OPTIMIZATION METHODS FOR TRAINING FEEDFORWARD NEURAL NETWORKS

By

HENDRA TIO

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1994

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, 1996

OPTIMIZATION METHODS FOR TRAINING

FEEDFORWARD NEURAL NETWORKS

Thesis Approved:

J Chandler

Thesis Advisor

Blayne E. Mayfield

W. Dick Stout

Thomas C. Collins

Dean Of Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my thesis advisor, Dr. John Chandler, for his guidance and support for my educational endeavors. It has been a long battle, but he was always there with encouraging words to keep me on track. Without his help it would not have been possible. My appreciation extends to the other members of my committee, Dr. Blayne Mayfield and Dr. Street for their helpfulness and support. Thanks also to Dr. Martin Hagan for his kindness in providing me with pertinent information.

I wish to dedicate this research to my parents, Thjung Sau Thin and Tio Sui Kim, for their encouragement, love and support, spiritual and financial, which enabled me to complete the degree.

Finally, I would like to thank my brother, Hidayat Tio, for his encouragement and support through this endeavor.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The invention of backpropagation [RuHi86] for training a multilayered network was a huge breakthrough in Artificial Neural Network (ANN) development. After a long search for a method to train such a network, which was earlier predicted to be dead-ended, a practical method was found. Multilayered ANNs show advantages over a single-layer network [Rose58]. Unlike a single-layered network, a multilayered network is not restricted to linearly separable problems [MinPa69]. Furthermore, a multilayered network can be trained to approximate most functions accurately by having a combination of transfer functions [Hagan95].

The backpropagation method operates in two sequences during training; a forward pass and a backward pass. In the forward pass, the input patterns are presented to the input layer, and the resulting activities flow to the output layer. In the backward pass, sensitivities are calculated to update the weights and biases from the output layer back to the input layer. The traditional backpropagation uses a steepest descent method for minimizing the mean squared errors [RuHi86]. This method, however, tends to be slow in convergence. For that reason, there has been much work on the improvement of the method. In this thesis, I will introduce some other numerical minimization (optimization) methods, as alternatives to the slow-converging steepest descent. The first minimization

technique is called the Conjugate Gradient (CG) method [Le82]. The CG method is characterized by

$$x_{k+1} = x_k - \alpha_k p_k$$

where k is the k-th iteration, x is the parameter vector, α is the scalar step size, and p is the (vector) search direction. The step size is computed by minimizing the performance index, F(x), along the search direction. The search direction p is initially set to the negative of the gradient of F(x) (steepest descent), and the next following iterations use search directions (p_k) computed using one of several possible formulas [Cheng93].

The second minimization method is the Scaled Conjugate Gradient (SCG) method of Moller [Moll93]. This method is different from the CG method. In the usual CG methods the step size is estimated by a line search; the SCG method employs a different technique in estimating the step size. The SCG method adopts the Levenberg-Marquardt idea (see below) to regulate the indefiniteness of the Hessian matrix.

The third method is the Levenberg method [Hagan95]. This method is a combination of steepest descent and the Gauss-Newton (GN) method [HaMen94]. In other words, this method guarantees convergence as in steepest descent. The disadvantage is that this method requires storage for the approximated Hessian matrix. That makes this method limited to networks with no more than several hundred parameters (weights and biases).

The final minimization method is the Levenberg-Marquardt method. This method uses a similar approach to the Levenberg method, except that this method scales the Hessian and the gradient vector to make the method scale-invariant.

In this thesis, a comparison study is done of several minimization methods for training feedforward neural networks. The primary concern is to focus on supervised training of the networks. The convergence rate, the reliability and the robustness of each method are investigated to exhibit the merit of the method.

In Chapter II, a brief review is done of an artificial neural network and the discovery of a way to train the network by backpropagation of error.

In Chapter III, a thorough overview is made of the minimization methods for training ANNs.

In Chapter IV, the numerical results for the different minimization methods will be shown.

