “energy” expended by a unit--how much its activity varies over the training patterns--is an indication of its importance. If the unit changes a lot, it probably encodes significant information; if it does not change much, it probably does not carry much information.

A magnitude-of-weights term may also be added to the cost function, giving

$$C = \mu_{\sigma} \sum_j^P \sum_i^O (d_{ij} - o_{ip})^2 + \mu_{en} \sum_j^P \sum_i^H e(o_{ij}^2) + \mu_w \sum_{ij}^W w_{ij}^2 \quad (2-16)$$

Since the derivative of the third term with respect to  $w_{ij}$  is  $2 \mu_w w_{ij}$ , this effectively introduces a weight-decay term into the back-propagation equations. Weights that are not essential to the solution decay to zero and can be removed.

### C. Ji Penalty Term

Ji et al. [18] modify the error function to minimize the number of hidden nodes and the magnitudes of the weights. A single-hidden-layer network with one input node and one linear output node is investigated in their research. Beginning with a network having more hidden units than necessary, the output is computed as

$$g(x, w, \theta) = \sum_{i=1}^N v_i f(u_i x - \theta_i) \quad (2-17)$$

where  $\theta_i$  is the threshold,  $f$  is the sigmoid function  $1 / (1 + e^{-x})$ , and  $u$  and  $v$  are the input and output weights of  $i$ th hidden unit respectively.

The significance of a hidden unit is computed based on its input and output weights

$$s_i = \sigma(u_i) \sigma(v_i) \quad (2-18)$$

where  $\sigma(w) = w^2 / (1 + w^2)$ .

The error is defined as the sum of  $\varepsilon_0$ , the normal sum of squared errors, and  $\varepsilon_1$ , term measuring node significance.

$$\begin{aligned} \varepsilon(w, \theta) &= \eta \varepsilon_0(w, \theta) + \lambda \varepsilon_1(w) \\ &= \eta \sum_{\pi=1}^M [g(x^\pi; w, \theta) - y^\pi]^2 + \lambda \sum_{i=1}^N \sum_{j=1}^{i-1} s_i s_j \end{aligned} \quad (2-19)$$

where  $\pi$  indexes the training patterns and  $x^\pi$  and  $y^\pi$  are the input and desired output for pattern  $\pi$ , and  $\mu$  and  $\lambda$  are learning rate parameters. The  $\varepsilon_1(w)$  term makes the algorithm favor solutions with fewer significant hidden units.

It is suggested the second term be added only after the network has learned the training set sufficiently well because conflict between the two error terms may cause local minim.

#### D. Bishop Penalty Term

Chris M. Bishop [16] proposed another penalty term. For error function (2-20), the penalty term is given by (2-21).

$$E = E^s + \lambda E^c \quad (2-20)$$

$$E^c = \frac{1}{2P} \sum_{p=1}^P \sum_{l=1}^L \sum_{n=1}^N \left( \frac{\partial^2 y_{np}}{\partial x_{lp}^2} \right)^2 \quad (2-21)$$

Where  $y_n$  and  $x_l$  denote the components of  $y$  and  $x$ , respectively, and the parameter  $\lambda$  controls the degree of smoothness of the network mapping. Bishop indicated: “Unfortunately, the optimum value for  $\lambda$  is problem dependent. It may be found by seeking the minimum error with respect to a cross-validation data set, or by a variety of techniques based on the statistical properties of the training data.” [16].

#### E. Simple Penalty Terms

Ishikawa [15] proposed another simple cost function

$$C = \sum_{k \in T} (t_k - o_k)^2 + \lambda \sum_{i,j} |w_{ij}| \quad (2-22)$$

If  $w_{ij} > 0$ , the weight is decremented by  $\lambda$ , otherwise, if  $w_{ij} < 0$ , then it is incremented by  $\lambda$ .

Russel Reed [6] described the following simple cost function:

$$C = \sum_{k \in T} (t_k - o_k)^2 + \lambda \sum_{i,j} w_{ij}^2 \quad (2-23)$$

Russel Reed evaluated the simple penalty term: “One of the characters of the  $\lambda w_{ij}$  penalty term is that it tends to favor vector with many small components over ones with a single large component, even when this is an effective choice.”

A constant  $\lambda$  is used in most of the penalty terms. There is no criteria to select a  $\lambda$ . Weigend et al. [11] indicated that “the value of  $\lambda$  requires some tuning and depends on the problem. If it is too small, it won’t have any significant effect; if it is too large, all the weights will be driven to zero.” Bishop [19] indicated that the optimum value for  $\lambda$  will be problem dependent, and may be found by seeking the minimum error with respect to a cross-validation data set, or by a variety of techniques based on the statistical properties of the training data. Ji et al. [18] suggested that the  $\lambda$  can be made a function of  $\epsilon_0$  such as  $\lambda = \lambda_0 e^{-\phi \epsilon_0}$ . They suggested the second penalty term be added only after the network has learned the training set sufficiently well, because of the conflict between the two error terms may cause local minimal. Ping Jiang [24] said, “the optimum point of  $\lambda$  is network architecture dependent. We need to choose  $\lambda$  to close to optimum point to improve the generalization performance.”

**Chapter III**  
**ARTIFICIAL NEURAL NETWORK ARCHTECTURE AND LEARNING**  
**ALGORITHMS**

**Architectures of Feedforward Artificial Neural Networks**

Some artificial neural networks were introduced in Chapter I. This thesis focuses on the most widely used multilayer feedforward networks. The architecture of a multilayer feedforward network is shown as Figure 3-1. This type of network arranges neurons in layers. All neurons in a layer are connected to all neurons in the adjacent layers through unidirectional links. These links are represented by synaptic weights. The input layer of the network is treated as connection nodes. All the layers except the output layer of the network are hidden layers. So the number of hidden layers is the number of layers in a network minus one.

The notations used are shown in Figure 3-1. All neurons in a layer are consecutively indexed starting from 1, in a top-down fashion. The layers are indexed in a left-to-right order and are identified by square-bracketed superscripts. All inputs to a neuron in layer  $k$  are denoted as  $a_i^{[k-1]}$ , where  $i = 0, 1, 2, \dots, S_{k-1}$  ( $S_{k-1}$  is the number of neurons in the  $(k-1)$ th layer). In the case of  $k-1 = 0$ ,  $a_i^{[0]}$  are the inputs of the network. For each layer, we assumed an extra bias node that has a constant output value of  $-1$ , i.e.

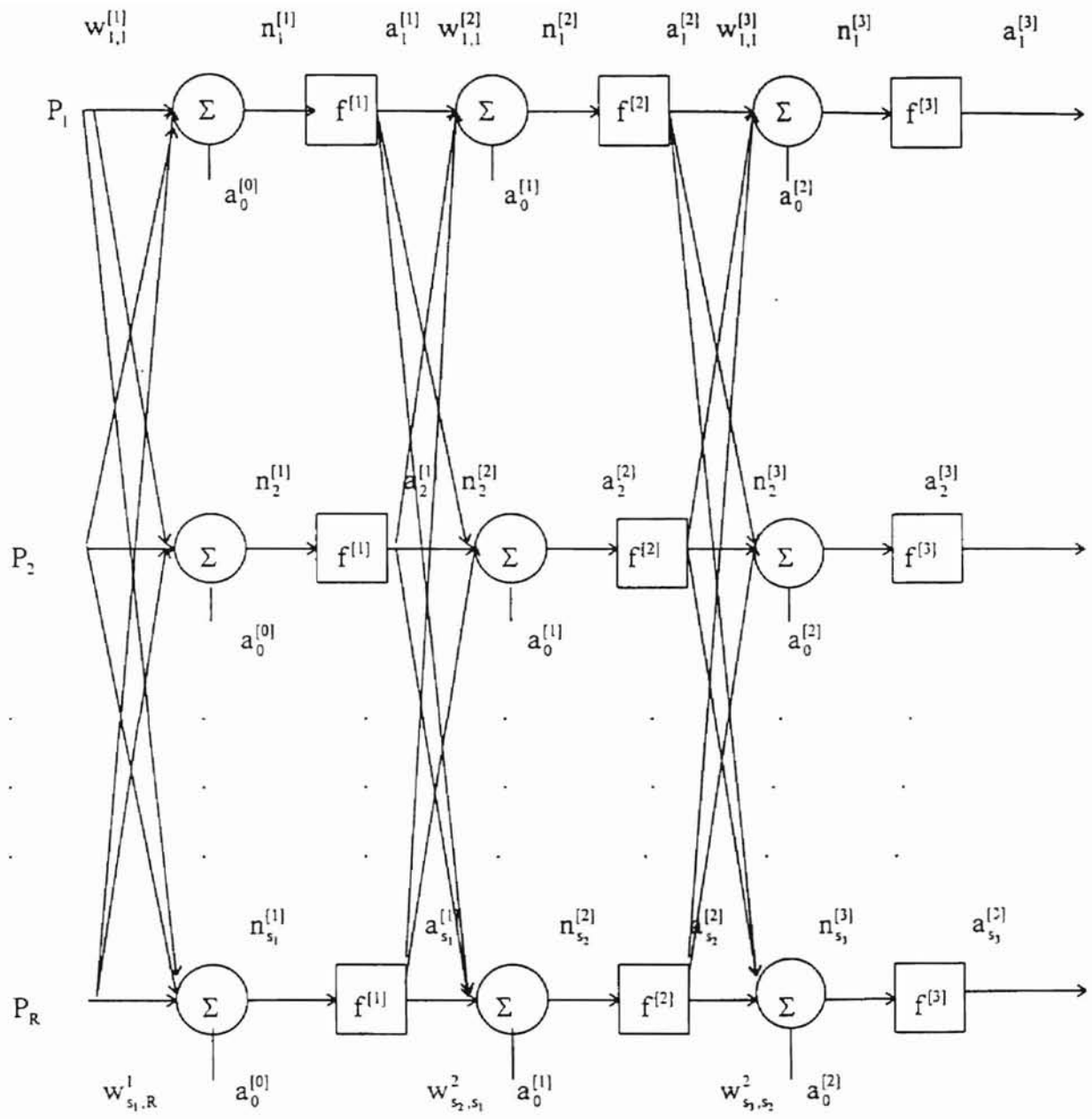


Figure 3-1 A Three Layer Feedforward Network



$a_0^{[k]} = -1$ . Notice that for each  $k > 2$ ,  $a_i^{[k-1]}$  is also the output of neuron  $i$  in  $(k-1)$ th layer. The outputs in the  $k$ th layer of the network can be written in vector form as  $\mathbf{a}^{[k]}$ . A weight is represented as  $w_{j,i}^{[k]}$ , where  $k$  is the layer index and “ $j,i$ ” means that the weight is the connection from the  $i$ th neuron in layer  $k-1$  to the  $j$ th neuron in layer  $k$ . In vector form, weights can be represented by  $\mathbf{w}^{[k]} = (\mathbf{w}_{ji}^{[k]})^T$ . The  $n_j^{[k]}$  represents the weighted sum of a neuron  $j$  in layer  $k$ . The weighted sum of the inputs of a neuron  $j$  in layer  $k$  can be expressed as

$$n_j^{[k]} = \sum_{i=0}^{n_{k-1}} w_{ji}^{[k]} \cdot a_i^{[k-1]} \quad (3-1)$$

The output of the neuron  $j$  in layer  $k$  can be expressed as

$$a_j^{[k]} = f_j^{[k]}(n_j^{[k]}) \quad j = 1, 2, \dots, n_k \quad (3-2)$$

Where  $f_j^{[k]}$  is the activation function of the neuron. We will discuss the activation function in the following section. In vector form, the formulas can be written as

$$\mathbf{n}^{[k]} = (\mathbf{w}^{[k]})^T \mathbf{a}^{[k-1]} \quad (3-3)$$

$$\mathbf{a}^{[k]} = \mathbf{f}^{[k]}(\mathbf{n}^{[k]}) \quad (3-4)$$

where  $\mathbf{f}^{[k]} = (f_j^{[k]})^T$  is a vector of the activation function.

## Activation Function

The original activation function is a binary function [18]. This limits the application of perceptron neural networks to classification problems only. In order to solve a general type of mapping application problem, we need to use nonlinear continuous activation functions. There are many nonlinear activation functions that can be used in multilayer networks as long as the functions are differentiable. The most commonly used functions are the sigmoid function and the hyperbolic function which are expressed as

$$\text{Sigmoid function } f(x) = \frac{1}{1 + e^{-x}} \quad (3-5)$$

$$\text{Hyperbolic function } f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3-6)$$

The graphs of sigmoid and hyperbolic functions are shown in Figure 3-2 and 3-3. Since we can always scale down the input and output values to the interval (0,1) or (-1,1), there is no significant difference between the two functions. The sigmoid function is used in this paper.

## Weights in the Neural Network

The weights in a neural network are initially chosen to be small random numbers. An activation function is active only in a small domain interval as shown in Figure 3-2. If the initial weights are too large, the activation functions may saturate at the beginning of

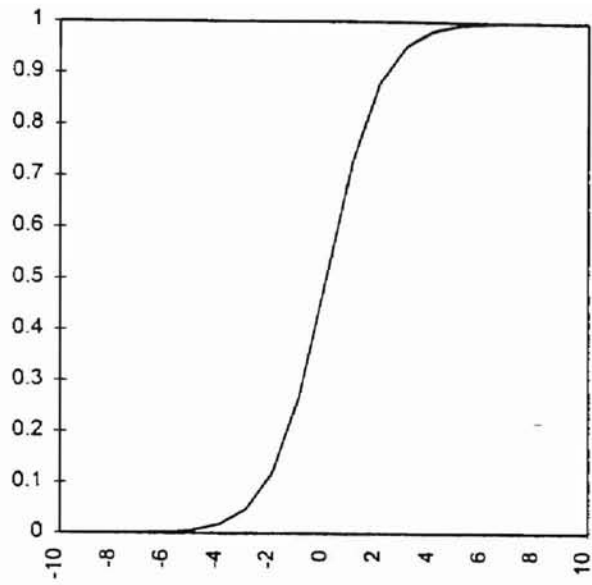


Figure 3-2 The Sigmoid Function

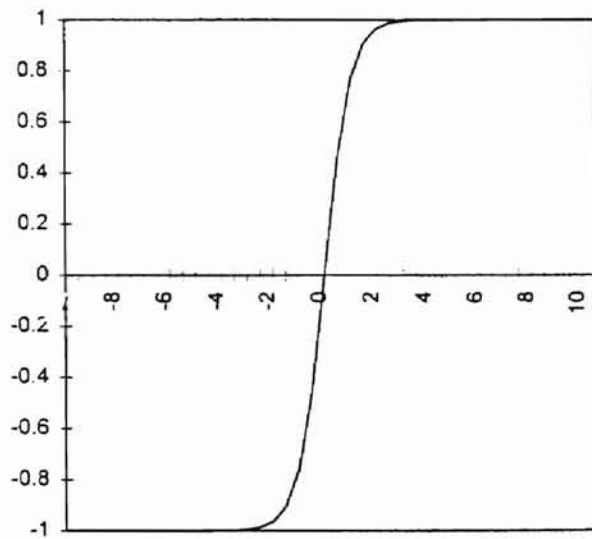


Figure 3-3 The Hyperbolic Function

the training and the network is prone to get stuck in a local minimum near the starting point [19]. In this paper, the initial weights of all neural networks are chosen as random numbers uniformly distributed between  $\frac{-0.5}{\text{fan-in of that node}}$  and  $\frac{0.5}{\text{fan-in of that node}}$  [20], where the fan-in of that node is the number of inputs including bias input to that node.

### Optimization Algorithm

From an optimization point of view, training a network is equivalent to minimizing a global error function, which is a multivariate function that depends on the weights in the network. In this paper we use the Conjugate Gradient Optimization Method. The method is introduced simply as shown below.

The Conjugate Gradient Method searches the minimum in the conjugate direction to guarantee the quadratic termination. Suppose that we want to minimize the following function:

$$F(\mathbf{x}) = 1/2\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c \quad (3-7)$$

From the Taylor series we know that the first order necessary condition for  $\mathbf{x}^*$  is equal to zero, i.e.

$$\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} = 0 \quad (3-8)$$

Any point that satisfies the above equation is called a stationary point. Even though the above equation is satisfied, there is no guarantee that the local minimum is reached. The second order necessary condition for a strong minimum is that the Hessian matrix to be semidefinite. Sufficient conditions for a strong minimum to exist require the Hessian matrix to be positive definitely.

The conjugate gradient method is to search the minimum in the conjugate direction to guarantee the quadratic termination. The conjugate direction is defined as follows:

A set of vectors  $\{P_k\}$  is mutually conjugate with respect to a positive definite Hessian Matrix A if and only if

$$P_k^T A P_j = 0 \quad k \neq j \quad (3-9)$$

Many vectors that satisfies (3-9). One set consists of the eigenvalues of A.

It can be shown [21] that if we make a sequence of exact linear searches along any set of conjugate directions  $\{p_1, p_2, \dots, p_n\}$ , then the exact minimum of any quadratic function with n parameters, will be reached in, at most, m searches. Recall that for a quadratic function, the gradient is

$$\nabla F(x) = Ax + d \quad (3-10)$$

If we calculate the change in the gradient at iteration k+1, we have

$$\Delta \mathbf{g}^{(k)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} = (\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{d}) - (\mathbf{A}\mathbf{x}^{(k)} + \mathbf{d}) = \mathbf{A}\Delta \mathbf{x}^{(k)} \quad (3-11)$$

Based on the Steepest Descent Method [21], we have

$$\Delta \mathbf{x}^{(k)} = (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha^{(k)} \mathbf{p}^{(k)} \quad (3-12)$$

where  $\alpha^{(k)}$  is chosen to minimize  $F(\mathbf{x})$  in the direction  $\mathbf{p}^{(k)}$ .