Chapter V contains the conclusions and an outline of possible future work.

Finally, the appendices include the specifications of data sets and the source code that was implemented in this project.

CHAPTER II

LITERATURE REVIEW

An artificial neural network (ANN) is an abstraction of a real nervous system that consists of a collection of neurons communicating with each other via axon connections [Kung93]. Artificial neural networks do not approach the complexity of the brain, but they do share some similarities.

2.1. One-Layered ANN

The first development of the modern view of neural networks began in the 1940's with the accomplishments of Warren McCulloch and Walter Pitts [McPitt43]. McCulloch and Pitts showed that artificial neural networks could solve arithmetic and logical problems. Their work is still the basis for most neural networks today. In the McCulloch-Pitts model, the neurode computes the weighted sum of the input signals and compares the net weighted input to a threshold value. If the net input is greater than or equal to the threshold, the neurode outputs 1, otherwise it outputs -1 (Symmetrical Hard Limit transfer function).

The McCulloch-Pitts model is the simple model of a neurode that has become a standard today. It took Frank Rosenblatt to develop the model into the first trainable neural network [Rose58]. His perceptron training procedure could be used to allow a

network to learn a task of separating patterns into two categories. The neurode's weight is adjusted as follows:

$W_{new} = W_{old} + \in yx$
Where $\in$ = 1 if perceptron answer is correct,
and -1 if perceptron answer is wrong.
y is the perceptron answer.
W is the weight vector.
x is the input vector.

Rosenblatt proved that his perceptron learning rule will always converge to the correct network weights, if weights exist that solve the problem. Learning in a perceptron is simple, in which examples of proper behavior are presented to the network, and the network adjusts the weights to yield the pertinent solution. However, perceptron learning is limited to linearly separable problems [MinPa69]. For instance, a single-layered network cannot produce a single straight line to solve a classical XOR classification problem, because no such simple solution exists. Figure 2.1 shows how the categories are classified with two straight lines, where the open circles indicate patterns that output ones and closed circles indicate patterns that output zeros.

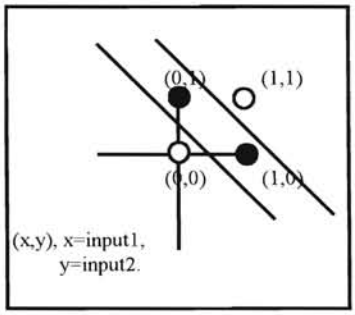

Figure 2.1. XOR classification problem that requires a minimum of two straight lines to classify the patterns correctly

Rosenblatt was aware of this limitation, and proposed a multilayered perceptron to overcome this limitation. However, the new proposed solution ended in a dead end, because Rosenblatt and Widrow were not able to modify their learning algorithm to train such a network.

2.2. Multi-Layered ANN

Multilayer perceptrons are feedforward neural networks with one or more layers (the hidden layer/s) between the output layer and the input layer [Hagan95]. The output of the first layer is the input to the second layer, the output of the second layer is the input to the third layer, and so on. Each layer may have a different number of neurons and also can have a different transfer function (activation function). Figure 2.2 shows a feedforward network, composed of three layers with R number of neurodes in each layer. These networks learn by adjusting the synaptic strengths (weights) between neurons. The knowledge acquired by the networks is from these weights and the interneuron connections. Multilayer networks are more powerful than single-layer networks. By having a combination of transfer functions in each layer, the networks can be trained to approximate most functions quite well. Single-layer networks cannot achieve this, and also single-layer networks can only solve linearly separable classification problems.