We can now restate the conjugate conditions by substituting (3-10) and (3-11) to (3-9).

$$\alpha^{(k)} \mathbf{p}^{(k)\top} \mathbf{A} \mathbf{p}^{(j)} = \Delta \mathbf{x}^{(k)\top} \mathbf{A} \mathbf{p}^{(j)} = \Delta \mathbf{g}^{(k)\top} \mathbf{p}^{(j)} = 0 \quad (3-13)$$

Usually we use steepest descent method to begin the search, i.e.

$$\mathbf{p}^{(1)} = -\mathbf{g}^{(1)} \quad (3-14)$$

Then at each iteration we need to construct a vector  $\mathbf{p}^{(k)}$  which is orthogonal to  $\{\Delta \mathbf{g}^{(1)}, \Delta \mathbf{g}^{(2)}, \dots, \Delta \mathbf{g}^{(k)}\}$ . It can be simplified [21] by for following form

$$\mathbf{p}^{(k)} = -\mathbf{g}^{(k)} + \beta^{(k)} \mathbf{p}^{(k-1)} \quad (3-15)$$

The  $\beta^{(k)}$  can be chosen by several different methods, which will produce equivalent results for quadratic functions. One of the most common choice [21] is

$$\beta^{(k)} = \frac{\Delta \mathbf{g}^{(k-1)\top} \mathbf{g}^{(k)}}{\Delta \mathbf{g}^{(k-1)\top} \mathbf{p}^{(k)}} \quad (3-16)$$

The algorithm is as follows:

Algorithm 3-1. The algorithm for the conjugate gradient method is as follows:

1. Set  $k = 1$ , guess  $\mathbf{x}^{(1)}$ ;
2. Select the first search direction according to the steepest descent method, i.e.

$$\mathbf{p}^{(1)} = -\mathbf{g}^{(1)}$$

3. Calculate  $\mathbf{g}^{(k)}$

$$\mathbf{g}^{(k)} = \nabla \mathbf{F}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(k)}}$$

4. Calculate  $\beta^{(k)}$  according to (3-16);

5. Calculate  $\mathbf{p}^{(k)}$  according to (3-15);

6. Calculate  $\Delta \mathbf{x}^{(k)}$  according to (3-12), i.e.

$$\Delta \mathbf{x}^{(k)} = (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha^{(k)} \mathbf{p}^{(k)}$$

7. Calculate  $\mathbf{x}^{(k+1)}$  as the following

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta \mathbf{x}^{(k)}$$

8. If  $\mathbf{x}^{(k+1)}$  satisfies the convergence criteria, stop. Otherwise,

9. Go to step 3.

### Forward Computations

As we know from Chapter I, the neural network learning process includes two phases: forward computation and backward computation. During forward computation, a set of input data is given to the neurons in the first layer (input layer). These neurons are activated and pass the results to neurons to next layer. The process continues until the output layer is reached and the outputs of the network have been calculated. The process can be summarized as follows:

### Algorithm 3-2. Forward algorithm

1. Given input vector  $\mathbf{x}$ , set  $\mathbf{n}^0 = \mathbf{x}$ ;
2. The weight matrix and activation function  $\mathbf{f}^{[k]}$ ,  $k=1,2,\dots,K$  are known, where  $k$  is the number of layers in the network;
3. Compute  $\mathbf{n}^{[k]} = (\mathbf{w}^{[k]})^T \mathbf{a}^{[k-1]}$  and  $\mathbf{a}^{[k]} = \mathbf{f}^{[k]}(\mathbf{n}^{[k]})$  for  $k=1,2,\dots,K$ ;
4.  $\mathbf{a}^{[k]}$  is the output of the network;

### Backpropagation Computation

We have discussed the forward computation in feedforward artificial neural networks in the last section. We will now formulate the backpropagation computation for feedforward artificial networks. We know that a feedforward artificial neural network changes its behavior (weights) dynamically during the training session. The error made by the network during training is measured by a predefined function called the error function (performance) [22], cost function, or energy function [23]. The error function is used to calculate errors and the distribution of errors among all neurons of a network. Then the connection weights are changed to reduce the error of the network. This dynamic adaptation of weights ends when the error is within a tolerance limit at an optimum point with respect to some optimization criterion. Considering a neural network of  $K$  layers, the general performance function can be shown as:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^Q \left( \left( \mathbf{f}^{[k]}(\mathbf{p}_i, \mathbf{w}) - \mathbf{t}_i \right)^T \left( \mathbf{f}^{[k]}(\mathbf{p}_i, \mathbf{w}) - \mathbf{t}_i \right) + \lambda \mathbf{P}_{ij} \right) \quad (3-17)$$

The first term is the performance function (error function). The second term is the penalty term. It could be the Weigend penalty term [11], Charvin penalty term [14], Ji



penalty term [18] or some other form.  $Q$  is the number of input/output samples.  $p_i$  is the  $i$ th input data,  $t_i$  is the desired  $i$ th output, and  $w_0$  are constants that are adjusted during training. Because the differentiation is additive, it is convenient to consider one input/output sample  $I$ . In practice, this is used for on-line training [22]. Summation over the entire input/output samples constitutes off-line training [22]. So we have

$$E_i = \frac{1}{2} \left( \left( f^{[k]}(p_i, w) - t_i \right)^2 + \lambda P \right) \quad (3-18)$$

To calculate the gradient element  $g_{ji}$ , we take the derivative of  $E_i$  with respect to  $w_{ji}^{[k]}$  and using the chain rule, we have

$$g_{ji}^{[k]} = \frac{\partial E_i}{\partial w_{ji}^{[k]}} = \frac{\partial E_i}{\partial n_{ji}^{[k]}} \cdot \frac{\partial n_{ji}^{[k]}}{\partial w_{ji}^{[k]}} + p_{ji} \quad (3-19)$$

where  $p_{ji}$  is an element of the penalty term and is defined as

$$P_{ji} = \lambda \frac{w_{ji}^{[k]} w_0^2}{\left( (w_{ji}^{[k]}) + w_0^2 \right)^2} \quad (3-20)$$

or

$$p_{ji} = \lambda w_{ji}^2 \quad (3-20)'$$

For Bishop's penalty term  $P = \frac{1}{2} \sum_{l=1}^L \sum_{n=1}^N \left( \frac{\partial^2 y_n}{\partial x_l^2} \right)^2$ ,  $\frac{\partial^2 y_n}{\partial x_l^2}$  can be expressed as

$$\begin{aligned}\frac{\partial^2 y_n}{\partial x_l^2} &= \frac{(y(x_l+h) - y(x_l)) - (y(x_l) - y(x_l-h))}{h^2} \\ &= \frac{y(x_l+h) + y(x_l-h) - 2y(x_l)}{h^2}\end{aligned}\quad (3-20)''$$

The  $p_{ij}$  can be shown as :

$$\begin{aligned}p_{ij} &= \frac{\partial P}{\partial w_{ij}} \\ &= \sum_{n=1}^N \sum_{l=1}^L \frac{y(x_l+h) + y(x_l-h) - 2y(x_l)}{h^2} \\ &\quad * \frac{1}{h^2} \left( \frac{\partial y(x_l+h)}{\partial w_{ij}} + \frac{\partial y(x_l-h)}{\partial w_{ij}} - 2 \frac{\partial y(x_l)}{\partial w_{ij}} \right)\end{aligned}\quad (3-20)'''$$

From (3-1), we have

$$\frac{\partial n_i^{[k]}}{\partial w_{ji}^{[k]}} = a_i^{[k-1]}\quad (3-21)$$

Here we define  $s_j$  as:

$$s_j^{[k]} \equiv \frac{\partial E_i}{\partial w_{ji}^{[k]}} = s_j^{[k-1]} \cdot a_i^{[k-1]} + p_{ji}\quad (3-22)$$

where  $s_j$  is the sensitivity of  $E_i$  to change in the  $j$ th element of the net input at layer  $[k]$ .

Then (3-19) becomes

$$g_j^{[k]} \equiv \frac{\partial E_i}{\partial w_{ji}^{[k]}} = s_j^{[k-1]} \cdot a_i^{[k-1]} + p_{ji}\quad (3-23)$$

Using the Jacobian matrix [24], we can derive the recurrence relationship for the sensitivities.

$$\frac{\partial \mathbf{n}^{[k+1]}}{\partial \mathbf{n}^{[k]}} \equiv \begin{bmatrix} \frac{\partial \mathbf{n}_1^{[k+1]}}{\partial \mathbf{n}_1^{[k]}} & \frac{\partial \mathbf{n}_1^{[k+1]}}{\partial \mathbf{n}_2^{[k]}} & \dots & \dots & \frac{\partial \mathbf{n}_1^{[k+1]}}{\partial \mathbf{n}_{S_k}^{[k]}} \\ \frac{\partial \mathbf{n}_2^{[k+1]}}{\partial \mathbf{n}_1^{[k]}} & \frac{\partial \mathbf{n}_2^{[k+1]}}{\partial \mathbf{n}_2^{[k]}} & \dots & \dots & \frac{\partial \mathbf{n}_2^{[k+1]}}{\partial \mathbf{n}_{S_k}^{[k]}} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \frac{\partial \mathbf{n}_{S_{k+1}}^{[k+1]}}{\partial \mathbf{n}_1^{[k]}} & \frac{\partial \mathbf{n}_{S_{k+1}}^{[k+1]}}{\partial \mathbf{n}_2^{[k]}} & \dots & \dots & \frac{\partial \mathbf{n}_{S_{k+1}}^{[k+1]}}{\partial \mathbf{n}_{S_k}^{[k]}} \end{bmatrix} \quad (3-24)$$

the element  $ij$  in (3-24) can be shown as:

$$\begin{aligned} \frac{\partial \mathbf{n}_i^{[k+1]}}{\partial \mathbf{n}_j^{[k]}} &= \frac{\partial \left( \sum_{l=1}^S \mathbf{w}_{il}^{[k+1]} \mathbf{a}_l^{[k]} \right)}{\partial \mathbf{n}_j^{[k]}} = \mathbf{w}_{ji}^{[k+1]} \frac{\partial \mathbf{a}_j^{[k]}}{\partial \mathbf{n}_j^{[k]}} \\ &= \mathbf{w}_{ij}^{[k+1]} \frac{\partial \mathbf{f}^{(k)}(\mathbf{n}_j^{[k]})}{\partial \mathbf{n}_j^{[k]}} = \mathbf{w}_{ij}^{[k+1]} \mathbf{f}^{(k)}(\mathbf{n}_j^{[k]}) \end{aligned} \quad (3-25)$$

where

$$\mathbf{f}^{(k)}(\mathbf{n}_j^{[k+1]}) = \frac{\partial \mathcal{F}^{(k)}(\mathbf{n}_j^{[k]})}{\partial \mathbf{n}_j^{[k]}} \quad (3-26)$$

So the Jacobian matrix can be written as

$$\frac{\partial \mathbf{n}^{[k+1]}}{\partial \mathbf{n}^{[k]}} = \mathbf{W}^{[k+1]} \cdot \mathbf{F}(\mathbf{n}^{[k]}) \quad (3-27)$$

where

$$\mathbf{F}^{[k]}(\mathbf{n}^{[k]}) = \begin{bmatrix} f^{[k]}(n_1^{[k]}) & 0 & \dots & 0 \\ 0 & f^{[k]}(n_2^{[k]}) & 0 & 0 \\ \cdot & 0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & f^{[k]}(n_s^{[k]}) \end{bmatrix} \quad (3-28)$$

Now the sensitivity recursively in matrix form is seen as:

$$\begin{aligned} \mathbf{s}^{[k]} &= \frac{\partial \mathbf{E}_i}{\partial \mathbf{n}^{[k]}} = \left( \frac{\partial \mathbf{n}^{[k+1]}}{\partial \mathbf{n}^{[k]}} \right) \frac{\partial \mathbf{E}_i}{\partial \mathbf{n}^{[k+1]}} = \mathbf{F}(\mathbf{n}^{[k]}) \cdot (\mathbf{W}^{[k+1]})^T \cdot \frac{\partial \mathbf{E}_i}{\partial \mathbf{n}^{[k+1]}} \\ &= \mathbf{F}(\mathbf{n}^{[k]}) \cdot (\mathbf{W}^{[k+1]}) \cdot \mathbf{s}^{[k+1]} \end{aligned} \quad (3-29)$$

The sensitivities are propagated backward through the network from the last layer to the first layer. The starting point can be obtained from the output layer.

$$s_i^{[k]} = \frac{\partial a_i^{[k]}}{\partial n_i^{[k]}} = -(t_i - a_i) \frac{\partial a_i}{\partial n_i^{[k]}} \quad (3-30)$$

Since

$$\frac{\partial a_i}{\partial n_i^{[k]}} = \frac{\partial a_i^{[k]}}{\partial n_i^{[k]}} = f^{[k]}(n_i^{[k]}) \quad (3-31)$$

$s_j$  can be expressed as

$$s_i^{[k]} = -(t_i - a_i) f^{[k]}(n_i^{[k]}) \quad (3-32)$$

(3-31) has the following matrix form

$$s^{[k]} = -F^{[k]}(n^{[k]})(t-a) \quad (3-33)$$

So we can recursively calculate the sensitivities from the last layer to the first layer. Knowing the sensitivities, we can calculate the gradient according to (3-22).

### Algorithms of Penalty Method and Improved Penalty Method

The penalty method and the improved penalty method used in this research are discussed in the following. The basic approach used in the penalty method involves adding penalty terms to the usual objective function in order to constrain the search and cause weights to differentially decay. By using the penalty method, the neural network generalization error can be reduced [24].

Algorithm 3-3 (penalty method):

Given a set of  $S = \{(p_i, t_i) \mid p_i \text{ is input, } t_i \text{ is desired output of } p_i\}$  of  $d$  training samples, and given a network of  $K$  layers with an input dimension  $u$  and an output dimension of  $v$ .

1. Initialize all weights  $w^{[k]} = (w_{ji}^{[k]})$ ,  $l = 1, 2, \dots, K$  as random numbers uniformly distributed between  $\frac{-0.5}{\text{fan-in of that set}}$  and  $\frac{0.5}{\text{fan-in of that set}}$ . Set  $w_0$ .

2. For each sample  $(x_i, t_i) \in S$ , repeat the following steps:

Initialize  $g^{(k)} = 0$ .

- 2.1 Compute the actual outputs of network according to (3-3) and (3-4) using the weight  $\mathbf{w}^{(k)}$
- 2.2 Calculate the gradient  $\mathbf{g}(\mathbf{x}_i)$  according to (3-3)
- 2.3 Sum up  $\mathbf{g}(\mathbf{x}_i)$ , i.e.,  $\mathbf{g}^{(k)} = \mathbf{g}^{(k)} + \mathbf{g}(\mathbf{x}_i)$
3. If  $k=1$  then set  $\mathbf{p}^{(1)} = \mathbf{r}^{(1)} = -\mathbf{g}^{(1)}$
4. Compute  $\alpha^{(k)}$  using a line search technique [23].
5. Compute  $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$  using step 2 to compute  $\mathbf{g}^{(k+1)}$ .
6. Compute  $\beta^{(k)}$  according to (3-16).
7. Compute  $\mathbf{p}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{p}^{(k)}$ .
8. If all the weights are such that the following convergence criterion is satisfied, then go to 9, otherwise set  $k=k+1$  and go to step 2.

$$\sqrt{\frac{\sum_{i=1}^d (\mathbb{E}(\mathbf{w}^{(k+1)}))}{d}} < \sqrt{\frac{\sum_{i=1}^d (\mathbb{E}(\mathbf{w}^{(k)}))}{d}} < \text{tol}$$

9. Set  $\mathbf{w} = \mathbf{w}^{(k+1)}$  and stop.

Actually the overfitting problem is not exactly a constrained optimization problem because the constrained condition is unknown. There is not a universally accepted method for a constrained nonlinear optimization problem.

Based on the penalty method, an improved penalty method is developed in this research. The main idea is training the network without adding any penalty term. Once the performance function value (RMS) begins to increase, a penalty term is added to the usual error function, and the network training process becomes continuous as same as the penalty method. If the generalization value (RMS) begins to increase again, then stop the

training. A performance function value (RMS) is used as a stopping criteria both in the penalty method and the improved penalty method.

Algorithm 3-4 (Improved penalty method 1):

Given a set of  $S = \{(\mathbf{p}_i, \mathbf{t}_i) \mid \mathbf{p}_i \text{ is input, } \mathbf{t}_i \text{ is desired output of } \mathbf{p}_i\}$  of  $d$  training samples, and given a network of  $K$  layers with an input dimension  $u$  and an output dimension of  $v$ .

1. Initialize all weights  $\mathbf{w}^{[k]} = (w_{ji}^{[k]})$ ,  $l = 1, 2, \dots, K$  as random numbers uniformly distributed between  $\frac{-0.5}{\text{fan-in of that set}}$  and  $\frac{0.5}{\text{fan-in of that set}}$ . Set  $w_0$  and  $\lambda$ .

2. For each sample  $(\mathbf{x}_i, \mathbf{t}_i) \in S$ , repeat the following steps.

Initialize  $\mathbf{g}^{(k)} = 0$ .

2.1 Compute the actual outputs of network according to (3-3) and (3-4) using the weight  $w^{(k)}$

2.2 Calculate the gradient  $\mathbf{g}(\mathbf{x}_i)$  according to (3-3)

2.3 Sum up  $\mathbf{g}(\mathbf{x}_i)$ , i.e.,  $\mathbf{g}^{(k)} = \mathbf{g}^{(k)} + \mathbf{g}(\mathbf{x}_i)$

3. If  $k=1$  then set  $\mathbf{p}^{(1)} = \mathbf{r}^{(1)} = -\mathbf{g}^{(1)}$

4. Compute  $\alpha^{(k)}$  using a line search technique [23].

5. Compute  $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha^{(k)} \mathbf{p}^{(k)}$  using step 2 to compute  $\mathbf{g}^{(k+1)}$ .

6. Compute  $\beta^{(k)}$  according to (3-16).

7. Compute  $\mathbf{p}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)} \mathbf{p}^{(k)}$ .

8. Before setting the value of  $\lambda$ , if all the weights are such that the following convergence criterion is satisfied, then set  $\lambda$ , otherwise set  $k=k+1$  and go to step 2. After

setting the value of  $\lambda$ , if all the weights are such that the following convergence criterion is satisfied, then go to step 9, otherwise set  $k = k+1$  and go to step 2.

$$\sqrt{\frac{\sum_{i=1}^d (E(\mathbf{w}^{(k+1)}))}{d}} < \sqrt{\frac{\sum_{i=1}^d (E(\mathbf{w}^{(k)}))}{d}} < \text{tol}$$

9. Set  $\mathbf{w} = \mathbf{w}^{(k+1)}$  and stop.

Based on algorithm 3-4, one more improved penalty method is given as algorithm 3-5. The main difference between algorithm 3-4 and algorithm 3-5 is that a series of  $\lambda$  is given for a penalty term in algorithm 3-5. The objective function is dynamically changed based on the performance of each different penalty parameter  $\lambda$ .

Algorithm 3-5 (Improved penalty method 2):

Given a set of  $\mathbf{S} = \{(\mathbf{p}_i, \mathbf{t}_i) \mid \mathbf{p}_i \text{ is input, } \mathbf{t}_i \text{ is desired output of } \mathbf{p}_i\}$  of  $d$  training samples, and given a network of  $K$  layers with an input dimension  $u$  and an output dimension of  $v$ .

1. Initialize all weights  $\mathbf{w}^{[l]} = (w_{ji}^{[l]})$ ,  $l = 1, 2, \dots, K$  as random numbers uniformly distributed between  $\frac{-0.5}{\text{fan-in of that set}}$  and  $\frac{0.5}{\text{fan-in of that set}}$ . Set  $w_0$ .

2. Select parameter  $\lambda_i$

3. For each sample  $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{S}$ , repeat the following steps:

Initialize  $\mathbf{g}^{(k)} = 0$ .

3.1 Compute the actual outputs of network according to (3-3) and (3-4), using the weight  $\mathbf{w}^{(k)}$ .

3.2 Calculate the gradient  $\mathbf{g}(\mathbf{x}_i)$  according to (3-3)

3.3 Sum up  $\mathbf{g}(\mathbf{x}_i)$ , i.e.,  $\mathbf{g}^{(k)} = \mathbf{g}^{(k)} + \mathbf{g}(\mathbf{x}_i)$



4. If  $k=1$  then set  $\mathbf{p}^{(1)} = \mathbf{r}^{(1)} = -\mathbf{g}^{(1)}$
5. Compute  $\alpha^{(k)}$  using a line search technique [23].
6. Calculate  $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$ .
7. Compute  $\beta^{(k)}$  based on equation (3-16).
8. Calculate  $\mathbf{p}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{p}^{(k)}$ .
9. Compute  $\sum E(W_{\lambda_i}^{(k+1)})$ . For each  $\lambda_i$ , repeat steps 2 to 9 and obtain  $\sum E(W_{\lambda_i}^{(k+1)})$ .
10. Set  $\sum E(W^{(k+1)}) = \text{Minimum}\{\sum E(W_{\lambda_1}^{(k+1)}), \sum E(W_{\lambda_2}^{(k+1)}), \sum E(W_{\lambda_i}^{(k+1)}), \dots\}$ .