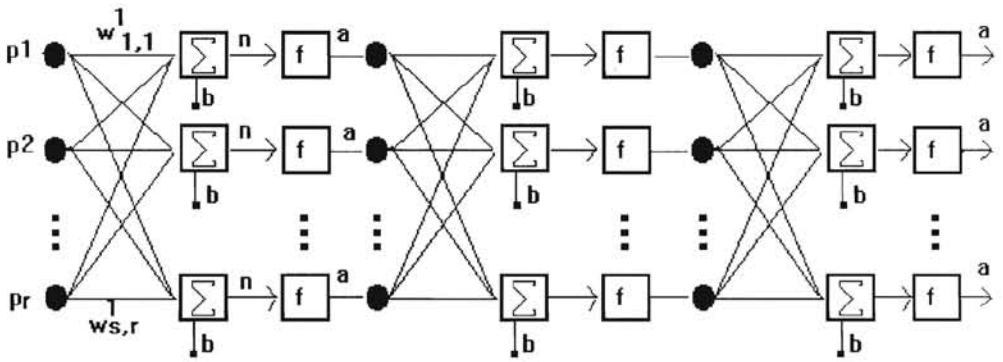

Figure 2.2. Feedforward network

After decades of silence in the development of multilayered ANNs, the problem was finally overcome by the discovery of the backpropagation algorithm for training multilayer perceptron networks [RuHi86]. Since the discovery of this algorithm and the development of multilayer ANN capabilities of handling real life problems, the backpropagation method has been widely used to train multilayer ANNs. Multilayer ANNs have become a popular tool in various areas of research. Their applications include signal processing, control, pattern recognition, speech production, speech recognition, automotive, business, medical, and many more areas.

2.3. Learning And Training of ANN

Neural networks learn to solve a problem; they are not programmed to do so. Learning and training are thus fundamental to nearly all neural networks. Learning is achieved by modifying the weights on the interconnections in the networks, and not by modifying the neurodes in the networks. Given that the transfer function is fixed, the neurode's output is determined by the incoming input signal and the weights on the input

connections to the neurode. Obviously, if the neurode is to learn to respond correctly to a given incoming input signal pattern, the only possibility is to adjust the weights on the connections to improve performance. In other words, learning in neural networks consist of making systematic changes to these weights in order to improve the network's response performance to desirable levels.

The difference between training and learning is that training is the procedure by which the networks learn, and learning is the end result of that procedure. Training is done by example. The following are three different ways of training:

- Supervised training: supervised training is a technique of training in which the networks are provided with an input pattern along with the desired output. The learning law for these networks typically computes the errors by calculating the difference between the desired output and the actual output that the networks generate. These errors are then used to modify the weights of the neurodes' interconnections. This type of training will be discussed in more depth later.
- Reinforcement training: There is a similarity to supervised training except that in this training the exact desired output is not given, instead a grade on how well the networks are performing is fed back. The grade, such as 'succeeded', 'failed', or 'too high', 'too low', can be given depending on how these are applicable to the networks.
- Unsupervised training: In this technique the networks are presented with a series of input patterns and are not given feedback on how well the networks are performing. Networks that use this training technique are most commonly used for categorizations and statistical applications, since the networks' results cannot be predetermined.

This paper concentrates only on supervised training. As mentioned earlier, in supervised training the desired output of the network is predetermined, and to calculate the error of the network is to compute the difference between the desired output and the actual output of the networks. Then this error is used to adjust the weight of the neurode's connections. After this adjustment the network is expected to generate an output that is closer to the desired output.

2.4. Performance Learning

Learning in neural networks consists of making systematic changes to the weights and biases in order to *optimize* the network's response *performance* to an acceptable level. There are two steps involved in this optimization process. The first step is to define and understand the quantitative measure of network performance, called the performance index. This performance index is characterized by the fact that it is small if the network performs well, and large if the network performs poorly. The second step is to adjust the network's weights and biases in order to reduce the performance index. In this case, our performance index parameters are the weights and biases.

In the optimization process, there are several methods that can be applied to reduce the performance index. To optimize the performance index, F(x), we will need to find the value of the vector x that minimizes F(x). Consider the following iterative equation:

$$x_{k+1} = x_k + \alpha_k p_k \qquad (1)$$

where x is the vector of performance index parameters (weights and biases), α is the scalar learning rate, and p is the search direction vector. According to this equation, if we begin with an initial guess at x_0, then we can update our guess iteratively. The optimization methods which will be presented here are distinguished by the choice of p_k (search direction) and α_k (step size). These methods evaluate the search direction, so that for a sufficiently small step the function will go downhill.

2.5. Backpropagation of Error (BP)

Backpropagation training modifies the [1] *delta rule* to make it appropriate for the multilayered network, which results in a generalized delta rule [RuHi86]. Backpropagation training operates in a two-step sequence during training. First, an input pattern is presented to the network's input layer, then the resulting activity flows through the network from layer to layer until the network's response is generated at the output layer (forward pass). In the second step, the network's output is compared to the desired output for the particular input pattern; then if it is not correct, an error is generated, which is passed and propagated backward through the network. This propagation step starts at the output layer and works back to the input layer, with the weights on the intralayer connections being updated as the error backpropagates (backward pass). Figure 2.3 shows the backpropagation of error for training a feedforward network.

---

[1] A popular method for training ANNs by Rumelhart, Hinton, and Williams. With this learning rule, patterns are repeatedly presented to the network, and the network's actual responses are compared to the responses that are desired for these patterns. This comparison involves computing an error term, which can be used to modify the pattern of connectivity in the network in such a way that the network's responses become more and more correct.

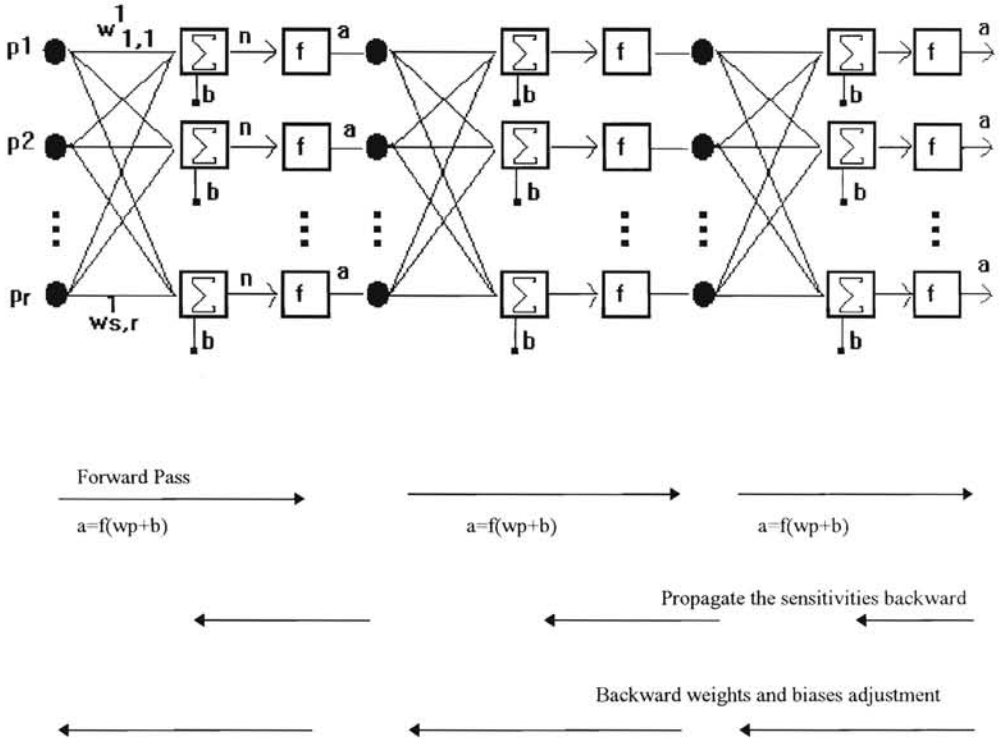

Figure 2.3. Backpropagation for training a feed-forward network

The following are the steps for backpropagation of error that will be used during the training session of an ANN:

1. Propagate the input forward through the network (forward pass)

   $a^0 = \mathbf{p}$

   $\mathbf{a}^{m+1} = \mathbf{F}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1})$ for m=0,1,2,...,M-1.