Let  $\mathbf{w}^{(k+1)} = \mathbf{w}_{\lambda_i}^{(k+1)}$ ,  $\mathbf{w}_{\lambda_i}^{(k+1)}$  corresponds to the minimum value of the error. If all the weights are such that the following convergence criterion is satisfied, then go to step 11, otherwise set  $k = k+1$  and go to step 2.

$$\sqrt{\frac{\sum_{i=1}^d (E(\mathbf{w}^{(k+1)}))}{d}} < \sqrt{\frac{\sum_{i=1}^d (E(\mathbf{w}^{(k)}))}{d}} < \text{tol}$$

11. Set  $\mathbf{w} = \mathbf{w}^{(k+1)}$  and stop.

## CHAPTER IV

### METHODS AND IMPLEMENTATION

#### **Neural Network Architecture Design**

To compare the effectiveness of different penalty methods, the performance of three training methods are studied in this research. These methods are the regular learning algorithm without a penalty term, the penalty method with different penalty terms, and the improved penalty method proposed in this research. The performance of each method is calculated using a computer program written in the ANSI Standard FORTRAN 77 language.

A small network is tested first. Then the hidden nodes will be added to the network. When the network becomes larger, the generalization error becomes larger and larger. Usually, the generalization error can be reduced by inducing a penalty term [1]. The improved penalty method proposed in this research has proved to be able to reduce the generalization error significantly.

Three penalty terms are tested in this research. There are many different types of penalty terms used in neural networks to reduce overfitting. Some of them are very complicated. Some of them have a disadvantage in that large weights decay at the same rate as small weights. Some of them include a few of user-dependent parameters. The

three penalty terms which will be tested in this research are  $\lambda \frac{w_{ji}^{l,k1} w_0^2}{((w_{ji}^{l,k1}) + w_0^2)^2}$ ,  $\lambda w_{ji}^2$ , and

$$\frac{1}{2p} \sum_{p=1}^P \sum_{l=1}^L \sum_{n=1}^N \left( \frac{\partial^2 y_{np}}{\partial x_{lp}^2} \right)^2.$$

The performance function value (RMS) is used as the stopping criterion. When to stop the training process is very important for a given problem. Therefore, an optimal stopping point is needed to obtain better generalization performance so that the network has a good generalization performance. This is especially important when a network is overfitting. In this thesis, the sample data are divided into two sets. One is the training set and the other is the validation set. When the network is trained, the generalization performance will be tested at certain numbers of iterations using the validation set.

The weights are initialized with random values which are uniformly distributed between -0.5 and 0.5 [2]. A curve fitting criterion is used to test all the learning algorithms. Two data sets are used. One is the training data and the other one is the validation data. Both of them contain 49 pairs. The training and validation data sets are listed in table A-35 and A-36 respectively. For all the methods tested, the same sample data were used.

In total seven methods are tested. Method A is the regular method and method B and C are penalty term methods based on Algorithm 3-3 with different penalty terms. Method D and E are improved penalty term methods based on Algorithm 3-4 with different penalty terms. Method F is an improved penalty term method based on Algorithm 3-5. Method G is a simplified Bishop's penalty term method.

The overview of the methods tested in this paper is shown in Table 4-1.

Table 4-1 Overview of methods tested

Methods*		Penalty Term
A	R	No
B	P	$\lambda \frac{w_{ji}^{k1} w_0^2}{((w_{ji}^{k1}) + w_0^2)^2}$
C	P	$\lambda w_{ij}^2$
D	NP	$\lambda \frac{w_{ji}^{k1} w_0^2}{((w_{ji}^{k1}) + w_0^2)^2}$
E	NP	$\lambda w_{ij}^2$
F	NP2	$\lambda \frac{w_{ji}^{k1} w_0^2}{((w_{ji}^{k1}) + w_0^2)^2}$
G	P	$\frac{1}{2P} \sum_{p=1}^P \sum_{l=1}^L \sum_{n=1}^N \left( \frac{\partial^2 y_{np}}{\partial x_{lp}^2} \right)^2$

- \* R — Regular method without any penalty term.  
P — Penalty term method  
NP — New penalty term method  
NP2 — New penalty term method 2, the objective function will be changed dynamically.

## Discussion of Test Results

First, a network with two input nodes, seven hidden nodes, and one output node (2/7/1) is tested. The network has 29 weights and methods A, B, C, D, and E are tested. The training and generalization performance of different method is listed in table A-1 through A-9. The training and generalization performance of method A is listed in table A-1. It takes about 11 epochs of training to get the training RMS value of 0.0707211 and generalization RMS value of 0.0724163. The training and generalization performance of method B is listed in table A-2 and A-3. It takes 5 epochs of training to get the training RMS value of 0.0706896 and generalization RMS value of 0.0741628. It is found that method B makes the generalization performance slightly decrease (-2.35%). The performance of method C is listed in table A-4 and A-5. It takes 7 epochs to get the training RMS and generalization RMS value of 0.0725740 and 0.0758244. It makes the generalization error increased by 4.49% and the training error increased by 25.07%. The performance of method D is listed in table A-6 and A-7. It takes about 12 epochs to get the training RMS value of 0.0691364 and generalization RMS value of 0.0698246. It improved the generalization performance by 3.71% and reduced the training error by 2.3%. Comparing with the penalty method (method B), the improved penalty method (method D) improved the generalization and training performance by 6.2% and 2.25% respectively. The performance of method E is listed in table A-8 and A-9. It has the same training and generalization performance as the method A because the penalty term used in this method makes the training error increase.

Next, the network with two input nodes, eight hidden nodes, and an output node (2/8/1) is tested. Similarly, training and generalization performance of each method of A,

B, C, D and E are listed in table A-10 through A-17 respectively. The comparison of the performance of different method is listed in Table 4-3. Method A takes 8 training epochs to get the training RMS value of 0.0730081 and generalization RMS value of 0.0767580. The performance of method B is listed in table A-11 and table A-12. It takes 6 training epochs to get the training RMS value of 0.071706 and generalization RMS value of 0.0751127. It improved the training and generalization performance by 1.81% and 2.19% respectively. The performance of method C is listed in table A-13 and A-14. It takes 10 epochs to get the training RMS value of 0.0718936 and generalization RMS 0.0744065. It improved the training and generalization 1.5% and 3.06% respectively. However, the value of  $\lambda$  should be selected very carefully. Otherwise, it will increase the training and generalization error. The performance of method D is listed in table A-15 and A-16. It takes about 15 training epochs to get the training RMS value of 0.0681585 and generalization RMS value of 0.0669311. It improved the training and generalization performance by 7.11% and 14.68% respectively. Comparing with the penalty term method, it improved the training and generalization performance by 5.2% and 12.22%. The performance of method E is listed in table A-17. It takes 15 epochs to get the training RMS value of 0.0718534 and the generalization RMS value of 0.0737869. It improved the training and generalization performance by 1.6% and 4.03% respectively.

Thirdly, the network with two input nodes, ten hidden nodes, and an output node (2/10/1) is tested. Methods A, B, C, D and E are tested. The performance of different method is listed in table A-18 through A-26. The performance of method A is listed in table A-18. It takes about 12 epochs to get the training RMS value of 0.0823795 and generalization RMS value of 0.0865625. In this case, the network is overfitting. Method

B takes about 13 training epochs to get the training RMS value of 0.0782682 and generalization RMS value of 0.0865625. It improved the training and generalization performance by 5.25% and 6.46% respectively. The performance of method C is listed in table A-21 and A-22. The performance of method D is listed in table A-23 and A-24. It takes about 17 epochs to get the training and generalization RMS value of 0.0753999 and 0.0763246 respectively. It improved training performance by 9.26% and the generalization performance by 13.41%. Comparing with the penalty term method, the improved penalty term method improved the training and generalization performance by 3.8% and 6.53% respectively.

To test the effectiveness of the method F, a series of  $\lambda$  (0.008, 0.006, 0.004, 0.002, 0.001, 0.0006, 0.0001, 0.00006, 0.00004, 0.00001) are tested in a network with two input nodes, seven hidden nodes, and an output node. The performance of training and generalization of different  $\lambda$  is listed in table A-27. The best performance is obtained when  $\lambda$  equals 0.0001. It is helpful to use the improved penalty term method 2 to get the best  $\lambda$  from a set of  $\lambda$  values. Once the  $\lambda$  is selected, the rest of the training process of the improved penalty term method 2 (method F) is as same as the penalty term method. So the weakness of the penalty term method still exists in the improved penalty term method 2.

Finally, the performance of Bishop's penalty term method G is tested. The performances of different networks are listed in table A-28 through table A-34. For the net work with two input nodes, seven hidden nodes, and an output node (2/7/1), Bishop's penalty term method takes 10 training epochs to get the training RMS value of 0.0707547 and generalization RMS value of 0.0724433. The generalization performance is very

close to the generalization performance of method A (Table 4-2). For the network (2/8/1), it takes 13 training epochs to get the training RMS value of 0.0788628 and generalization RMS value of 0.0748189. It improved the generalization performance by 2.5%. For the network with two input nodes, ten hidden nodes, and an output nodes (2/10/1), it takes 8 training epochs to get the training RMS value of 0.0823851 and generalization RMS value of 0.0855824. It improved the generalization performance by 1.13%. The performance comparison of different method for different networks is listed in table 4-2 through tab 4-4.



Table 4-2 Performance Comparison of Different Method  
For the Network with 7 hidden nodes

Method	Epoch	Training RMS	Generalization RMS
A	10	0.0707211	0.0724163
B-1	5	0.0706896	0.0741628
B-2	6	0.0895382	0.0892582
C-1	7	0.0725740	0.0758244
C-2	2	0.0884512	0.0866356
D-1	12	0.0691364	0.0698246
D-2	10	0.0707211	0.0724163
E-1	10	0.0707211	0.0724163
E-2	10	0.0707211	0.0724163
G	10	0.0707547	0.0724433

Table 4-3 Performance Comparison of Different Method  
For the Network with 8 hidden nodes

Method	Epoch	Training RMS	Generalization RMS
A	8	0.0730081	0.0767580
B-1	6	0.0717076	0.0751127
B-2	10	0.0728760	0.0764509
C-1	10	0.0718936	0.0744065
C-2	2	0.0865809	0.0864760
D-1	16	0.0681907	0.0677225
D-2	15	0.0681585	0.0669311
E-1	15	0.0718534	0.0737869
G	13	0.0728628	0.0748189

Table 4-4 Performance Comparison of Different Method  
for Network with 10 hidden nodes

Method	Epoch	Training RMS	Generalization RMS
A	12	0.0823795	0.0865625
B-1	5	0.0878567	0.0872217
B-2	13	0.0782682	0.0813108
C-1	7	0.0823783	0.0855836
C-2	2	0.0853115	0.0865137
D-1	17	0.0753999	0.0763246
D-2	12	0.0823741	0.0855825
E-1	10	0.0823793	0.0855825
E-2	10	0.0823793	0.0855822
G	8	0.0823851	0.0855824

## CHAPTER V

### CONCLUSION

Overfitting is a very important issue in artificial neural networks. Penalty term methods are useful way to reduce overfitting. Seven different training algorithms are studied in this research. The following conclusions can be drawn from this study:

1. Overfitting does exist in artificial neural networks. As the neural network becomes larger, the generalization performance becomes worse. It is better to use the smallest network that fits the data.
2. For a network which is not overfitting, the penalty term method has no significant improvement for training and generalization performance of the network. If the penalty term or the constant  $\lambda$  is not chosen properly, the penalty term method will decrease the performance significantly. On the other side, the improved penalty method can slightly increase the generalization performance of the network if the penalty term and  $\lambda$  are chosen properly. If the penalty term and  $\lambda$  are not chosen properly, the improved penalty term method can also be used to train the network and has no risk to decrease the performance. Usually it is difficult to know if the network is overfitting or not. Therefore, it is better to use the improved penalty term method than to use the penalty term method in any situation.

3. When the network is overfitting, the penalty method can be used to improve the generalization performance of the networks. Compared with the penalty term method, the improved penalty method improves the training and generalization performance more significantly and has no risk to decrease the performance.
4. Penalty term and the constant  $\lambda$  are problem and network architecture dependent. The improved penalty method 2 can be used to choose a  $\lambda$  properly and improve the performance significantly as well.

Future work could be done in several areas as listed below:

1. To investigate the performance of each method, a training data set and a validation set are used in this research. Since the training procedure used in the research can itself lead to some over-fitting to the validation set, the performance of each training method may be confirmed by measuring its performance on a third independent set of data called a test set.
2. A constant  $\lambda$  is used in most of the penalty methods. There is no criteria to select a  $\lambda$ . It is valuable to conduct a method to choose  $\lambda$  to close to the optimum point to improve the generalization performance.
3. Another method that can be investigated is an interactive method in which the designer checks the trained network and decides which nodes to remove. Several heuristics are used to identify units that don't constant output over all training patterns. When a number of nodes have highly correlated responses over all patterns, they can be combined into one node.

## Bibliography

- [1] Hecht-Nielsen, Robert, *Neurocomputing*, Addison-Wesley Publishing Company, 1990.
- [2] Rumelhart, D. and McClelland, J., *Parallel distributed processing: exploitations in the micro-structure of cognition*, Volumes 1 and 2, Cambridge: MIT Press, 1986.
- [3] Kohonen, T., *Self-Organising and Associative Memory* (3rd ed.), Berlin: Springer-Verlag, 1989.
- [4] Albus, J. S., A new approach to manipulator control: cerebellar model articulation control (CMAC). *Trans. ASME, J. of Dynamics Syst., Meas. and Contr.*, 97, 220-227, 1975.
- [5] Dietterich, Tom, Overfitting and Undercomputing in Machine Learning, *ACM Computing Survey*, Vol. 27, No. 3, pp.326-327, Sept. 1995.
- [6] Reed, Russell, Pruning Algorithms-A Survey, *IEEE Transactions on Neural Networks*, Vol. 4, No. 5, 1993.
- [7] Hoerl, Arthur E. and Kennard, Robert W., Applications to Nonorthogonal Problems, *Eechnometrics*, Vol. 12, No. 1, pp. 69-82, Feb.1970.
- [8] Subatai, Ahmad and Tesauro, Gerald, Scaling and Generalization in Neural Networks:A Case Study, *Advances in Neural Information Processing 1*, D.S. Touretzky, Ed. pp. 160-168, 1989.
- [9] Chauvin, Y., Generalization Performance of Overtrained Back-propagation Networks, in *Lecture Notes in Computer Science*, Edited by L. B. Almieda and C.J. Wellekens, Springer-Verlag, 1990.

- [10] Melvyn, W. J., *Mathematical Programming, An Introduction to Optimization*, Marcel Dekker, Inc. 1986.
- [11] Weigend, A.S., Rumelhart, D. E., and Huberman, B. A., Back-propagation, weight-elimination and time series prediction, in Proc. 1990 Connectionist Models Summer School, D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton. Eds., 1990, pp. 105-116.
- [12] Weigend, A. S., Rumelhart, D. E., and Huberman, B. A., Generalization by weight-elimination applied to currency exchange rate prediction, in Proc. Int. Joint Conf. *Neural Networks*, Vol. I (Seattle), 1991, pp. 837-841.
- [13] Weigend, A. S., Rumelhart, D. E., and Huberman, B. A., Generalization by weight-elimination with application to forecasting, in *Advances in Neural Information Processing* (3), R. Lippmann, J. Moody, and D. Touretzky, Eds., 1991, pp. 875-882.
- [14] Chauvin, Y., A back-propagation algorithm with optimal use of hidden units, I *Advances in Neural Information Processing* (1), D.S. Touretzky, Ed. (Denver), 1989, pp. 519-526.
- [15] Ishikawa, M., A structural learning algorithm with forgetting of link weights, Tech. Rep. TR-90-7, *Electrotechnical Lab.*, Tsukuba-City, Japan, 1990.
- [16] Bishop, C. M., Curvature-Driven Smoothing: A Learning Algorithm for Feedforward Networks, *IEEE Transactions on Neural Networks*, Vol. 4, No. 5, September 1993.
- [17] Valiant, L. G., A theory of the learnable, *Commun. ACM*, Vol. 27, No. 11, pp. 1134-1142, 1984.

- [18] McCulloch, W.S. and Pitts, W., A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin on Math. Bio.*, 5, 1943.
- [19] Hagan, Martin T., *Neural Network Design*, Lecture Notes, Oklahoma State University, 1995.
- [20] Cichocki, A. and Unbehauen, R., *Neural Networks for Optimization and Signal Processing*, Wiley, 1993.
- [21] Scales, L. E., *Introduction to Nonlinear Optimization*, New York, Springer-Verlag, 1985.
- [22] Cichocki, A. and Unbehauen, R., *Neural Networks for Optimization and Signal Processing*, Wiley, 1993.
- [23] Chauvin, Y., Dynamic Behavior of Constrained Back-Propagation Networks, in *Advances in Neural Information Processing 2*, D.S. Toretzky, Ed. Pp.642-649, 1989.
- [24] Jiang, P., A Penalty Method to Reduce Overfitting in Artificial Neural Networks, Masters degree thesis, Oklahoma State University, 1996.
- [25] Hestenes, M. R., Multiplier and Gradient Methods, *Journal of Optimization Theory and Applications*, No. 4, pp. 303-320, 1969.