   $\mathbf{a} = \mathbf{a}^M$

   where $\mathbf{a}^m$ is the output vector of layer m, and $\mathbf{b}^m$ is the bias vector of layer m.

2. Propagate the sensitivities backward through the network

   $\mathbf{s}^M = -2\mathbf{F}^{\cdot M}(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$

$$\mathbf{s}^m = \mathbf{F}^{\cdot m}(\mathbf{n}^m)(\mathbf{W}^{m+1})\,\mathbf{s}^{m+1} \quad , \text{for } m = M-1,\ldots,2,1.$$

where M is the last layer. $\mathbf{F}^{\cdot m}$ is the derivative of the transfer function vector of layer m, $\mathbf{n}^m$ is the net output vector of layer m, and $\mathbf{W}^{m+1}$ is the weight vector of layer m+1.

3. Update the weights and biases; in this case we apply an incremental steepest descent method

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

where T is a matrix transpose operation.

The first step of BP is the forward pass, where inputs are presented to the network input layer, and the resulting activities flow from layer to layer. The second step is the backward pass, where computation of sensitivities is required to adjust the weights and biases from the output layer back to the input layer.

CHAPTER III

MINIMIZATION METHODS

3.1. Steepest Descent (SD)

In the steepest descent method, the search direction is set to the negative of the gradient ($-\nabla F(x_k)$). The following are the steps of the steepest descent method:

1. Select an initial guess for x_0, and set $k=0$.
2. Evaluate the gradient $\nabla F(x_k)$
3. Set the search direction, $p_k = -\nabla F(x_k)$.
4. Compute $\alpha_k > 0$ such that $F(x_k+\alpha_k p_k) = \min_\alpha\{F(x_k+\alpha p_k)\}$.
5. Substituting (2) into (1), the iterative equation becomes

$$x_{k+1} = x_k - \alpha_k \nabla F(x_k). \tag{2a}$$

6. Increment k and go to step 2.

In step 4, the step size (α_k) is determined by computing the minimum point of $F(x)$ along the search direction (p_k). In step 5, the evaluated search direction ($-\nabla F(x_k)$) is substituted into the iterative Equation (1) to produce an Equation (2a). Equation (2a) is called the standard steepest descent method. Although the method of steepest descent is useful for a large class of well-conditioned problems, it has been shown that the method can be extremely slow [Nash79].

One way to improve the algorithm is to add a "momentum" term to the standard method. This modification might help to smooth out the oscillations in the trajectory. After applying the momentum filter, the parameter changes becomes

$$\Delta x_{k+1} = \gamma \Delta x_k - (1-\gamma)\alpha \nabla F(x_k), \tag{2b}$$

and consequently the iterative Equation (2a) becomes

$$x_{k+1} = x_k - \gamma \Delta x_k - (1-\gamma)\alpha \nabla F(x_k). \tag{2c}$$

This modification of steepest descent can often provide faster convergence for some problems, but this modified method require an additional parameter (the momentum coefficient, γ). In most cases, the performance of the algorithm is sensitive to changes of this parameter.

3.2. Conjugate Gradient (CG) [Le82]

In Conjugate Gradient (CG) method the direction of minimization is always chosen such that the minimization steps in all previous directions are not spoiled. That is to say when the direction p_k is chosen and a line search is performed in this direction, leading to a point x_{k+1}, then the gradient $\nabla F(x_{k+1})$ at x_{k+1} must be perpendicular to $\nabla F(x_k)$, $\nabla F(x_{k-1})$, $\nabla F(x_{k-2})$,.., $\nabla F(x_0)$.

Various combinations of CG methods exist today. These combinations are generated by various choices of computation of new search directions, line searches, and restart criteria. The following are steps for CG methods:

1. Select an initial guess for x_0 and set $k=0$.
2. Compute the gradient of the error ($\nabla F(x_k)$).

3. Update the parameters,

$$x_{k+1} = x_k + \alpha_k p_k. \quad (3)$$

4. Compute the new search direction, p_k.

5. Reevaluate the parameters

$$x_{k+1} = x_k + \alpha_k p_k.$$

6. Increment k and go to step 4.

The initial operation in determining the search direction in the CG method is the same as the steepest descent method, where the search direction (p_k) is set equal to the negative of the gradient ($-\nabla F(x_k)$). In step 4, a new search direction p_k must be chosen. There are various approaches in determining the search directions, Fletcher-Reeves method [FleRe64], for instance, applying the idea of Hestenes-Stiefel method for choosing the search direction sequence

$$p_k = -\nabla F(x_k) + \beta_k p_{k-1}$$

where $\beta_k = (\nabla F^T(x_k)\ \nabla F(x_k)) / (\nabla F(x_{k-1})^T\ \nabla F(x_{k-1}))$. The well known Polak-Ribiere method [Nash79] differs from the Fletcher-Reeves method only in the choice of β_k. The formula to obtain β_k is

$$\beta_k = (\nabla F^T(x_k)\ (\nabla F(x_k)-\nabla F(x_{k-1}))) / (\nabla F(x_{k-1})^T\ \nabla F(x_{k-1}))$$

The formula for β_k in the Beale-Sorenson method [Beale72] is

$$\beta_k = (\nabla F^T(x_k)\ (\nabla F(x_k)-\nabla F(x_{k-1}))) / (p^T_{k-1}\ (\nabla F(x_k)-\nabla F(x_{k-1})))$$

Unlike the above three methods, the Perry method [Nash79] uses a different formula to choose the search direction p_k

$$p_k = -\nabla F(x_k) + ((y_k - \alpha_k p_k)^T\ \nabla F(x_k)) / ((y_k\ p_{k-1})\ p_{k-1})$$

where $y_k = \nabla F(x_k) - \nabla F(x_{k-1})$.

In steps 3 and 5, a line search must be carried out to locate the minimum point along the search direction (p_k). There are two types of line search method. The first type of line search uses only function evaluations, and the second type uses both function and gradient evaluations.

Since the conjugate gradient methods are designed to minimize a quadratic form in n steps, it is necessary to employ a method for continuing the iteration after n steps (n is the number of dimensions of the problem). To improve the rate of convergence on general nonlinear functions, especially when there are more than 2 variables, Fletcher & Reeves [FleRe64] recommend restarting the CG method with the steepest descent direction every n or (n+1) steps (traditional restart). Unlike the traditional restart, Beale's restart [Beale72] procedure takes into account the previous search direction in deriving the new search direction. The Shanno restart procedure [ShaPhu78] combines both Beale's and Powell's restart criteria into a double update scheme. His update scheme modified the gradient $\nabla F(x_k)$ with a positive definite matrix which best estimates the inverse Hessian without adding storage requirements. The variations of CG methods are due to the combination of the implementation of various search directions, line searches, and restart criteria.

The Conjugate Gradient (CG) method that is implemented in this project was originally published by Le for a general unconstrained optimization purpose. This CG method uses a different approach in determining a sequence of search directions, line

searches and restart criterion. The next sections show how Le's method resolves the choice of search directions, determining the choice of line search and restart criterion.