**APPENDIX A**  
**TESTING TABLES**

Table A-1 Performance of Training and Generalization (RMS)  
Method A with 7 hidden nodes

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214704E-00	0.209518E-00	
1	0.812552E-01	0.841985E-01	0.133448E-00
2	0.801644E-01	0.834818E-01	0.109082E-02
3	0.790322E-01	0.825625E-01	0.113216E-02
4	0.779749E-01	0.816252E-01	0.105730E-02
5	0.750445E-01	0.787846E-01	0.293044E-02
6	0.736659E-01	0.774192E-01	0.137859E-02
7	0.722210E-01	0.755748E-01	0.144488E-02
8	0.717990E-01	0.732518E-01	0.422016E-03
9	0.718357E-01	0.732768E-01	0.366718E-04
10	0.707211E-01*	0.724163E-01*	0.111452E-02
11	0.707575E-01	0.724437E-01	0.363737E-04

Table A-2 Performance of Training and Generalization (RMS)  
Method B with 7 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.896909E-01	0.891520E-01	0.125540E-00
2	0.895401E-01	0.892551E-01	0.150718E-03
3	0.895380E-01	0.892587E-01	0.218302E-05
4	0.895385E-01	0.892578E-01	0.506639E-06
5	0.895382E-01	0.892583E-01	0.275671E-06
6	0.895382E-01*	0.892582E-01*	0.447035E-07
7	0.895383E-01	0.892581E-01	0.596046E-07

Table A-3 Performance of Training and Generalization (RMS)  
Method B with 7 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.813377E-01	0.842336E-01	0.133371E-00
2	0.801080E-01	0.834001E-01	0.122967E-02
3	0.782553E-01	0.818001E-01	0.185277E-02
4	0.774735E-01	0.810877E-01	0.781715E-03
5	0.706896E-01*	0.741628E-01*	0.678393E-02
6	0.711018E-01	0.745780E-01	0.412233E-03

Table A-4 Performance of Training and Generalization (RMS)  
Method C with 7 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.812634E-01	0.842007E-01	0.133440E-00
2	0.801132E-01	0.834342E-01	0.115024E-02
3	0.788925E-01	0.824305E-01	0.122075E-02
4	0.778096E-01	0.814628E-01	0.108288E-02
5	0.744281E-01	0.781553E-01	0.338145E-02
6	0.732613E-01	0.769954E-01	0.116679E-02
7	0.725740E-01*	0.758244E-01*	0.687353E-03
8	0.729824E-01	0.743964E-01	0.408381E-03

Table A-5 Performance of Training and Generalization (RMS)  
Method C with 7 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.886283E-01	0.867396E-01	0.126221E-00
2	0.884512E-01*	0.866238E-01*	0.177145E-03
3	0.884734E-01	0.866356E-01	0.221804E-04



Table A-6 Performance of Training and Generalization (RMS)  
Method D with 7 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.812552E-01	0.841985E-01	0.133448
2	0.801644E-01	0.834818E-01	0.109082E-02
3	0.790322E-01	0.825625E-01	0.113216E-02
4	0.779749E-01	0.816252E-01	0.105730E-02
5	0.750445E-01	0.787846E-01	0.293044E-02
6	0.736659E-01	0.774192E-01	0.137859E-02
7	0.722210E-01	0.755748E-01	0.144488E-02
8	0.717990E-01	0.732518E-01	0.422016E-03
9	0.718357E-01	0.732768E-01	0.366718E-04
10	0.707211E-01	0.724163E-01	0.111452E-02
11	0.723729E-01	0.728283E-01	0.909194E-03
12	0.691364E-01*	0.698246E-01*	0.323655E-02
13	0.704434E-01	0.708009E-01	0.130697E-02

Table A-7 Performance of Training and Generalization (RMS)  
Method D with 7 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.812552E-01	0.841985E-01	0.133448
2	0.801644E-01	0.834818E-01	0.109082E-02
3	0.790322E-01	0.825625E-01	0.113216E-02
4	0.779749E-01	0.816252E-01	0.105730E-02
5	0.750445E-01	0.787846E-01	0.293044E-02
6	0.736659E-01	0.774192E-01	0.137859E-02
7	0.722210E-01	0.755748E-01	0.144488E-02
8	0.717990E-01	0.732518E-01	0.422016E-03
9	0.718357E-01	0.732768E-01	0.366718E-04
10	0.707211E-01*	0.724163E-01*	0.111452E-02
11	0.707750E-01	0.724481E-01	0.422075E-04

Table A-8 Performance of Training and Generalization (RMS)  
Method E with 7 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.812552E-01	0.841985E-01	0.133448
2	0.801644E-01	0.834818E-01	0.109082E-02
3	0.790322E-01	0.825625E-01	0.113216E-02
4	0.779749E-01	0.816252E-01	0.105730E-02
5	0.750445E-01	0.787846E-01	0.293044E-02
6	0.736659E-01	0.774192E-01	0.137859E-02
7	0.722210E-01	0.755748E-01	0.144488E-02
8	0.717990E-01	0.732518E-01	0.422016E-03
9	0.718357E-01	0.732768E-01	0.366718E-04
10	0.707211E-01*	0.724163E-01*	0.111452E-02
11	0.708656E-01	0.724377E-01	0.282601E-04

Table A-9 Performance of Training and Generalization (RMS)  
Method E with 7 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709E-00	0.209518E-00	
1	0.812552E-01	0.841985E-01	0.133448
2	0.801644E-01	0.834818E-01	0.109082E-02
3	0.790322E-01	0.825625E-01	0.113216E-02
4	0.779749E-01	0.816252E-01	0.105730E-02
5	0.750445E-01	0.787846E-01	0.293044E-02
6	0.736659E-01	0.774192E-01	0.137859E-02
7	0.722210E-01	0.755748E-01	0.144488E-02
8	0.717990E-01	0.732518E-01	0.422016E-03
9	0.718357E-01	0.732768E-01	0.366718E-04
10	0.707211E-01*	0.724163E-01*	0.111452E-02
11	0.707701E-01	0.724475E-01	0.415072E-04

Table A-10 Performance of Training and Generalization (RMS)  
Method A with 8 hidden nodes

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.818643E-01	0.848730E-01	0.131138
2	0.815437E-01	0.847499E-01	0.320621E-03
3	0.813113E-01	0.846303E-01	0.232413E-03
4	0.808220E-01	0.842961E-01	0.489302E-03
5	0.796452E-01	0.833113E-01	0.117680E-02
6	0.778920E-01	0.816870E-01	0.175317E-02
7	0.737960E-01	0.775554E-01	0.409602E-02
8	0.730081E-01*	0.767580E-01*	0.787854E-03
9	0.733293E-01	0.770622E-01	0.321187E-03

Table A-11 Performance of Training and Generalization (RMS)  
Method B with 8 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.819372E-01	0.848970E-01	0.131071
2	0.816100E-01	0.847689E-01	0.327244E-03
3	0.812510E-01	0.845630E-01	0.358932E-03
4	0.800752E-01	0.836221E-01	0.117583E-02
5	0.791521E-01	0.828127E-01	0.923134E-03
6	0.717076E-01*	0.751127E-01*	0.744448E-02
7	0.722484E-01	0.756384E-01	0.540763E-03

Table A-12 Performance of Training and Generalization (RMS)  
Method B with 8 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.818717E-01	0.848753E-01	0.131132
2	0.815524E-01	0.847525E-01	0.319220E-03
3	0.813135E-01	0.846287E-01	0.238933E-03
4	0.807856E-01	0.842638E-01	0.527889E-03
5	0.796023E-01	0.832680E-01	0.118332E-02
6	0.776854E-01	0.814787E-01	0.191688E-02
7	0.739080E-01	0.776723E-01	0.377746E-02
8	0.728923E-01	0.766369E-01	0.101567E-02
9	0.730239E-01	0.767609E-01	0.131637E-03
10	0.728760E-01*	0.764509E-01*	0.147901E-03
11	0.735858E-01	0.771347E-01	0.709720E-03

Table A-13 Performance of Training and Generalization (RMS)  
Method C with 8 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.818691E-01	0.848744E-01	0.131134E-00
2	0.815431E-01	0.847472E-01	0.325955E-03
3	0.812866E-01	0.846118E-01	0.256523E-03
4	0.806990E-01	0.841960E-01	0.587605E-03
5	0.794732E-01	0.831542E-01	0.122583E-02
6	0.772993E-01	0.811025E-01	0.217392E-02
7	0.738339E-01	0.776072E-01	0.346541E-02
8	0.728451E-01	0.765909E-01	0.988781E-03
9	0.728672E-01	0.766129E-01	0.221059E-04
10	0.718936E-01*	0.744065E-01*	0.973582E-03
11	0.722725E-01	0.747144E-01	0.378869E-03

Table A-14 Performance of Training and Generalization (RMS)  
Method C with 8 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.866797E-01	0.865098E-01	0.126489E-00
2	0.865809E-01*	0.864760E-01	0.987947E-04*
3	0.865913E-01	0.864781E-01	0.103712E-04

Table A-15 Performance of Training and Generalization (RMS)  
Method D with 8 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.818643E-01	0.848730E-01	0.131138
2	0.815437E-01	0.847499E-01	0.320621E-03
3	0.813113E-01	0.846303E-01	0.232413E-03
4	0.808220E-01	0.842961E-01	0.489302E-03
5	0.796452E-01	0.833113E-01	0.117680E-02
6	0.778920E-01	0.816870E-01	0.175317E-02
7	0.737960E-01	0.775554E-01	0.409602E-02
8	0.730081E-01	0.767580E-01	0.787854E-03
9	0.733293E-01	0.770622E-01	0.321187E-03
10	0.731537E-01	0.765954E-01	0.175618E-03
11	0.759106E-01	0.774551E-01	0.186249E-02
12	0.757639E-01	0.773498E-01	0.146680E-03
13	0.762574E-01	0.775192E-01	0.493556E-03
14	0.763435E-01	0.768638E-01	0.860468E-04
15	0.747234E-01	0.740842E-01	0.162011E-02
16	0.681907E-01*	0.677225E-01*	0.653267E-02
17	0.695431E-01	0.688462E-01	0.135239E-02

Table A-16 Performance of Training and Generalization (RMS)  
Method D with 8 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.818643E-01	0.848730E-01	0.131138
2	0.815437E-01	0.847499E-01	0.320621E-03
3	0.813113E-01	0.846303E-01	0.232413E-03
4	0.808220E-01	0.842961E-01	0.489302E-03
5	0.796452E-01	0.833113E-01	0.117680E-02
6	0.778920E-01	0.816870E-01	0.175317E-02
7	0.737960E-01	0.775554E-01	0.409602E-02
8	0.730081E-01	0.767580E-01	0.787854E-03
9	0.733293E-01	0.770622E-01	0.321187E-03
10	0.759254E-01	0.774626E-01	0.186847E-02
11	0.757737E-01	0.773533E-01	0.151746E-03
12	0.731537E-01	0.765954E-01	0.175618E-03
13	0.713689E-01	0.703227E-01	0.510639E-02
14	0.695947E-01	0.682037E-01	0.177421E-02
15	0.681585E-01*	0.669311E-01*	0.143620E-02
16	0.681996E-01	0.669665E-01	0.410751E-04

Table A-17 Performance of Training and Generalization (RMS)  
Method E with 8 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003E-00	0.207839E-00	
1	0.818643E-01	0.848730E-01	0.131138
2	0.815437E-01	0.847499E-01	0.320621E-03
3	0.813113E-01	0.846303E-01	0.232413E-03
4	0.808220E-01	0.842961E-01	0.489302E-03
5	0.796452E-01	0.833113E-01	0.117680E-02
6	0.778920E-01	0.816870E-01	0.175317E-02
7	0.737960E-01	0.775554E-01	0.409602E-02
8	0.730081E-01	0.767580E-01	0.787854E-03
9	0.733293E-01	0.770622E-01	0.321187E-03
10	0.731537E-01	0.765954E-01	0.175618E-03
11	0.756865E-01	0.775078E-01	0.187512E-02
12	0.747758E-01	0.767588E-01	0.910699E-03
13	0.743792E-01	0.763368E-01	0.975490E-03
14	0.735836E-01	0.754677E-01	0.156695E-02
15	0.718534E-01*	0.737869E-01*	0.173014E-02
16	0.724389E-01	0.726581E-01	0.585444E-03

Table A-18 Performance of Training and Generalization (RMS)  
Method A with 10 hidden nodes

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210536E-00	0.205405E-00	
1	0.825110E-01	0.855849E-01	0.128025
2	0.824249E-01	0.855839E-01	0.861138E-04
3	0.824023E-01	0.855835E-01	0.226125E-04
4	0.823923E-01	0.855831E-01	0.999123E-05
5	0.823858E-01	0.855828E-01	0.648201E-05
6	0.823841E-01	0.855827E-01	0.171363E-05
7	0.823805E-01	0.855825E-01	0.366569E-05
8	0.823793E-01	0.855825E-01	0.111759E-05
9	0.823794E-01	0.856824E-01	0.447035E-07
10	0.823790E-01	0.856824E-01	0.402331E-06
11	0.823793E-01	0.866823E-01	0.312924E-06
12	0.823795E-01*	0.865625E-01*	0.178814E-06
13	0.823806E-01	0.855825E-01	0.111759E-05

Table A-19 Performance of Training and Generalization (RMS)  
Method B with 10 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543	0.205405	
1	0.878857E-01	0.872232E-01	0.123397
2	0.878564E-01	0.872218E-01	0.293776E-04
3	0.878569E-01	0.872217E-01	0.514090E-06
4	0.878569E-01	0.872217E-01	0.00000
5	0.878567E-01*	0.872217E-01*	0.186265E-06
6	0.878574E-01	0.872217E-01	0.707805E-06

Table A-20 Performance of Training and Generalization (RMS)  
Method B with 10 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543	0.205405	
1	0.825718E-01	0.855999E-01	0.127972
2	0.824931E-01	0.855999E-01	0.787005E-04
3	0.824664E-01	0.855993E-01	0.266880E-04
4	0.824511E-01	0.855984E-01	0.153333E-04
5	0.824397E-01	0.855975E-01	0.113398E-04
6	0.824122E-01	0.855968E-01	0.753254E-05
7	0.823246E-01	0.855960E-01	0.759959E-05
8	0.814179E-01	0.857852E-01	0.672042E-05
9	0.814121E-01	0.835944E-01	0.582635E-05
10	0.804066E-01	0.825736E-01	0.544637E-05
11	0.782809E-01	0.813131E-01	0.562519E-05
12	0.782748E-01	0.813120E-01	0.648946E-05
13	0.782682E-01*	0.813108E-01*	0.693649E-05
14	0.782996E-01	0.813791E-01	0.901520E-05

Table A-21 Performance of Training and Generalization (RMS)  
Method C with 10 hidden nodes and  $\lambda = 0.00001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543E-00	0.205405E-00	
1	0.825117E-01	0.855866E-01	0.128024E-00
2	0.824285E-01	0.855855E-01	0.832081E-04
3	0.824051E-01	0.855849E-01	0.234768E-04
4	0.823928E-01	0.855844E-01	0.122115E-04
5	0.823875E-01	0.855841E-01	0.533462E-05
6	0.823824E-01	0.855839E-01	0.509620E-05
7	0.823805E-01	0.855837E-01	0.195205E-05
8	0.823797E-01	0.855837E-01	0.789762E-06
9	0.823783E-01*	0.855836E-01*	0.137091E-05
10	0.823790E-01	0.855836E-01	0.707805E-06



Table A-22 Performance of Training and Generalization (RMS)  
Method C with 10 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543E-00	0.205405E-00	
1	0.853625E-01	0.865173E-01	0.125381E-00
2	0.853090E-01	0.865138E-01	0.534728E-04
3	0.853115E-01*	0.865137E-01*	0.249594E-05
4	0.853128E-01	0.865136E-01	0.130385E-05

Table A-23 Performance of Training and Generalization (RMS)  
Method D with 10 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210536	0.205405	
1	0.825110E-01	0.855849E-01	0.128025
2	0.824249E-01	0.855839E-01	0.861138E-04
3	0.824023E-01	0.855835E-01	0.226125E-04
4	0.823923E-01	0.855831E-01	0.999123E-05
5	0.823858E-01	0.855828E-01	0.648201E-05
6	0.823841E-01	0.855827E-01	0.171363E-05
7	0.823805E-01	0.855825E-01	0.366569E-05
8	0.823793E-01	0.855923E-01	0.111759E-05
9	0.823792E-01	0.856874E-01	0.447035E-07
10	0.823790E-01	0.855824E-01	0.402331E-06
11	0.825205E-01	0.850717E-01	0.786036E-05
12	0.814152E-01	0.846813E-01	0.528991E-05
13	0.802317E-01	0.835607E-01	0.454485E-05
14	0.800074E-01	0.821903E-01	0.326335E-05
15	0.784048E-01	0.815800E-01	0.263005E-05
16	0.772402E-01	0.794055E-01	0.261515E-05
17	0.753999E-01*	0.763246E-01*	0.227243E-05
18	0.766967E-01	0.795789E-01	0.315905E-05

Table A-24 Performance of Training and Generalization (RMS)  
Method D with 10 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543E-00	0.205405E-00	
1	0.825110E-01	0.855849E-01	0.128025
2	0.824249E-01	0.855839E-01	0.861138E-04
3	0.824023E-01	0.855835E-01	0.226125E-04
4	0.823923E-01	0.855831E-01	0.999123E-05
5	0.823858E-01	0.855828E-01	0.648201E-05
6	0.823841E-01	0.855827E-01	0.171363E-05
7	0.823805E-01	0.855825E-01	0.366569E-05
8	0.823793E-01	0.855825E-01	0.111759E-05
9	0.823794E-01	0.855824E-01	0.447035E-07
10	0.823790E-01	0.855824E-01	0.402331E-06
11	0.823841E-01	0.855824E-01	0.186265E-06
12	0.823741E-01*	0.855825E-01*	0.566244E-06
13	0.823931E-01	0.855824E-01	0.151992E-05

Table A-25 Performance of Training and Generalization (RMS)  
Method E with 10 hidden nodes and  $\lambda = 0.01$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543E-00	0.205405E-00	
1	0.825110E-01	0.855849E-01	0.128025
2	0.824249E-01	0.855839E-01	0.861138E-04
3	0.824023E-01	0.855835E-01	0.226125E-04
4	0.823923E-01	0.855831E-01	0.999123E-05
5	0.823858E-01	0.855828E-01	0.648201E-05
6	0.823841E-01	0.855827E-01	0.171363E-05
7	0.823805E-01	0.855825E-01	0.366569E-05
8	0.823793E-01	0.855825E-01	0.111759E-05
9	0.823794E-01	0.855824E-01	0.447035E-07

Table A-26 Performance of Training and Generalization (RMS)  
Method E with 10 hidden nodes and  $\lambda = 0.00001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543E-00	0.205405E-00	
1	0.825110E-01	0.855849E-01	0.128025
2	0.824249E-01	0.855839E-01	0.861138E-04
3	0.824023E-01	0.855835E-01	0.226125E-04
4	0.823923E-01	0.855831E-01	0.999123E-05
5	0.823858E-01	0.855828E-01	0.648201E-05
6	0.823841E-01	0.855827E-01	0.171363E-05
7	0.823805E-01	0.855825E-01	0.366569E-05
8	0.823793E-01	0.855825E-01	0.111759E-05
9	0.823794E-01	0.855824E-01	0.447035E-07
10	0.823790E-01	0.855824E-01	0.402331E-06
11	0.823795E-01	0.855823E-01	0.154972E-05
12	0.823793E-01	0.855823E-01	0.178814E-06
13	0.823779E-01	0.855822E-01	0.144541E-05



Table A-27 Performance of Training and Generalization (RMS)  
 Method F with 7 hidden nodes and different  $\lambda$  (0.008, 0.006,  
 0.004, 0.002, 0.001, 0.0006, 0.0001, 0.00006, 0.00004, 0.00001)

Epoch	$\lambda$	Training RMS	Generalization RMS	Convergence Error
1	0.80000E-02	0.860905E-01	0.848817E-01	0.128730E-00
2	0.60000E-02	0.846864E-01	0.847823E-01	0.465959E-03
3	0.40000E-02	0.837784E-01	0.847662E-01	0.478327E-04
4	0.20000E-02	0.828646E-01	0.846507E-01	0.511557E-04
5	0.10000E-02	0.802596E-01	0.806985E-01	0.211523E-02
6	0.60000E-03	0.791398E-01	0.792753E-01	0.787571E-03
7	0.10000E-03*	0.771920E-01*	0.786071E-01	0.133017E-02
8	0.60000E-04	0.774715E-01	0.784241E-01	0.323653E-03
9	0.40000E-04	0.774570E-01	0.784329E-01	0.331238E-03
10	0.10000E-04	0.774264E-01	0.784434E-01	0.333793E-03
11	0.10000E-03	0.771808E-01	0.783094E-01	0.358023E-03
12	0.10000E-03	0.771222E-01	0.782939E-01	0.586659E-04
13	0.10000E-03	0.813377E-01	0.842336E-01	0.133371E-00

Table A-28 Performance of Training and Generalization (RMS)  
 Method G with 7 hidden nodes and  $\lambda = 0.001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214756	0.209518	
1	0.817576E-01	0.841984E-01	0.132999
2	0.807776E-01	0.835390E-01	0.979960E-03
3	0.798116E-01	0.827304E-01	0.966057E-03
4	0.788325E-01	0.818250E-01	0.979044E-03
5	0.766781E-01	0.796450E-01	0.215444E-02
6	0.752509E-01	0.781470E-01	0.142720E-02
7	0.714120E-01*	0.736984E-01*	0.383891E-02
8	0.730710E-01	0.752282E-01	0.165908E-02

Table A-29 Performance of Training and Generalization (RMS)  
 Method G with 7 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214709	0.209518	
1	0.813054E-01	0.841985E-01	0.133404
2	0.802277E-01	0.834886E-01	0.107767E-02
3	0.791121E-01	0.825798E-01	0.111558E-02
4	0.780597E-01	0.816429E-01	0.105239E-02
5	0.752288E-01	0.788873E-01	0.283096E-02
6	0.738405E-01	0.775041E-01	0.138825E-02
7	0.721871E-01	0.754372E-01	0.165343E-02
8	0.716075E-01	0.730340E-01	0.579566E-03
9	0.716354E-01	0.730538E-01	0.278950E-04
10	0.709259E-01*	0.725106E-01*	0.709549E-03
11	0.709502E-01	0.725289E-01	0.243112E-04