3.2.1. Search Direction

In determining the search direction, this method requires only three n-dimensional vectors. Both this CG method and most CG methods in general would generate the same sequences of p_k and x_k when minimizing a quadratic function using exact line search with given same initial conditions x_0 and $d_0 = -\nabla F(x_k)$. In this particular method the new search direction is obtained by evaluating the following formulas:

$$w_{k+1} = x_{k+1} + \beta_{k+1} \left(-\nabla F(x_{K+1}) / \| g_{K+1} \| \right), \tag{4}$$

$$z_{k+1} = w_{k+1} + \gamma_{k+1} p_k , \tag{5}$$

$$p_{k+1} = (z_{k+1} - x_{k+1}) / \| (z_{k+1} - x_{k+1}) \|. \tag{6}$$

In Equation (5), a line search is needed to determine γ_{k+1} that minimize $F(w_{k+1} + \gamma p_k)$, and in Equation (4), β_{k+1} is set larger than that required to minimize $F(x_{k+1})$ along the steepest descent direction $(-\nabla F(_{k+1}))$. The reason for this is that it could help to speed up the rate of convergence by passing over curved ridges. Figure 3.1 shows a couple of iterations of this method in a contour plot of a quadratic function.

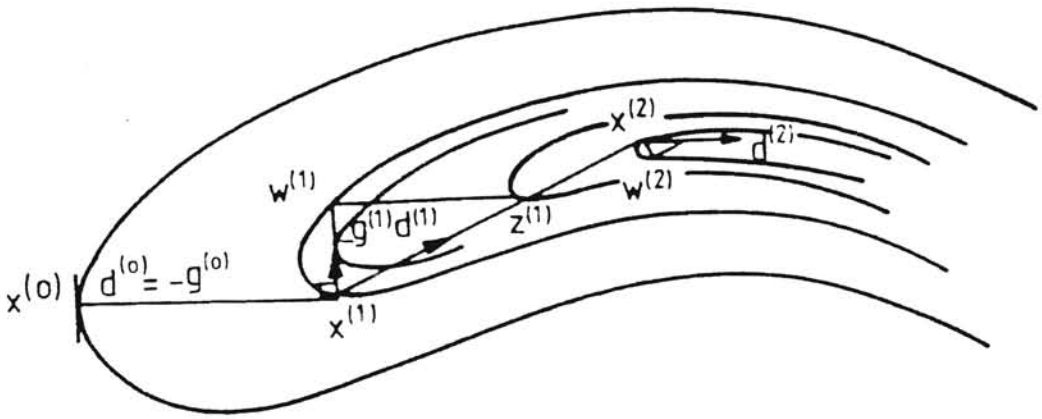

Figure 3.1. Iterations of Le's method in a quadratic function.
(Figure adapted from [Le82])

The graph shows that when β_{k+1} is set larger than necessary (twice the value necessary for the case of a quadratic form) to minimize $F(x_{k+1})$, it crosses over the third contour line and halts at the forth contour line.

3.2.2. Line Search

The line search plays an important role for most multidimensional optimization algorithms, and this also applies to Le's CG method. The motivation of a line search is to locate the minimum of a function in a specified direction. Le's CG method uses the first type of the line search, where only function evaluations are used. He stated that the first type of line search would become superior for large dimensional problems. Along with this type of line search, his method also incorporates an inexact line search method, where the minimum point need not to be located accurately. According to him, by having this method implemented, the inexact line search generally required fewer function evaluations per iteration than the exact line search. He also mentioned that to avoid the possibility of

instability of the algorithm caused by a relaxed accuracy of the line search, his method guarantees that the error function value is always reduced at the end of each iteration.

3.2.3. Restart Criterion

In Le's restart criterion he uses a method that measures the progress of a restart needed by the distance between the [2] *two most recent approximations*. The criterion is to restart the algorithm with the steepest descent direction ($-\nabla F(x_k)$) if the inequality

$$\| x_{k+1} - x_k \| \leq \| z_k - x_k \| \tag{7}$$

is satisfied for a certain number of consecutive iterations. Based on Le's numerical results and study, two consecutive iterations is a reasonably good criterion to be assigned to this method. According to Le's experimental results, the algorithm had shown significant improvement in performance, however, the method induced too many restarts near convergence. He explained that this phenomenon occured because near the optimum region, the objective function can usually be approximated by a quadratic function and β_k is set larger than required to minimize $F(x_k)$ along $-\nabla F(x_k)$ causing the inequality (7) to be repeatedly satisfied. To avoid this, he suggested to incorporating two additional conditions to regulate the restart mechanism. The first additional condition is to check for quadraticity of the objective function, and the second additianal condition is to check if the total number of iterations counted from the previous restart is less than n+1.