Table A-30 Performance of Training and Generalization (RMS)  
Method G with 7 hidden nodes and  $\lambda = 0.0002$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.214705	0.209518	
1	0.812652E-01	0.841985E-01	0.133439
2	0.801743E-01	0.834814E-01	0.109088E-02
3	0.790483E-01	0.825664E-01	0.112601E-02
4	0.779875E-01	0.816249E-01	0.106080E-02
5	0.750842E-01	0.788082E-01	0.290331E-02
6	0.736947E-01	0.774301E-01	0.138956E-02
7	0.721961E-01	0.755340E-01	0.149855E-02
8	0.717005E-01	0.731551E-01	0.495657E-03
9	0.717307E-01	0.731760E-01	0.302196E-04
10	0.707547E-01*	0.724244E-01*	0.976011E-03
11	0.707797E-01	0.724433E-01	0.250116E-04

Table A-31 Performance of Training and Generalization (RMS)  
Method G with 8 hidden nodes and  $\lambda = 0.0002$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213004	0.207839	
1	0.818767E-01	0.848726E-01	0.131127
2	0.815572E-01	0.847500E-01	0.319563E-03
3	0.813266E-01	0.846321E-01	0.230514E-03
4	0.808509E-01	0.843091E-01	0.475705E-03
5	0.796952E-01	0.833439E-01	0.115574E-02
6	0.779820E-01	0.817583E-01	0.171316E-02
7	0.738800E-01	0.776176E-01	0.410205E-02
8	0.730631E-01	0.767895E-01	0.816934E-03
9	0.733877E-01	0.770932E-01	0.324629E-03
10	0.731922E-01	0.766110E-01	0.195459E-03
11	0.733967E-01	0.768062E-01	0.204444E-03
12	0.731807E-01	0.750615E-01	0.215985E-03
13	0.732087E-01	0.750793E-01	0.279844E-04
14	0.730509E-01*	0.749718E-01*	0.157736E-03
15	0.730594E-01	0.749775E-01	0.849366E-05

Table A-32 Performance of Training and Generalization (RMS)  
Method G with 8 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.213003	0.207839	
1	0.818695E-01	0.848729E-01	0.131134
2	0.815518E-01	0.847507E-01	0.317685E-03
3	0.813209E-01	0.846324E-01	0.230849E-03
4	0.808411E-01	0.843060E-01	0.479840E-03
5	0.796791E-01	0.833353E-01	0.116200E-02
6	0.779562E-01	0.817408E-01	0.172289E-02
7	0.738410E-01	0.775888E-01	0.411517E-02
8	0.730438E-01	0.767817E-01	0.797175E-03
9	0.733867E-01	0.771038E-01	0.342883E-03
10	0.731672E-01	0.765858E-01	0.219509E-03
11	0.733556E-01	0.767654E-01	0.188418E-03
12	0.728855E-01	0.748343E-01	0.470124E-03
13	0.728628E-01*	0.748189E-01*	0.227168E-04
14	0.729582E-01	0.748835E-01	0.954196E-04

Table A-33 Performance of Training and Generalization (RMS)  
Method G with 10 hidden nodes and  $\lambda = 0.0001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210543	0.205405	
1	0.825592E-01	0.855851E-01	0.127984
2	0.824773E-01	0.855839E-01	0.819638E-04
3	0.824542E-01	0.855834E-01	0.230819E-04
4	0.824432E-01	0.855830E-01	0.109598E-04
5	0.824401E-01	0.855829E-01	0.310689E-05
6	0.824371E-01	0.855827E-01	0.302494E-05
7	0.824355E-01	0.855827E-01	0.164658E-05
8	0.824354E-01	0.855827E-01	0.745058E-07
9	0.824343E-01*	0.855826E-01*	0.110269E-05
10	0.824355E-01	0.855827E-01	0.122935E-05

Table A-34 Performance of Training and Generalization (RMS)  
Method G with 10 hidden nodes and  $\lambda = 0.00001$

Epoch	Training RMS	Generalization RMS	Convergence Error
0	0.210537	0.205405	
1	0.825160E-01	0.855848E-01	0.128021
2	0.824300E-01	0.855838E-01	0.860468E-04
3	0.824081E-01	0.855834E-01	0.218600E-04
4	0.823960E-01	0.855830E-01	0.120923E-04
5	0.823906E-01	0.855827E-01	0.540167E-05
6	0.823877E-01	0.855826E-01	0.288337E-05
7	0.823852E-01	0.855824E-01	0.255555E-05
8	0.823851E-01*	0.855824E-01*	0.968575E-07
9	0.823867E-01	0.855825E-01	0.166148E-05

Table A-35 The Training Data Set

0.0000000E+00	0.0000000E+00	0.0000000E+00
0.0000000E+00	0.8333336E-01	0.69444450E-02
0.0000000E+00	0.1666667E+00	0.27777780E-01
0.0000000E+00	0.2500000E+00	0.6250000E-01
0.0000000E+00	0.3333334E+00	0.11111112E+00
0.0000000E+00	0.4166666E+00	0.17361110E+00
0.0000000E+00	0.5000000E+00	0.2500000E+00
0.8333336E-01	0.0000000E+00	0.69444450E-02
0.8333336E-01	0.8333336E-01	0.1388890E-01
0.8333336E-01	0.1666667E+00	0.3472224E-01
0.8333336E-01	0.2500000E+00	0.6944448E-01
0.8333336E-01	0.3333334E+00	0.1180556E+00
0.8333336E-01	0.4166666E+00	0.1805555E+00
0.8333336E-01	0.5000000E+00	0.2569445E+00
0.1666667E+00	0.0000000E+00	0.2777780E-01
0.1666667E+00	0.8333336E-01	0.3472224E-01
0.1666667E+00	0.1666667E+00	0.5555560E-01
0.1666667E+00	0.2500000E+00	0.9027776E-01
0.1666667E+00	0.3333334E+00	0.1388890E+00
0.1666667E+00	0.4166666E+00	0.2013888E+00
0.1666667E+00	0.5000000E+00	0.2777779E+00
0.2500000E+00	0.0000000E+00	0.6250000E-01
0.2500000E+00	0.8333336E-01	0.6944448E-01
0.2500000E+00	0.1666667E+00	0.9027776E-01
0.2500000E+00	0.2500000E+00	0.1250000E+00
0.2500000E+00	0.3333334E+00	0.17361112E+00
0.2500000E+00	0.4166666E+00	0.23611110E+00
0.2500000E+00	0.5000000E+00	0.3125000E+00
0.3333334E+00	0.0000000E+00	0.11111112E+00
0.3333334E+00	0.8333336E-01	0.1180557E+00
0.3333334E+00	0.1666667E+00	0.1388890E+00
0.3333334E+00	0.2500000E+00	0.17361112E+00
0.3333334E+00	0.3333334E+00	0.2222224E+00
0.3333334E+00	0.4166666E+00	0.2847221E+00
0.3333334E+00	0.5000000E+00	0.36111110E+00
0.4166666E+00	0.0000000E+00	0.17361110E+00
0.4166666E+00	0.8333336E-01	0.1805555E+00
0.4166666E+00	0.1666667E+00	0.2013888E+00
0.4166666E+00	0.2500000E+00	0.23611110E+00
0.4166666E+00	0.3333334E+00	0.2847221E+00
0.4166666E+00	0.4166666E+00	0.3472221E+00
0.4166666E+00	0.5000000E+00	0.42361110E+00
0.5000000E+00	0.0000000E+00	0.2500000E+00
0.5000000E+00	0.8333336E-01	0.2569445E+00
0.5000000E+00	0.1666667E+00	0.2777779E+00
0.5000000E+00	0.2500000E+00	0.3125000E+00
0.5000000E+00	0.3333334E+00	0.36111110E+00
0.5000000E+00	0.4166666E+00	0.42361110E+00
0.5000000E+00	0.5000000E+00	0.5000000E+00

Table A-36 The Validation Data Set

0.99999998E-02	0.99999998E-02	0.19999999E-03
0.99999998E-02	0.93333334E-01	0.88111116E-02
0.99999998E-02	0.17666666E+00	0.31311110E-01
0.99999998E-02	0.25999999E+00	0.67699999E-01
0.99999998E-02	0.34333333E+00	0.11797778E+00
0.99999998E-02	0.42666668E+00	0.18214445E+00
0.99999998E-02	0.50999999E+00	0.26019999E+00
0.93333334E-01	0.99999998E-02	0.88111106E-02
0.93333334E-01	0.93333334E-01	0.17422222E-01
0.93333334E-01	0.17666666E+00	0.39922219E-01
0.93333334E-01	0.25999999E+00	0.76311104E-01
0.93333334E-01	0.34333333E+00	0.12658890E+00
0.93333334E-01	0.42666668E+00	0.19075556E+00
0.93333334E-01	0.50999999E+00	0.26881111E+00
0.17666666E+00	0.99999998E-02	0.31311110E-01
0.17666666E+00	0.93333334E-01	0.39922222E-01
0.17666666E+00	0.17666666E+00	0.62422220E-01
0.17666666E+00	0.25999999E+00	0.98811105E-01
0.17666666E+00	0.34333333E+00	0.14908889E+00
0.17666666E+00	0.42666668E+00	0.21325557E+00
0.17666666E+00	0.50999999E+00	0.29131109E+00
0.25999999E+00	0.99999998E-02	0.67699999E-01
0.25999999E+00	0.93333334E-01	0.76311111E-01
0.25999999E+00	0.17666666E+00	0.98811105E-01
0.25999999E+00	0.25999999E+00	0.13519999E+00
0.25999999E+00	0.34333333E+00	0.18547778E+00
0.25999999E+00	0.42666668E+00	0.24964444E+00
0.25999999E+00	0.50999999E+00	0.32769999E+00
0.34333333E+00	0.99999998E-02	0.11797778E+00
0.34333333E+00	0.93333334E-01	0.12658890E+00
0.34333333E+00	0.17666666E+00	0.14908889E+00
0.34333333E+00	0.25999999E+00	0.18547778E+00
0.34333333E+00	0.34333333E+00	0.23575556E+00
0.34333333E+00	0.42666668E+00	0.29992223E+00
0.34333333E+00	0.50999999E+00	0.37797776E+00
0.42666668E+00	0.99999998E-02	0.18214445E+00
0.42666668E+00	0.93333334E-01	0.19075556E+00
0.42666668E+00	0.17666666E+00	0.21325555E+00
0.42666668E+00	0.25999999E+00	0.24964444E+00
0.42666668E+00	0.34333333E+00	0.29992223E+00
0.42666668E+00	0.42666668E+00	0.36408889E+00
0.42666668E+00	0.50999999E+00	0.44214442E+00
0.50999999E+00	0.99999998E-02	0.26019996E+00
0.50999999E+00	0.93333334E-01	0.26881108E+00
0.50999999E+00	0.17666666E+00	0.29131109E+00
0.50999999E+00	0.25999999E+00	0.32769996E+00
0.50999999E+00	0.34333333E+00	0.37797776E+00
0.50999999E+00	0.42666668E+00	0.44214442E+00
0.50999999E+00	0.50999999E+00	0.52019995E+00

## APPENDIX B

### COMPUTER PROGRAMS

#### PROGRAM DRIVER

```

C*****
C THIS DRIVER IS TO GENERATE THE RANDOM WEIGHTS *
C W(MLAYR, MNODE, 0:MNODE) --THE WEIGHT OF *
C EACH LAYER. *
C P(MNODE) -- THE INPUT DATA OF THE SAMPLE. *
C O(MNODE) -- THE OUTPUT CALCULATED FROM THE INPUT *
C DATA SAMPLE. *
C N(MLAYR, MNODE) -- THE WEIGHTED SUM OF THE *
C INPUTS OF A NEURON MNODE IN LAYER MLAYR *
C REF (3.1.1) *
C A(0:MLAYR, 0:MNODE) -- THE OUTPUT OF THE NEURON *
C MNODE IN LAYER MLAYR. REF (3.1.2) *
C NOTICE THAT A(0,*) REPRESENTS THE INPUT *
C LAYER. A(*,0) REPRESENTS THE BIAS. *
C NNODE(0:MLAYR) -- THE NUMBER OF NODE IN EACH *
C LAYER. *
C LAYER -- THE ACTUAL TOTAL LAYER OF THE NET. (EXCLUDING *
C THE INPUT LAYER) *
C MLAYR -- THE MAXMUM LAYER A NET CAN HAVE. *
C MNODE -- THE MAXMUM NODE ONE LAYER OF A NET CAN HAVE *
C LL -- SAMPLE INDEX *
C*****
PARAMETER(MLAYR = 4, MNODE = 100,MSAMP = 200)
DOUBLE PRECISION DRANDOM, W(MLAYR, MNODE, 0:MNODE),
+ SEED,TOL,W0,LAMDA,LAMDA2,P(MSAMP,MNODE),O(MSAMP,MNODE),
+ N(MLAYR,MNODE), A(0:MLAYR,0:MNODE),
+ SENSI(MLAYR,MNODE),T(MSAMP,MNODE),ERROR2,ERROR1,
+ G(MLAYR,MNODE,0:MNODE),TG(MLAYR,MNODE,0:MNODE),
+ FRET,TOL1,ERROR,ERRORV
INTEGER K,I,J,LL,NNODE(0:MLAYR),METHOD,LAYER,NSAMP
INTEGER NUM,ITER,MAXNUM,NWEIG,PSTAT,JUDGE
C
C THE FOLLOWING DATA IS USED IN CONJUGATE GRADIENT METHOD
C
DOUBLE PRECISION PP(MLAYR,MNODE,0:MNODE),BETA,
+ TG0(MLAYR,MNODE,0:MNODE),PP0(MLAYR,MNODE,0:MNODE)
PSTAT=10

```



```

C
C   SET UP NETWORK
C
CALL NETSETUP(LAYER,MLAYR,NNODE,SEED,W0,LAMDA,LAMDA2,METHOD)
WRITE(*,123)LAMDA,LAMDA2
123  FORMAT(1X,'LAM1=',F16.12,'LAM2=',F16.12)
CALL NETPRINT(LAYER,MLAYR,NNODE,SEED,W0,LAMDA,LAMDA2,METHOD)
C
C   INITIAL WEIGHT WITH RANDOM NUMBER.
C
CALL INIWEIGHT(W,LAYER,MLAYR,NNODE,MNODE,SEED,
+ NWEIG)
C
C   PRINT THE NUMBER OF WEIGHT
C
WRITE(*,1001)NWEIG
1001  FORMAT(1X,'THE NUMBER OF WEIGHT IS:',15)
C
C   READ IN TRAINING DATA P(I) AND T(I)
C   READ IN THE INPUT AND DESIRED OUTPUT OF ONE TRAINING SAMPLE
C
CALL GETINPUTDATA(P,T,MNODE,NNODE(0),NNODE(LAYER),
+ MSAMP,NSAMP)
C
C   CALCULATE THE PERFORMANCE FUNCTION
C
ERROR = SQRT(TEST(MSAMP,MNODE,W,MLAYR,LAYER,
+ NNODE,LAMDA,W0)/NSAMP)
PRINT*, 'BEFORE TRAINING GENERALIZATION ERROR: ',ERROR
C
C   LOOP OVER ITERATION
C   SET TOLERANCE AND MAXIMUM ITERATION NUMBER
C
TOL = 4.0D-10
TOL1 = 3.5D-2
MAXITER=10
ITER=0
JUDGE=0
C
1000  ITER = ITER + 1
C
C   ENTER ITERATION
C
CALL INITG(LAYER,MLAYR,NNODE,MNODE,TG)
C
C   SUM TOTAL GRADIENT
C
DO 320 LL=1,NSAMP
C
C   FEEDFORWARD COMPUTATION
C
CALL FORWARD(P,O,N,MLAYR,LAYER,MNODE,A,
+ NNODE,W,LL,MSAMP)
C
C   CALCULATE THE SENSITIVITY MATRIX
C

```

```

        CALL SENSITIVITY(SENSI,W,LAYER,MLAYR,NNODE,
+ MNODE,T,O,N,LL,MSAMP)
C
C CALCULATE THE GRADIENT OF THE PERFORMANCE FUNCTION
C
        CALL GRAD(SENSI,A,W,LAYER, MLAYR,
+ NNODE,MNODE,G,W0,LAMDA)
C
C SUM UP THE TOTAL GRADIENT
C
        CALL SUMGRAD(G,LAYER,MLAYR,NNODE,MNODE,TG)
320  CONTINUE
C
C FIND THE PERFORMANCE FUNCTION VALUE, BEFORE LINE SEARCH
C
        ERROR1= SQRT(FINDE(P,T,MSAMP,NSAMP,MNODE,
+ W,MLAYR,LAYER,NNODE,O,LAMDA,W0) /NSAMP)
        IF (MOD(ITER,PSTAT) .EQ. 1) THEN
C      WRITE(*,1600)ITER,ERROR1
1600  FORMAT(1X,'BEFORE LINE SEARCH, ITER #',I5,2X,
+ 'ERROR1 VALUE = ',G25.20)
        ENDIF
C
C      FIRST START AND RESTART USING STEEPEST DESCENT
C
        IF (ITER .EQ. 1 .OR. MOD(ITER,NWEIG) .EQ. 0) THEN
            CALL GETPP(PP,PP0,TG,MLAYR,LAYER,MNODE,NNODE,
+ 0.D0)
        ENDIF
C
C ASSIGN THE TG TO TG0
C
        CALL ASSIGN(TG,TG0,MLAYR,LAYER,MNODE,NNODE)
        CALL ASSIGN(PP,PP0,MLAYR,LAYER,MNODE,NNODE)
C
C COMPUTE ALGORITHM 3.6.1 (4) AND (5).
        CALL LINMIN(FRET,P,T,MSAMP,NSAMP,
+ MNODE,W,PP0,MLAYR,LAYER,NNODE,O,LAMDA,W0)
C
C USING STEP 2 TO COMPUTE THE G(K+1)
C
        CALL INITG(LAYER,MLAYR,NNODE,MNODE,TG)
        DO 321 LL=1,NSAMP
C
C      FEEDFORWARD COMPUTATION
C
        CALL FORWARD(P,O,N,MLAYR,LAYER,MNODE,A,
+ NNODE,W,LL,MSAMP)
C
C CALCULATE THE SENSITIVITY MATRIX
C
        CALL SENSITIVITY(SENSI,W,LAYER,MLAYR,NNODE,
+ MNODE,T,O,N,LL,MSAMP)
C
C CALCULATE THE GRADIENT OF THE PERFORMANCE FUNCTION
C

```



```

        CALL GRAD(SENSI,A,W,LAYER,MLAYR,
+ NNODE,MNODE,G,W0,LAMDA)
C
C SUM UP THE TOTAL GRADIENT
C
        CALL SUMGRAD(G,LAYER,MLAYR,NNODE,MNODE,TG)
321  CONTINUE
C
C FIND THE PERFORMANCE FUNCTION VALUE, AFTER LINE SEARCH
C
        ERROR2= SQRT(FINDE(P,T,MSAMP,NSAMP,MNODE,
+ W,MLAYR,LAYER,NNODE,O,LAMDA,W0)/NSAMP)
C  IF(MOD(ITER,PSTAT) .EQ. 0) THEN
C      WRITE(*,1100)ITER,ERROR2
1100  FORMAT(1X,'AFTER LINE SEARCH, ITER# ',I5,2X,
+ 'ERROR2 VALUE = ',G25.20)
C  ENDIF

C  IF(MOD(ITER,PSTAT) .EQ. 1) THEN
C  IF (ABS(ERROR2 - ERROR1) .LT. TOL) THEN
        ERROR = ABS(ERROR2 - ERROR1)
C  WRITE(*,101)ERROR
101  FORMAT (1X,'ERROR = ',G25.20)
C  STOP
C  ENDIF

C VALIDATE THE NETWORK USING VALIDATION SET.

C  IF(MOD(ITER,PSTAT) .EQ. 1) THEN
        ERRORV = SQRT(TEST(MSAMP,MNODE,W,MLAYR,LAYER,
+ NNODE,LAMDA,W0)/NSAMP)
        WRITE(*,1211)ITER,ERROR2,ERRORV,ERROR
1211  FORMAT(I2,',',1X,G15.6,',',1X,G15.6,',',1X,G15.6)
1200  FORMAT(1X,'AFTER TRAINING GENERALIZATION ERROR: ',G25.20)
C  ENDIF
C
        BETA=FINDBETA(TG,TG0,MLAYR,LAYER,MNODE,
+ NNODE)
        CALL GETPP(PP,PP0,TG,MLAYR,LAYER,MNODE,NNODE,
+ BETA)
C
C  PRINT TG, AFTER STEP 7
C
C ASSING TG TO TG0, TG0 STORES P(K+1)
C
        CALL ASSIGN(TG,TG0,MLAYR,LAYER,MNODE,NNODE)
C
        IF(JUDGE.EQ.0)THEN
        IF ((ABS(ERROR2 - ERROR1) .GT. TOL .OR. ERROR1 .GT. TOL1
+ .OR. ERROR2 .GT. TOL1) .AND. ITER .LT. MAXITER)THEN
        ERROR1 = ERROR2
        GOTO 1000
        ELSE
        JUDGE=1
        LAMDA=LAMDA2
        ITER=0

```

```

        GOTO 1000
    ENDIF
ENDIF
IF(JUDGE.EQ.1)THEN
IF ((ABS(ERROR2 - ERROR1) .GT. TOL .OR. ERROR1 .GT. TOL1
+ .OR. ERROR2 .GT. TOL1) .AND. ITER .LT. MAXITER)THEN
    ERROR1 = ERROR2
    GOTO 1000
ENDIF
ENDIF
    WRITE(*,1300)
    WRITE(*,1301)ITER, MAXITER
1301  FORMAT(1X,'ITER=',I5,4X,'MAXITER=', I5)
1300  FORMAT(1X,'SOLUTION CONVERGE TO THE TOLERANCE')
    WRITE(*,1400)ITER,ERROR1,ERROR2,ABS(ERROR2-ERROR1)
1400  FORMAT(1X,'ITER=',I5,2X,'ERROR1= ',G25.10,2X,'ERROR2= ',
+      G25.10,2X,'ERROR=', G25.10)
C
C TEST THE NETWORK USING TEST SET
C
    ERROR = SQRT(TEST(MSAMP,MNODE,W,MLAYR,LAYER,
+ NNODE,LAMDA,W0)/NSAMP)
    WRITE(*,1500)ERROR
1500  FORMAT(1X,'AFTER TRAINING ERROR=',G25.10)
C
    STOP
    END
C*****

SUBROUTINE ASSIGN(ORIG,NEW,MLAYR,NLAYR,MNODE,
+ NNODE)
C*****
C  THIS SUBROUTINE IS TO COPY A ORIG MATRIX TO NEW MATRIX.  *
C  IT IS USED TO COPY TG.  *
C*****
    INTEGER MLAYR,NLAYR,MNODE,NNODE(0:MLAYR)
    DOUBLE PRECISION ORIG(MLAYR,MNODE,0:MNODE),
+ NEW(MLAYR,MNODE,0:MNODE)
    INTEGER I,J,K,KK,LL
C
    DO 10 K=1,NLAYR
        KK=NNODE(K)
        LL=NNODE(K-1)
        DO 20 J=1,KK
            DO 30 I=0,LL
                NEW(K,J,I)=ORIG(K,J,I)
30          CONTINUE
20        CONTINUE
10      CONTINUE
C
    RETURN
    END
C*****

FUNCTION BRENT(AX,BX,CX,F,TOL,XMIN)

```

```

C*****
C   GIVEN A FUNCTION F, AND GIVEN A BRACKETING      *
C   TRIPLET OF ABSCESSAS AX, BX, CX(SUCH THAT BX IS *
C   BETWEEN AX, AND CX, AND F(BX) IS LESS THAN BOTH *
C   F(AX) AND F(CX)), THIS ROUTINE ISOLATES THE MINIMUM *
C   TO A FRACTIONAL PRECISION OF ABOUT TOL USING BRENT'S *
C   METHOD. THIS ABXCISSA OF THE MINIMUM IS RETURNED AS *
C   XMIN, AND MUNIMUM FUNCTION VALUE IS RETURNED AS BRENT, *
C   THE RETURNED FUNCTION VALUE.                    *
C
C   PARAMETERS: MAXIMUM ALLOWED NUMBER OF ITERATIONS; GOLDEN*
C   RATIO; AND A SMALL NUMBER THAT PROTECTS AGAINST TRYING *
C   TO ACHIEVE FRACTION ACCURACY FOR A MINIMUM THAT HAPPENS *
C   TO BE EXACTLY ZERO.                             *
C*****
C   INTEGER ITMAX
C   DOUBLE PRECISION BRENT, AX,BX,CX,TOL,XMIN,F,CGOLD,ZEPS
C   EXTERNAL F
C   PARAMETER(ITMAX=100, CGOLD=.381966D0,ZEPS=1.0D-10)
C
C   INTEGER ITER
C   DOUBLE PRECISION A,B,D,E,ETEMP,FU,FV,FW,FX,P,Q,R,TOL1,TOL2,
+  U,V,W,X,XM
C   A=MIN(AX,CX)
C   B=MAX(AX,CX)
C   V=BX
C   W=V
C   X=V
C   E=0.D0
C   FX=F(X)
C   FV=FX
C   FW=FX
C   DO 11 ITER = 1, ITMAX
C     XM = .5D0*(A+B)
C     TOL1 = TOL*ABS(X) +ZEPS
C     TOL2 = 2.D0*TOL1
C     IF(ABS(X-XM) .LE. (TOL2 - .5D0*(B-A))) GOTO 3
C     IF(ABS(E) .GT. TOL1)THEN
C       R=(X-W)*(FX-FV)
C       Q=(X-V)*(FX-FW)
C       P=(X-V)*Q-(X-W)*R
C       Q=2.D0*(Q-R)
C       IF(Q.GT.0) P=-P
C       Q=ABS(Q)
C       ETEMP=E
C       E=D
C       IF(ABS(P).GE.ABS(.5D0*Q*ETEMP).OR.P.LE.Q*(A-X).OR.
+       P.GE.Q*(B-X))GOTO 1
C       D=P/Q
C       U=X+D
C       IF(U-A.LT.TOL2 .OR. B-U .LT. TOL2)D=DSIGN(TOL1,XM-X)
C       GOTO 2
C     ENDIF
C   1   IF(X.GE.XM)THEN
C       E=A-X
C     ELSE

```

```

        E=B-X
    ENDIF
    D=CGOLD*E
2    IF(ABS(D).GE.TOL1)THEN
        U=X+D
    ELSE
        U=X+DSIGN(TOL1,D)
    ENDIF
    FU = F(U)
    IF(FU.LE.FX)THEN
        IF(U.GE.X)THEN
            A=X
        ELSE
            B=X
        ENDIF
        V=W
        FV=FW
        W=X
        FW=FX
        X=U
        FX=FU
    ELSE
        IF(U.LT.X)THEN
            A=U
        ELSE
            B=U
        ENDIF
        IF(FU.LE.FW .OR. W.EQ.X)THEN
            V=W
            FV=FW
            W=U
            FW=FU
        ELSEIF(FU .LE. FV .OR. V.EQ.X .OR. V.EQ.W)THEN
            V=U
            FV=FU
        ENDIF
    ENDIF
11   CONTINUE
C
3    XMIN=X
    BRENT=FX
    RETURN
    END
C*****

SUBROUTINE CONVERT(TG,MLAYR,MNODE,NLAYR,
+ NNODE,A,MAXNUM,NUM)
C*****
C   THIS ROUTINE IS TO CONVERT THE 3-DIMENSIONAL      *
C   ARRAYS INTO 1-DIMENSIONAL ARRAY. IT IS USED      *
C   TO APPLY LINE SEARCH ROUTINE                      *
C*****
    INTEGER MLAYR,MNODE,NLAYR,NNODE(0:MLAYR),
+ MAXNUM,NUM,I,J,K,KK,LL
    DOUBLE PRECISION A(MAXNUM),TG(MLAYR,MNODE,0:MNODE)
C

```

```

NUM=0
DO 10 K=1,NLAYR
  KK=NNODE(K)
  LL=NNODE(K-1)
  DO 20 J=1,KK
    DO 30 I=1,LL
      NUM=NUM+1
      A(NUM)=TG(K,J,I)
30    CONTINUE
20  CONTINUE
10  CONTINUE
C
RETURN
END
C*****
C*****
C  GIVEN A FUNCTION F AND ITS DERIVATIVE FUNCTION DF, AND  *
C  GIVEN A BRACKETING TRIPLET OF ABSCISSAS AX, BX, CX[SUCH  *
C  THAT BX IS BETWEEN AX AND CX AND F(BX) IS LESS THAN BOTH *
C  F(AX) AND F(CX)], THIS ROUTINE ISOLATES THE MINIMUM TO A  *
C  FRACTIONAL PRECISION OF ABOUT TOL USING A MODIFICATION OF *
C  BRENT'S METHOD THAT USES DERIVATIVES. THE ABSCISSA OF THE *
C  MINIMUM IS RETURNED AS XMIN, AND THE MINIMUM FUNCTION  *
C  VALUE IS RETURNED AS DBRENT, THE RETURNED FUNCTION VALUE. *
C*****
FUNCTION DBRENT(AX,BX,CX,F,DF,TOL,XMIN)
INTEGER ITMAX
DOUBLE PRECISION DBRENT,AX,BX,CX,TOL,XMIN,DF,F,ZEPS
EXTERNAL DF,F
PARAMETER(ITEM=100,ZEPS=1.0D-10)
INTEGER ITER
DOUBLE PRECISION A,B,D,D1,D2,DU,DV,DW,DX,E,FU,FV,FW,FX,OLDE,
+ TOL1, TOL2, U,U1,U2,V,W,X,XM
LOGICAL OK1,OK2
A=MIN(AX,CX)
B=MAX(AX,CX)
V=BX
W=V
X=V
E=0.
FX=F(X)
FV=FX
FW=FX
DX=DF(X)
DV=DX
DW=DX
DO 11 ITER=1,ITMAX
  XM=0.5*(A+B)
  TOL1=TOL*ABS(X)+ZEPS
  TOL2=2.*TOL1
  IF(ABS(X-XM) .LE. (TOL2 - .5*(B-A)))GOTO 3
  IF(ABS(E) .GT. TOL1) THEN
    D1=2.*(B-A)
    D2=D1
    IF(DW.NE.DX)D1=(W-X)*DX/(DX-DW)

```

```

IF(DV.NE.DX)D2=(V-X)*DX/(DX-DV)
U1=X+D1
U2=X+D2
OK1=((A-U1)*(U1-B).GT.0) .AND. (DX*D1 .LE. 0.)
OK2=((A-U2)*(U2-B).GT.0) .AND. (DX*D2 .LE. 0.)
OLDE=E
E=D
IF(.NOT. (OK1.OR.OK2))THEN
  GOTO 1
ELSEIF (OK1 .AND. OK2)THEN
  IF(ABS(D1).LT.ABS(D2))THEN
    D=D1
  ELSE
    D=D2
  ENDIF
ELSEIF (OK1) THEN
  D=D1
ELSE
  D=D2
ENDIF
IF(ABS(D) .GT. ABS(0.5*OLDE))GOTO 1
U=X+D
IF(U-A .LT. TOL2 .OR. B-U .LT. TOL2)D=SIGN(TOL1,XM-X)
GOTO 2
ENDIF
1  IF(DX.GE.0.)THEN
    E=A-X
  ELSE
    E=B-X
  ENDIF
  D=.5*E
2  IF(ABS(D) .GE. TOL1)THEN
    U=X+D
    FU=F(U)
  ELSE
    U=X+SIGN(TOL1,D)
    FU=F(U)
    IF(FU.GT.FX)GOTO 3
  ENDIF
  DU=DF(U)
  IF(FU.LE.FX)THEN
    IF(U.GE.X) THEN
      A=X
    ELSE
      B=X
    ENDIF
    V=W
    FV=FW
    DV=DW
    W=X
    FW=FX
    DW=DX
    X=U
    FX=FU
    DX=DU
  ELSE

```

```

        IF(U.LT.X)THEN
            A=U
        ELSE
            B=U
        ENDIF
        IF(FU.LE.FW .OR. W.EQ.X)THEN
            V=W
            FV=FW
            DV=DW
            W=U
            FW=FU
            DW=DU
        ELSEIF(FU .LE. FV .OR. V.EQ.X .OR. V.EQ.W)THEN
            V=U
            FV=FU
            DV=DU
        ENDIF
    ENDIF
11  CONTINUE
3   XMIN=X
    DBRENT=FX
    RETURN
    END
C*****

FUNCTION FINDBETA(TG,TG0,MLAYR,NLAYR,MNODE,
+ NNODE)
C*****
C   THIS ROUTINE IS TO FIND THE BETA ACCORDING TO   *
C   (2.4.8) -- (2.4.10).                             *
C   TG(MLAYR, MNODE, 0:MNODE) STORES TOTAL         *
C   GRADIENT                                         *
C*****
C*****
    END
    RETURN
    FINDBETA=SUM/SUM1
C
10  CONTINUE
20  CONTINUE
    FUNCTION FINDBETA(TG,TG0,MLAYR,NLAYR,MNODE,
+ NNODE)
C*****
C   THIS ROUTINE IS TO FIND THE BETA ACCORDING TO   *
C   (2.4.8) -- (2.4.10).                             *
C   TG(MLAYR, MNODE, 0:MNODE) STORES TOTAL         *
C   GRADIENT                                         *
C*****
C*****
    INTEGER MLAYR,NLAYR,MNODE,NNODE(0:MLAYR)
    DOUBLE PRECISION TG(MLAYR,MNODE,0:MNODE),
+ TG0(MLAYR,MNODE,0:MNODE),FINDBETA,SUM,SUM1
    INTEGER I,J,K,KK,LL
C
    SUM=0.D0
    SUM1=0.D0
    DO 10 K=1,NLAYR

```

```

        KK=NNODE(K)
        LL=NNODE(K-1)
        DO 20 J=1,KK
            DO 30 I=0,LL
                SUM = SUM + TG(K,J,I)*TG(K,J,I)
                SUM1 = SUM1 + TG0(K,J,I)*TG0(K,J,I)
30         CONTINUE
20     CONTINUE
10     CONTINUE
C
        FINDBETA=SUM/SUM1
        RETURN
        END
C*****

FUNCTION FINDE(P,T,MSAMP,NSAMP,MNODE,
+ W,MLAYR,NLAYR,NNODE,O,LAMDA,W0)
C*****
C THIS FUNCTION IS TO FIND THE PERFORMANCE *
C FUNCTION E(W) REF. (3.6.1). *
C FINDE -- THE PERFORMANCE VALUE. REF (3.6.1) *
C T(MSAMP,MNODE)-- THE DESIRED OUTPUT OF THE NET *
C W(MLAYR,MNODE,0:MNODE)-- WEIGHT MATRIX OF THE NET *
C O(MSAMP,MNODE)-- THE CALCULATED OUTPUT OF THE NET *
C LAMDA-- THE CONSTANT IN THE PENALTY TERM. *
C W0 -- THE CONSTANTS IN THE PENALTY. *
C*****
C
C INTEGER MSAMP,NSAMP,MNODE,MLAYR,NLAYR,
+ NNODE(0:MLAYR)
C DOUBLE PRECISION O(MSAMP,MNODE),T(MSAMP,MNODE),
+ W(MLAYR,MNODE,0:MNODE),LAMDA,W0,SUM,SUM1,FINDE,
+ P(MSAMP,MNODE)
C DOUBLE PRECISION N(MLAYR,MNODE),A(0:MLAYR,0:MNODE)
C INTEGER I,J,K,L,KK,LL
C
C CALCULATE THE PENALTY TERM.
C
        SUM =0.D0
        SUM1=0.D0
        DO 100 K=1,NLAYR
            KK=NNODE(K)
            LL=NNODE(K-1)
            DO 200 J=1,KK
                DO 300 I=0,LL
                    SUM=SUM+LAMDA*(W(K,J,I)**2/(W0**2+W(K,J,I)**2))
300         CONTINUE
200     CONTINUE
100     CONTINUE
C
C CALCULATE THE FIRST TERM
C
        DO 10 L=1,NSAMP
            CALL FORWARD(P,O,N,MLAYR,NLAYR,MNODE,A,NNODE,
+ W,L,MSAMP)

```



```

        DO 20 K=1,NNODE(NLAYR)
            SUM1=SUM1+(T(L,K)-O(L,K))**2
20    CONTINUE
10    CONTINUE
C
        FINDE = 0.5D0* (SUM +SUM1)
C
        RETURN
        END
C*****

SUBROUTINE FORWARD(P,O,N,MLAYR,NLAYR,MNODE,A,
+ NNODE,W,SN,MSAMP)
C*****
C THIS SUBROUTINE IS TO CALCULATE THE SUM OF *
C THE INPUTS OF A NEURON J IN LAYER K *
C PLEASE REFER TO (3.1.1) *
C N(MLAYR,MNODE)--STORES THE SUM OF INPUTS OF *
C NEURON J IN LAYER K *
C A(0:MLAYR,MNODE)--STORES THE OUTPUT OF *
C NEURON J IN LAYER K *
C A(0:MLAYR,0:MNODE) -- STORES THE INPUT DATA. *
C P(MSAMP,MNODE) -- IS THE INPUT DATA FROM ONE SAMPLE *
C T(MSAMP,MNODE) -- IS THE DESIRED OUTPUT DATA FROM ONE *
C SAMPLE *
C O(MSAMP,MNODE) -- IS THE OUTPUT CALCULATED FROM THE *
C NET. *
C W(MLAYR,MNODE,0:MNODE) -- THE WEIGHT OF THE NET. *
C SN -- THE SAMPLE INDEX. *
C*****
        INTEGER MLAYR,MNODE,NNODE(0:MLAYR),I,J,K,
+ NLAYR,L,SN,MSAMP,KK,LL
        DOUBLE PRECISION N(MLAYR,MNODE),A(0:MLAYR,0:MNODE),
+ W(MLAYR,MNODE,0:MNODE),SUM,P(MSAMP,MNODE),
+ O(MSAMP,MNODE)
C
C STORE INPUT DATA INTO A(0,MNODE)
C
        DO 100 I=1, NNODE(0)
            A(0,I)=P(SN,I)
100    CONTINUE
C
C STORE THE BIAS
C
        A(0,0) = -1.D0
C
C CALCULATE THE SUM OF THE INPUTS OF A NEURON J IN LAYER K
C
C LOOP OVER LAYER
C LOOP OVER LAYER
        DO 10 K=1,NLAYR
C LOOP OVER CURRENT NODE (TARGET)
            KK=NNODE(K)
            LL=NNODE(K-1)
            DO 20 J=1,KK
C LOOP OVER PRVIOUS NODE (SOURCE)

```

```

        SUM = 0.0D0
        DO 30 I=0,LL
            SUM = SUM + W(K,J,I)*A(K-1,I)
30     CONTINUE
C
C CALCULATE THE SUM OF 1 NEURON J IN LAYER K
C
        N(K,J) = SUM
C
C CALCULATE THE OUTPUT OF NEURON J IN LAYER K
C
        A(K,J) = SIGF(N(K,J))
20     CONTINUE
C
C THE BIAS
C
        A(K,0) = -1
10     CONTINUE
C
C STORED THE OUTPUT IN A(NNODE(NLAYR))
C
        KK=NNODE(NLAYR)
        DO 200 I=1, KK
            O(SN,I)=A(NLAYR,I)
200    CONTINUE
        RETURN
        END
C*****

SUBROUTINE FRPRMN(P,N,FTOL,ITER,FRET)
C*****
C   GIVEN A STARTING POINT P THAT IS A VECTOR OF LENGTH *
C   N, FLETCH-REEVES-POLAK-RIBIERE MINIMIZATION IS *
C   PERFORMED ON A FUNCTION FUNC, USING ITS GRADIENT AS *
C   CALCULATED BY A ROUTINE DFUNC. THE CONVERGENCE TOLERANCE *
C   ON THE FUNCTION VALUE IS INPUT AS FTOL. RETURNED *
C   QUANTITIES ARE P(THE LOCATION OF THE MINIMUM), ITER(THE *
C   NUMBER OF ITERATIONS THAT WERE PERFORMED),AND FRET(THE *
C   MINIMUM VALUE OF THE FUNCTION). THE ROUTINE LINMIN IS *
C   CALLED TO PERFORM LINE MINIMIZATIONS. *
C   PARAMETERS: NMAX IS THE MAXIMUM ANTICIPATED VALUE OF N; *
C   ITMAX IS THE MAXIMUM ALLOWED NUMBER OF ITERATIONS; EPS *
C   IS A SMALL NUMBER TO RECTIFY SPECIAL CASE OF CONVERGING *
C   TO EXACTLY ZERP FUNCTION VALUE. *
C*****
        INTEGER ITER,N,NMAX,ITMAX
        DOUBLE PRECISION FRET,FTOL,P(N),EPS,FUNC
        EXTERNAL FUNC
        PARAMETER(NMAX=50,ITMAX=200,EPS=1.0D-10)
C
C USES DFUNC,FUNC,LINMIN
C
        INTEGER ITS,J
        DOUBLE PRECISION DGG,GAM,GG,G(NMAX),H(NMAX),XI(NMAX)
        FP = FUNC(P)
        CALL DFUNC(P,XI)

```

```

DO 11 J=1,N
  G(J)=-XI(J)
  H(J)=G(J)
  XI(J)=H(J)
11 CONTINUE
DO 14 ITS=1,ITMAX
  CALL LINMIN(P,XI,N,FRET)
  IF(2.*ABS(FRET-FP) .LE. FTOL*(ABS(FRET)+ABS(FP)+EPS))RETURN
  FP = FUNC(P)
  CALL DFUNC(P,XI)
  GG =0.D0
  DGG =0.D0
  DO 12 J=1,N
    GG=GG+G(J)**2
    DGG=DGG+(XI(J)+G(J))*XI(J)
12 CONTINUE
  IF(GG .EQ. 0)RETURN
  GAM=DGG/GG
  DO 13 J=1,N
    G(J)=-XI(J)
    H(J)=G(J)+GAM*H(J)
    XI(J)=H(J)
13 CONTINUE
14 CONTINUE
C
  RETURN
  END
C*****

SUBROUTINE GETINPUTDATA(P,T,MNODE,DIMIN,DIMOUT,MSAMP,
+ NSAMP)
C*****
C THIS SUBROUTINE IS TO READ THE INPUT DATA FROM *
C TRAINING SAMPLE AND THE TARGET OUTPUT DATA. *
C*****
  INTEGER MNODE, DIMIN,DIMOUT,I,NSAMP,MSAMP,J
  DOUBLE PRECISION P(MSAMP,MNODE), T(MSAMP,MNODE)
C
  IN = 20
  OPEN(UNIT = IN, FILE = 'TRAIN.DAT',STATUS = 'OLD',IOSTAT=IOERR)
  IF(IOERR .NE. 0) THEN
    WRITE(*,10) IOERR
10  FORMAT(1X,'CANNOT OPEN NETWORK TRAINING DATA FILE(TRAIN.DAT)',
+ 15)
    STOP
  ENDIF
C READ IN NUMBER OF TRAINING SAMPLE
  READ(IN,*)NSAMP
C
  DO 100 J= 1, NSAMP
C READ IN THE INPUT DATA
  READ(IN,*)(P(J,I),I=1,DIMIN)
C
C READ IN THE DESIRED OUTPUT DATA(TARGET DATA)
  READ(IN,*)(T(J,I),I=1,DIMOUT)
100 CONTINUE

```

```

C
CLOSE (UNIT=IN)
C
RETURN
END
C*****

SUBROUTINE GETPP(PP,PP0,TG,MLAYR,NLAYR,MNODE,
+ NNODE, BETA)
C*****
C THIS SUBROUTINE IS TO CALCULATE METRIX PP. REF. *
C ALGORITHM 3.6.1 (3) AND (7). IT ADDS THE PREVIOUS *
C GRADIENT TO THE CURRENT GRADIENT ACCORDING TO *
C DIFFERENT BETA. REF.(2.4.8)-(2.4.10). STORED THE *
C WHOLE GRADIENT IN PP. *
C*****
INTEGER MLAYR,NLAYR,MNODE,NNODE(0:MLAYR)
DOUBLE PRECISION PP(MLAYR,MNODE,0:MNODE),BETA,
+ TG(MLAYR,MNODE,0:MNODE),
+ PP0(MLAYR,MNODE,0:MNODE)
INTEGER I,J,K,LL
C
C CALCULATE THE GRADIENT AND STORE IT IN PP
C
DO 10 K=1,NLAYR
  KK=NNODE(K)
  LL=NNODE(K-1)
  DO 20 J=1,KK
    DO 30 I=0,LL
      PP(K,J,I) = -TG(K,J,I) + BETA * PP0(K,J,I)
30    CONTINUE
20  CONTINUE
10  CONTINUE
C
RETURN
END
C*****

SUBROUTINE GRAD(SENSI,A,W,NLAYR, MLAYR,
+ NNODE,MNODE,G,W0,LAMDA)
C*****
C THIS SUBROUTINE IS TO CALCULATE THE *
C GRADIENT OF THE PERFORMANCE W.R.T WEIGHT. *
C REF. (3.6.6). *
C SENSI(MLAYR,MNODE)--THE SENSITVITY MATRIX. REF(3.6.12) *
C A(0:MLAYR,0:MNODE) -- THE OUTPUT OF A NEURON. REF(3.1.2) *
C W(MLAYR,MNODE,0:MNODE)-- THE WEIGHT MATRIX *
C G(MLAYR,MNODE,0:MNODE)-- THE GRADIENT OF THE NET *
C OF ONE SAMPLE DATE. *
C W0 -- THE CONSTANTS IN PENALTY TERM W0. *
C LAMDA -- THE CONSTANT IN THE PENALTY. *
C*****
INTEGER NLAYR, MLAYR,MNODE,I,J,K,LL,
+ NNODE(0:MLAYR)
DOUBLE PRECISION SENSI(MLAYR,MNODE),A(0:MLAYR,
+ 0:MNODE),W(MLAYR,MNODE,0:MNODE),

```

```

+ G(MLAYR,MNODE,0:MNODE),W0,LAMDA
C
C CALCULATE THE GRADIENT OF PERFORMACE FUNCTION W.R.T
C WEIGHTS ACCORDING TO (3.6.6)
C
C LOOP OVER LAYER
C
DO 10 K=1,NLAYR
  KK=NNODE(K)
  LL=NNODE(K-1)
  DO 20 J=1,KK
C
C BIAS TERM
C
  G(K,J,0)=SENSI(K,J)+
+   LAMDA * (W(K,J,0) * W0*W0)/((W0*W0 + W(K,J,0)**2)**2)
  DO 30 I=1,LL
    G(K,J,I)=SENSI(K,J) * A(K-1,I) +
+   LAMDA * (W(K,J,I) * W0*W0)/((W0*W0 + W(K,J,I)**2)**2)
30  CONTINUE
20  CONTINUE
10  CONTINUE
C
  RETURN
  END
C*****

SUBROUTINE INITG(NLAYR,MLAYR,NNODE,
+ MNODE,TG)
C*****
C THIS FUNCTION IS TO INITIALIZE THE TOTAL *
C GRADIENT TO 0. *
C*****
  INTEGER NLAYR, MLAYR,MNODE,I,J,K,KK,LL,
+ NNODE(0:MLAYR)
  DOUBLE PRECISION TG(MLAYR,MNODE,0:MNODE)
C
C INITIALIZE THE TOTAL GRADIENT TO 0
C AND NUMOFSAMPLE TO 0
C
DO 10 K=1,NLAYR
  KK=NNODE(K)
  LL=NNODE(K-1)
  DO 20 J=1,KK
    DO 30 I=0,LL
      TG(K,J,I)= 0
30  CONTINUE
20  CONTINUE
10  CONTINUE
C
  RETURN
  END
C*****

SUBROUTINE INIWEIGHT(WEIGHT, NLAYR, MLAYR,NNODE,
+ MNODE, SEED,NWEIG)

```

```

C*****
C  INITIALIZE THE WEIGHT OF INPUT LAYER          *
C                                          *
C  NLAYR -- THE NUMBER OF LAYER (INCLUDING      *
C          OUTPUT AND HIDDEN LAYERS).          *
C                                          *
C  NUMNODE(I) -- THE NUMBER OF NODE AT LAYER I. *
C  NUMNODE(0) -- THE NUMBER OF INPUT (NODE).   *
C  NUMNODE(NLAYR) -- NUMBER OF NODE IN OUTPUT LAYER *
C  NWEIG -- THE NUMBER OF WEIGHT                *
C                                          *
C  WEIGHT(LAYER, N, 0:N)-- LAYER IN THE LAYER INDEX *
C          N,M CORRESPONDING TO W(J,I), I.E.,   *
C          WEIGHT(LAYER, N, M) IS THE WEIGHT    *
C          OF THE CONNECTION FROM NODE M OF    *
C          THE (LAYER-1)TH LAYER TO NODE N OF  *
C          THE LAYERTH LAYER.                  *
C          WEIGHT(LAYER, N, 0) IS THE BIAS.     *
C                                          *
C*****
C  INTEGER MLAYR, MNODE,NWEIG
C  INTEGER I,J,K,FANIN,NNODE(0:MLAYR),NLAYR,KK,LL
C  DOUBLE PRECISION WEIGHT(MLAYR, MNODE,0:MNODE), TEMP,
C  + DRANDOM,SEED,TEMP1
C
C  GENERATE THE RANDOM NUMBER BETWEEN -0.5 TO 0.5
C
C      TEMP1 = SEED
C      TEMP = DRANDOM(TEMP1) - .5D0
C      NWEIG = 0
C
C  LOOP OVER LAYER
C
C      DO 10 K=1, NLAYR
C
C  CALCULATE THE FAN-IN OF THE LAYER.
C
C      KK=NNODE(K)
C      LL=NNODE(K-1)
C      FANIN = NNODE(K-1) + 1
C
C  LOOP OVER ALL NEURONS IN CURRENT LAYER
C
C      DO 20 J=1, KK
C
C  LOOP OVER ALL NEURONS IN PREVIOUS LAYER
C
C      DO 30 I = 0, LL
C          WEIGHT(K,J,I) = TEMP/FANIN
C          NWEIG = NWEIG + 1
30      CONTINUE
20      CONTINUE
10      CONTINUE
RETURN
END

```

```

C*****
SUBROUTINE LINMIN(FRET,P,T,MSAMP,NSAMP,
+ MNODE,W,TG,MLAYR,NLAYR,NNODE,O,LAMDA,W0)
C*****
C   GIVEN AN N-DIMENSIONAL POINT P(1:N) AND AN      *
C   N-DIMENSIONAL DIRECTION XI(1:N), MOVES AND      *
C   RESETS P TO WHERE THE FUNCTION FUNC(P) TAKES ON *
C   A MINIMUM ALONG THE DIRECTION XI FROM P AND     *
C   REPLACES XI BY THE ACTUAL VECTOR DISPLACEMENT  *
C   THAT P WAS MOVED. ALSO RETURNS AS FRET THE VALUE *
C   OF FUNC AT THE RETURNED LOCATION P. THIS IS    *
C   ACTUALLY ALL ACCOMPLISHED BY CALLING THE ROUTINES *
C   MNBRAK AND BRENT.                               *
C
C   REF "NUMERICAL RECIPIES"                         *
C*****
INTEGER MSAMP,NSAMP,MNODE,MLAYR,NLAYR,
+ NNODE(0:MLAYR)
DOUBLE PRECISION O(MSAMP,MNODE),T(MSAMP,MNODE),
+ W(MLAYR,MNODE,0:MNODE),TG(MLAYR,MNODE,0:MNODE),
+ P(MSAMP,MNODE),LAMDA,W0,TOL,FRET
INTEGER MSCOM,NSCOM,MNCOM,MLCOM,
+ NLCOM
PARAMETER(MLCOM=4,MNCOM=100,MSCOM=200)
INTEGER NNCOM(0:MLCOM)
DOUBLE PRECISION OCOM(MSCOM,MNCOM),TCOM(MSCOM,
+ MNCOM),WCOM(MLCOM,MNCOM,0:MNCOM),
+ LAMCOM,W0COM,TGCOM(MLCOM,MNCOM,0:MNCOM),
+ PCOM(MSCOM,MNCOM)
DOUBLE PRECISION AX,BX,FA,FB,FX,XMIN,XX,BRENT
COMMON /F1/NSCOM, NLCOM, NNCOM
COMMON /F2/PCOM,OCOM,TCOM,WCOM,TGCOM,LAMCOM,W0COM
INTEGER I,J,K,KK,LL
EXTERNAL FIDIM

C
C INITIALIZE THE PARAMETERS
C
  NSCOM=NSAMP
  NLCOM=NLAYR
  LAMCOM=LAMDA
  W0COM=W0
  TOL=1.0D-4
  DO 10 I=0,NLCOM
    NNCOM(I)=NNODE(I)
10  CONTINUE
  DO 50 I=1,NSCOM
    KK=NNCOM(NLCOM)
    DO 60 J=1,KK
      OCOM(I,J)=O(I,J)
      TCOM(I,J)=T(I,J)
      PCOM(I,J)=P(I,J)
60  CONTINUE
50  CONTINUE
C
  DO 20 K=1,NLCOM

```

```

        KK=NNODE(K)
        LL=NNODE(K-1)
        DO 30 J=1, KK
            DO 40 I=0, LL
                WCOM(K,J,I)=W(K,J,I)
                TGCOM(K,J,I)=TG(K,J,I)
40         CONTINUE
30     CONTINUE
20     CONTINUE
C
C     USES BRENT,F1DIM,MNBRAK
C
        AX = -1.0D0
        XX =1.0D0
        CALL MNBRAK(AX,XX,BX,FA,FX,FB,F1DIM)
        FRET = BRENT(AX,XX,BX,F1DIM,TOL,XMIN)
C
C CALCULATE THE TOTAL GRADIENT
C
        DO 70 K=1,NLCOM
            KK=NNODE(K)
            LL=NNODE(K-1)
            DO 80 J=1, KK
                DO 90 I=0, LL
                    TG(K,J,I)=XMIN*TG(K,J,I)
                    W(K,J,I)=W(K,J,I)+TG(K,J,I)
90         CONTINUE
80     CONTINUE
70     CONTINUE
C
        RETURN
        END

C*****
        FUNCTION F1DIM(X)
        INTEGER MSCOM,NSCOM,MNCOM,MLCOM,
+ NLCOM
        PARAMETER(MLCOM=4,MNCOM=100,MSCOM=200)
        INTEGER NNCOM(0:MLCOM)
        DOUBLE PRECISION OCOM(MSCOM,MNCOM),TCOM(MSCOM,
+ MNCOM),WCOM(MLCOM,MNCOM,0:MNCOM),
+ LAMCOM,W0COM,TGCOM(MLCOM,MNCOM,0:MNCOM),
+ PCOM(MSCOM,MNCOM)
        DOUBLE PRECISION XT(MLCOM,MNCOM,0:MNCOM)
C
C THE COMMON BLOCK
C
        COMMON /F1/NSCOM, NLCOM, NNCOM
        COMMON /F2/PCOM,OCOM,TCOM,WCOM,TGCOM,LAMCOM,W0COM
        DOUBLE PRECISION F1DIM,X
        EXTERNAL FINDE
C
C USES FINDE
C     USED BY LINMIN AS THE FUNCTION PASSED MNBRAK AND BRENT
C
        INTEGER I,J,K, KK, LL

```



```

DO 100 K=1,NLCOM
  KK=NNCOM(K)
  LL=NNCOM(K-1)
  DO 200 J=1,KK
    DO 300 I=0,LL
      XT(K,J,I)=WCOM(K,J,I)+X*TGCOM(K,J,I)
300    CONTINUE
200    CONTINUE
100    CONTINUE
  F1DIM = FINDE(PCOM,TCOM,MSCOM,NSCOM,MNCOM,XT,
+ MLCOM,NLCOM,NNCOM,OCOM,LAMCOM,W0COM)
  RETURN
  END
C*****

SUBROUTINE MNBRAK(AX,BX,CX,FA,FB,FC,FUNC)
C*****
C   THIS ROUTINE IS TO INITIALLY BRACKETING          *
C   A MINIMUM. REF " NUMERICAL RECIPIES            *
C   IN FORTRAN, THE ART OF SCIENTIFIC COMPUTING"    *
C   BY WILLIAM H. PRESS, ETC.                       *
C                                                    *
C   GIVEN A FUNCTION FUNC AND GIVEN DISTINCT        *
C   INITIAL POINTS AX AND BX, THIS ROUTINE          *
C   SEARCHES IN THE DOWNHILL DIRECTION (DEFINED     *
C   BY THE FUNCTION AS EVALUATED AT THE INITIAL     *
C   POINTS) AND RETURNS NEW POINTS AX, BX,         *
C   CX THAT BRACKET A MINIMUM OF THE FUNCTION.     *
C   ALSO RETURNED ARE THE FUNCTION VALUES AT      *
C   THE THREE POINTS, FA, FB AND FC.               *
C   PARAMETERS: GOLD IS THE DEFAULT RATIO BY       *
C   WHICH SUCCESSIVE INTERVALS ARE MAGNIFIED;      *
C   GLIMIT IS THE MAXIMUM MAGNIFICATION FOR        *
C   A PARABOLIC-FIT STEP.                          *
C*****
  DOUBLE PRECISION AX,BX,CX,FA,FB,FC,FUNC,GOLD,GLIMIT,TINY
  EXTERNAL FUNC
  PARAMETER (GOLD=1.618034D0,GLIMIT=100.D0,TINY=1.D-20)
  DOUBLE PRECISION DUM,FU,Q,R,U,ULIM
  FA=FUNC(AX)
  FB=FUNC(BX)
  IF(FB .GT. FA) THEN
    DUM=AX
    AX=BX
    BX=DUM
    DUM=FB
    FB=FA
    FA=DUM
  ENDIF
C
C   FIRST GUESS FOR C
C
  CX = BX +GOLD*(BX-AX)
  FC = FUNC(CX)
C
C INITIALIZE THE ITERATION COUNT

```

```

C
  ITER=0
1  IF(FB.GE.FC)THEN
    R=(BX-AX)*(FB-FC)
    Q=(BX-CX)*(FB-FA)
    U=BX-((BX-CX)*Q-(BX-AX)*R)/(2.*SIGN(MAX(ABS(Q-R),
+   TINY),Q-R))
    ULIM=BX + GLIMIT *(CX-BX)
    IF((BX-U)*(U-CX) .GT. 0) THEN
      FU = FUNC(U)
      IF(FU .LT. FC)THEN
        AX = BX
        FA = FB
        BX = U
        FB = FU
        RETURN
      ELSE IF(FU .GT. FB) THEN
        CX = U
        FC = FU
        RETURN
      ENDIF
      U = CX +GOLD*(CX - BX)
      FU = FUNC(U)
    ELSE IF((CX-U)*(U-ULIM) .GT.0)THEN
      FU = FUNC(U)
      IF(FU .LT. FC) THEN
        BX = CX
        CX = U
        U = CX + GOLD*(CX - BX)
        FB = FC
        FC = FU
        FU = FUNC(U)
      ENDIF
    ELSE IF((U - ULIM)*(ULIM - CX) .GE. 0)THEN
      U = ULIM
      FU = FUNC(U)
    ELSE
      U = CX + GOLD * (CX - BX)
      FU = FUNC(U)
    ENDIF
    AX = BX
    BX = CX
    CX = U
    FA = FB
    FB = FC
    FC = FU
    ITER = ITER +1
    GO TO 1
  ENDIF
  RETURN
  END
.....
SUBROUTINE NETPRINT(LAYER,MLAYR,NNODE,SEED,W0,
+   LAMDA,LAMDA2,METHOD)
C*****
C   THIS SUBROUTINE IS TO PRINT THE NETWORK ARCHITCTURE AND *
C   INITIAL PARAMETERS.                                     *

```

```

C*****
  INTEGER LAYER,MLAYR,NNODE(0:MLAYR),METHOD,NSAMP
  DOUBLE PRECISION SEED, TOL, W0, LAMDA,LAMDA2
C
  WRITE(*,10)LAYER
10  FORMAT(1X,'THE NUMBER OF LAYER IN THE NETWORK IS: ',I4)
  WRITE(*,20)NNODE(0)
20  FORMAT(1X,'THE INPUT DIMENSION IS ', I4)
  DO 30 I=1, LAYER
    WRITE(*,40)I,NNODE(I)
40  FORMAT(1X,'THE NUMBER OF NODE IN LAYER ',I4, 'IS ', I4)
30  CONTINUE
  WRITE(*,60)NNODE(LAYER)
60  FORMAT(1X,'THE OUTPUT DIMENSION IS ', I4)
  IF (METHOD .EQ. 0) THEN
    WRITE(*,100)
100  FORMAT(1X,'THE PENALTY METHOD IS USED')
  ELSE IF(METHOD .EQ.1) THEN
    WRITE(*,200)
200  FORMAT(1X,'THE STOP TRAINING METHOD IS USED')
  ELSE
    WRITE(*,300)
300  FORMAT(1X,'METHOD DATA ERROR')
    STOP
  ENDIF
C  PRINT THE PARAMETERS
  WRITE(*,50)SEED,W0,LAMDA,LAMDA2
50  FORMAT(1X,'THE SEED IS ',F10.4/1X,
+ /1X,'THE W0 IS ',F16.12/1X,'THE LAMDA IS ', F16.12,
+ /1X,'The LAMDA2 IS ', F16.12)
C
  RETURN
  END
C*****

SUBROUTINE NETSETUP(LAYER,MLAYR,NNODE,SEED,
+ W0,LAMDA,LAMDA2,METHOD)
C*****
C
C
C          *
C  THIS SUBROUTINE IS TO READ THE INPUT FILE AND SET UP  *
C  THE NETWORK ARCHITECTURE AND INITIALIZE PARAMETERS  *
C          *
C*****
  INTEGER LAYER, MLAYR, MAXNODE, NNODE(0:MLAYR),METHOD,
+ NSAMP
  DOUBLE PRECISION SEED,TOL,W0,LAMDA,LAMDA2
C
  IN=50
  OPEN(IN, FILE = 'NET.DAT', STATUS = 'OLD', IOSTAT= IOERR)
  IF(IOERR .NE. 0) THEN
    WRITE(*,10)IOERR
10  FORMAT('CANNOT OPEN NETWORK DATA FILE (NET.DAT), IOERR= ',I10)
    STOP
  ENDIF
C
C  READ IN THE NUMBER OF LAYER

```

```

C
  READ(IN,*)LAYER
C
C  READ IN THE NUMBER OF NODE IN EACH LAYER,THE NUMBER OF NODE IN
C  INPUT LAYER IS IN NNODE(0).
C
  READ(IN,*)(NNODE(I),I=0,LAYER)
C
C  READ IN METHOD, (0 FOR PENALTY METHOD, 1 FOR STOP TRAINING METHOD)
  READ(IN,*) METHOD
C
C  READ IN SEED NUMBER, TOLERANCE, W0 AND LAMDA LAMDA2
C
  READ(IN,*) SEED, W0, LAMDA
  READ(IN,*) LAMDA2
  WRITE(*, 1111) LAMDA, LAMDA2
1111  FORMAT('LAMDA=',F16.12, 'LAMDA2=',F16.12)
  CLOSE (UNIT = IN)
C
  RETURN
  END
C*****

SUBROUTINE PRINT3D(A,MLAYR,NLAYR,MNODE,
+ NNODE)
C*****
C  THIS SUBROUTINE IS TO COPY A ORIGINAL MATRIX TO NEW MATRIX. *
C  IT IS USED TO COPY TG. *
C*****
  INTEGER MLAYR,NLAYR,MNODE,NNODE(0:MLAYR)
  DOUBLE PRECISION A(MLAYR,MNODE,0:MNODE)
  INTEGER I,J,K,KK,LL
  DO 10 K=1,NLAYR
    KK=NNODE(K)
    LL=NNODE(K-1)
    DO 20 J=1,KK
      DO 30 I=0,LL
        WRITE(*, 100)K,J,I
100  FORMAT(1X,'LAYER # ',I5, 'J# ', I5, 'I# ',I5)
        WRITE(*, 200)A(K,J,I)
200  FORMAT(1X,'A VALUE: ',E15.7)
30  CONTINUE
20  CONTINUE
10  CONTINUE
  RETURN
  END
C*****

SUBROUTINE PRINTINPUTDATA(P,T,MNODE,DIMIN,DIMOUT,MSAMP,
+ NSAMP)
C*****
C  THIS SUBROUTINE IS TO PRINT THE INPUT DATA OF *
C  TRAINING SAMPLE AND THE TARGET OUTPUT DATA. *
C*****
  INTEGER MNODE, DIMIN,DIMOUT,I,NSAMP,MSAMP,J
  DOUBLE PRECISION P(MSAMP,MNODE), T(MSAMP,MNODE)

```

```

C
C PRINT IN THE INPUT DATA
WRITE(*,100)NSAMP
100 FORMAT(1X,'NUMBER OF SAMPLE IS: ',I5)
C
DO 200 J=1,NSAMP
WRITE(*,300)J
300 FORMAT(1X,'SAMPLE # ',I5)
DO 20 I= 1,DIMIN
WRITE(*,400)P(I,J)
400 FORMAT(1X,'THE INPUT DATA ARE: ',E15.7)
20 CONTINUE
C
DO 30 I= 1,DIMOUT
WRITE(*,500)T(I,J)
500 FORMAT(1X,'THE DESIRED OUTPUT DATA ARE: ',E15.7)
30 CONTINUE
200 CONTINUE
C
RETURN
END
C*****

FUNCTION DRANDOM(DL)
C*****
C THIS FUNCTION IS TO CREATE A RANDOM NUMBER BETWEEN *
C 0 TO 1 *
C*****
DOUBLE PRECISION DL, DRANDOM
C
10 DL=DMOD(16807.0D0*DL,2147483647.0D0)
DRANDOM=DL/2147483648.0D0
IF(DRANDOM.LE.0.0D0 .OR. DRANDOM.GE.1.0D0) GO TO 10
END
C*****

SUBROUTINE SENSITIVITY(SENSI,W,NLAYR,MLAYR,NNODE,
+ MNODE,T,OUT,N,SN,MSAMP)
C*****
C THIS SUBROUTINE IS TO CALCULATE THE *
C SENSITIVITY DEFINED IN (3.6.6). PLEASE REFER *
C TO (3.6.6)-(3.6.16) *
C SENSI(MLAYR,MNODE)--THE SENSITIVITY MATRIX. REF(3.6.12) *
C W(MLAYR,MNODE,0:MNODE)--WEIGHT MATRIX *
C T(MSAMP,MNODE)--THE DESIRED OUTPUT OF THE NET *
C OUT(MSAMP,MNODE)--THE CALCULATED OUTPUT OF THE NET *
C N(MLAYR,MNODE)--THE SUMMATION OF THE WEIGHT. REF(3.1.1) *
C SN -- THE SAMPLE INDEX. *
C*****
INTEGER NLAYR,MLAYR,NNODE(0:MLAYR),I,J,K,LL,
+ MNODE,MSAMP,SN
DOUBLE PRECISION W(MLAYR,MNODE,0:MNODE),
+ SENSI(MLAYR,MNODE),T(MSAMP,MNODE),
+ OUT(MSAMP,MNODE),N(MLAYR,MNODE),SUM
C
C CALCULATE THE SENSITIVITY OF FINAL LAYER (3.6.16)

```

```

C
  KK=NNODE(NLAYR)
  DO 10 I=1,KK
    SENSI(NLAYR,I) = -(T(SN,I) - OUT(SN,I))
    + * SIGFD(N(NLAYR,I))
10  CONTINUE
C
C  CALCULATE THE SENSITIVITY OF EACH LAYER STARTING
C  FROM THE FINAL LAYER. (3.6.12)
C
C  LOOP OVER LAYER
C
  DO 20 K=NLAYR-1,1,-1
    KK=NNODE(K)
    LL=NNODE(K+1)
    DO 40 I = 1,KK
      SUM =0.D0
      DO 30 J=1,LL
        SUM = SUM + SIGFD(N(K,I))*W(K+1,J,I)*SENSI(K+1,J)
30    CONTINUE
      SENSI(K,I) = SUM
40    CONTINUE
20  CONTINUE
C
  RETURN
  END
C*****

FUNCTION SIGF(X)
C*****
C  SIGMOID TRANSFER FUNCTION *
C  INPUT: DOUBLE PRECISION: X *
C  OUTPUT: DOUBLE PRECISION: SIGF *
C*****
  DOUBLE PRECISION X, SIGF
  SIGF = 1.D0 / (1.D0 + EXP(-X))
  RETURN
  END
C*****

FUNCTION SIGFD(X)
C*****
C  DERIVATIVE OF SIGMOID FUNCTION *
C  INPUT: DOUBLE PRECISION: X *
C  OUTPUT: DOUBLE PRECISION SIGFD *
C*****
  DOUBLE PRECISION X, SIGFD
  SIGFD= EXP(-X) / ((1.D0+EXP(-X))**2)
  RETURN
  END
C*****

SUBROUTINE SUMGRAD(G,NLAYR,MLAYR,NNODE,
+ MNODE,TG)
C*****
C  THIS FUNCTION IS TO SUM UP THE GRADIENTS *
```

```

C   OF EACH EPOCH.
C   TG(MLAYR,MNODE,0:MNODE) - STORES
C   THE TOTAL GRADIENTS OF NUMOFSAMPLE SAMPLES.
C   REF. ALGORITHM 3.6.1 (2.2)
C   G(MLAYR,MNODE,0:MNODE)--THE GRADIENT OF THE NET OF
C   ONE SAMPLE.
C   TG(MLAYR,MNODE,0:MNODE)-- THE TOTAL(SUMMATION) GRADIENT
C   OF ALL SAMPLES. REF. ALG(2.3)
C*****
C   INTEGER NLAYR, MLAYR,MNODE,I,J,K,KK,LL,
C   + NNODE(0:MLAYR)
C   DOUBLE PRECISION G(MLAYR,MNODE,0:MNODE),
C   + TG(MLAYR,MNODE,0:MNODE)
C
C   CALCULATE THE GRADIENT OF PERFORMACE FUNCTION W.R.T
C   WEIGHTS ACCORDING TO (3.6.6)
C   LOOP OVER LAYER
C
C   DO 10 K=1,NLAYR
C     KK=NNODE(K)
C     LL=NNODE(K-1)
C     DO 20 J=1,KK
C       DO 30 I=0,LL
C         TG(K,J,I)=TG(K,J,I) + G(K,J,I)
30     CONTINUE
20     CONTINUE
10    CONTINUE
      RETURN
      END
C*****

SUBROUTINE SUMWEIGHT(P,O,N,MLAYR,NLAYR,MNODE,A,
+ NNODE,W,SN,MSAMP)
C*****
C   THIS SUBROUTINE IS TO CALCULATE THE SUM OF
C   THE INPUTS OF A NEURON J IN LAYER K
C   PLEASE REFER TO (3.1.1)
C   N(MLAYR,MNODE)--STORES THE SUM OF INPUTS OF
C   NEURON J IN LAYER K
C   A(0:MLAYR,MNODE)--STORES THE OUTPUT OF
C   NEURON J IN LAYER K
C   A(0:MLAYR,0:MNODE) -- STORES THE INPUT DATA.
C   P(MSAMP,MNODE) -- IS THE INPUT DATA FROM ONE SAMPLE
C   T(MSAMP,MNODE) -- IS THE DESIRED OUTPUT DATA FROM ONE
C   SAMPLE
C   O(MSAMP,MNODE) -- IS THE OUTPUT CALCULATED FROM THE
C   NET.
C   W(MLAYR,MNODE,0:MNODE) -- THE WEIGHT OF THE NET.
C   SN -- THE SAMPLE INDEX.
C*****
C   INTEGER MLAYR,MNODE,NNODE(0:MLAYR),I,J,K,
C   + NLAYR,L,SN,MSAMP
C   DOUBLE PRECISION N(MLAYR,MNODE),A(0:MLAYR,0:MNODE),
C   + W(MLAYR,MNODE,0:MNODE),SUM,P(MSAMP,MNODE),
C   + O(MSAMP,MNODE)

```

```

C   STORE INPUT DATA INTO A(0,MNODE)

      DO 100 I=1, NNODE(0)
      A(0,I)=P(SN,I)
C   PRINT *,'SN= ',SN,'P(SN,I)= ',P(SN,I)
100  CONTINUE

C STORE THE BIAS
      A(0,0) = 1.D0

C
C   CALCULATE THE SUM OF THE INPUTS OF A NEURON J IN LAYER K
C
C   LOOP OVER LAYER
C LOOP OVER LAYER
      DO 10 K=1,NLAYR
C LOOP OVER CURRENT NODE (TARGET)
      DO 20 J=1,NNODE(K)
C LOOP OVER PRVIOUS NODE (SOURCE)
      SUM = 0.0D0
      DO 30 I=0,NNODE(K-1)
      SUM = SUM + W(K,J,I)*A(K-1,I)
30   CONTINUE
C CALCULATE THE SUM OF I NEURON J IN LAYER K
      N(K,J) = SUM
C   WRITE (*, 500) K,J,N(K,J)
C500  FORMAT(1X,'LAYER # ',I3,' NODE # ',I3,' N = ', F16.10)
C CALCULATE THE OUTPUT OF NEURON J IN LAYER K
      A(K,J) = SIGF(N(K,J))
C   WRITE (*, 400) K,J,A(K,J)
C400  FORMAT(1X,'LAYER # ',I3,' NODE # ',I3,' A = ', F16.10)
20   CONTINUE
C THE BIAS
      A(K,0) = 1.D0
10   CONTINUE

C   STORED THE OUTPUT IN A(NNODE(NLAYR))
      DO 200 I=1, NNODE(NLAYR)
      O(SN,I)=A(NLAYR,I)
200  CONTINUE
      RETURN
      END

C*****
C   THIS FUNCTION IS TO FIND THE PERFORMANCE
C   FUNCTION E(W) REF. (3.6.1) USING VALIDATION
C   DATA SET OR TEST DATA SET.
C   TEST--IS THE PERFORMANCE FUNCTION VALUE. REF(3.6.1)
C   T(MSAMP,MNODE)--THE DESIRED OUTPUT OF THE NET
C   W(MLAYR,MNODE, 0:MNODE)--WEIGHT MATRIX THAT
C   IS UPDATED BY TRAINING SAMPLES.
C   O(MSAMP,MNODE)--THE CALCULATED OUTPUT OF THE
C   NET
C   LAMDA--THE CONSTANT IN THE PENALTY TERM.
C   W0--THE CONSTANTS IN THE PENALTY.
C*****

```



```

FUNCTION TEST(MSAMP,MNODE,
+ W,MLAYR,NLAYR,NNODE,LAMDA,W0)

INTEGER MSAMP,NSAMP,MNODE,MLAYR,NLAYR,
+ NNODE(0:MLAYR)
DOUBLE PRECISION O(MSAMP,MNODE),T(MSAMP,MNODE),
+ W(MLAYR,MNODE,0:NNODE),LAMDA,W0,SUM,SUM1,FINDE,
+ P(MSAMP,MNODE)
DOUBLE PRECISION N(MLAYR,MNODE),A(0:MLAYR,0:NNODE)
INTEGER I,J,K,L,IN
C
C READ IN VALIDATION/TEST DATA SET
IN=20
OPEN(UNIT=IN,FILE='VALID.DAT',STATUS='OLD',IOSTAT=IOERR)
IF(IOERR .NE. 0) THEN
PRINT 11, IOERR
11 FORMAT('CANNOT OPEN NETWORK VALIDATION/TEST DATA
+ FILE(VALID.DAT)',I10)
STOP
ENDIF

READ(IN,*)NSAMP
DO 101 J=1,NSAMP
C READ IN THE INPUT DATA
READ(IN,*)(P(J,I),I=1,NNODE(0))
C READ IN THE DESIRED OUTPUT DATA
READ(IN,*)(T(J,I),I=1,NNODE(NLAYR))
101 CONTINUE
CLOSE(UNIT=IN)
C CALCULATE THE PENALTY TERM.
SUM =0.D0
SUM1=0.D0

C DO 400 K=1,NLAYR
C DO 500 J=1,NNODE(K)
C DO 600 I=0,NNODE(K-1)
C PRINT*,IN FINDE, W= ',W(K,J,I)
C600 CONTINUE
C500 CONTINUE
C400 CONTINUE

C DO 100 K=1,NLAYR
C DO 200 J=1,NNODE(K)
C DO 300 I=0,NNODE(K-1)
C SUM=SUM+LAMDA*(W(K,J,I)**2/(W0**2+W(K,J,I)**2))
C300 CONTINUE
C200 CONTINUE
C100 CONTINUE

C CALCULATE THE FIRST TERM

DO 10 L=1,NSAMP
CALL FORWARD(P,O,N,MLAYR,NLAYR,MNODE,A,NNODE,
+ W,L,MSAMP)
DO 20 K=1,NNODE(NLAYR)

```

```
        SUM1=SUM1+(T(L,K)-O(L,K))**2
20    CONTINUE
10    CONTINUE
C     PRINT *,'SUM= ',SUM
C     PRINT *,'SUM1= ',SUM1
C     PRINT *,'LAMDA= ',LAMDA
C     PRINT *,'W0= ',W0

TEST = .5D0*SUM1

RETURN
END
```

VITA

Zhong Xiang Luo

Candidate for the Degree of

Master of Science

Thesis: PENALTY METHODS TO REDUCE OVERFITTING  
IN ARTIFICIAL NEURAL NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Tian Men City, Hubei, China, on September 3, 1959, the son of Xian Bin Luo and Ying Tong.

Education: Received Bachelor of Engineering degree in Civil Engineering from Gezhouba Hydroelectric Engineering University, Yichang, China in July 1982. Received Master of Engineering degree in Hydroelectric Engineering from Wuhan Hydraulic and Electric Engineering University, Wuhan, China in July 1989. Completed the requirements for the Master of Science with a major in Computer Science at Oklahoma State University in December 1988.

Experience: Employed by Gezhouba Hydraulic Engineering University as a teaching faculty (July 1982 to July 1986), as an assistant professor (July 1989 to April 1993). Employed by Computer Science Department at Oklahoma State University as a teaching assistant from January 1996 to December 1996. Employed by W. R. Hess Oil Company as a software Engineer from August 1996 to March 1998. Employed by First Data Corporation as a software engineer from April 1998 to present.