In summary, the restart mechanism is triggered when all the following conditions are satisfied:

---

[2] Let x_k be the current best approximation of the minimum point before an iteration k+1 and x_{k+1} be the next approximation at iteration k+1, then x_k and x_{k+1} are the two most recent approximations.

1. Inequality (7) holds twice successively.
2. The objective function does not appear to be well approximated by a quadratic function.
3. Total number of iteration counted from the previous restart is greater or equal to n+1.

Le's method is still inefficient on problems that are moderately ill-conditioned or worse, such as the Osborne 1 test problem [MillSpo78], which has a condition number of about 2400. Miller and Spooner [MillSpo78] have shown that at least one CG method is numerically unstable, and it may be that all common CG methods are unstable.

3.3. Scaled Conjugate Gradient (SCG) [Moll93]

Unlike a CG method where the step size is computed using a line search, the SCG method uses a different approach in estimating the step size. The step size can be obtained by estimating the term

$$s_k = \nabla^2 F(x_k)p_k \approx (\nabla F(x_k+\sigma_k p_k) - \nabla F(x_k))/\sigma_k, \quad (8)$$

where $0 < \sigma_k << 1$.

Since the SCG method only works for functions with positive definite Hessian ($\nabla^2 F(x_k)$, or A) matrices, this method needs to be refined to ensure that the Hessian is positive definite. This can be done by applying a model-trust region from the Levenberg-Marquardt method to the SCG. Consequently, after introducing the Langrange multiplier (λ_k), Equation (8) becomes

$$s_k \approx (\nabla F(x_k+\sigma_k p_k) - \nabla F(x_k))/\sigma_k + \lambda_k p_k, \quad (9)$$

where $0 < \sigma_k << 1$.

In adjusting the step size, this method uses the formula

$$\alpha_k = \mu_k / \delta_k = \mu_k / (p_k^T s_k + \lambda_k |p_k|^2). \quad (10)$$

Note that the value of λ_k is affecting the step size in such a way that the higher λ_k is, the smaller the step size is. The following are the steps for SCG:

1. Set $\sigma_0 = 0 < \sigma_0 << 10^{-4}$, $\lambda_0 = 0 < \lambda_0 << 10^{-6}$, $\lambda_{\sim k} = 0$, $p_0 = r_0 = -\nabla F(x_k)$, and success = false.

2. If success = true, then calculate the second order term,

   $\sigma_k = \sigma_k / |p_k|$,

   $s_k = (\nabla F(x_k + \sigma_k p_k) - \nabla F(x_k)) / \sigma_k$,

   $\delta_k = p_k s_k$.

3. Scale δ_k : $\delta_k = \delta_k + (\lambda_k - \lambda_{\sim k}) |p_k|^2$

4. If $\delta_k \leq 0$ (Hessian matrix is not positive definite), then

   $\lambda_{\sim k} = 2(\lambda_k - \delta_k / |p_k|^2)$

   $\delta_k = -\delta_k + \lambda_k |p_k|^2$

   $\lambda_k = \lambda_{\sim k}$.

5. Evaluate the step size,

   $\mu_k = p_k r_k$

   $\alpha_k = \mu_k / \delta_k$.

6. Calculate the comparison parameter,

   $\Delta_k = 2 \delta_k (F(x_k) - F(x_k + \alpha_k p_k)) / \mu_k^2$

7. If $\Delta_k \geq 0$ (successful in error reduction), then

   $x_{k+1} = x_k + \alpha_k p_k$,

$r_k = -\nabla F(x_{k+1})$,

$\lambda_k = 0$,

success = true,

if k mod n = 0 (restart is triggered) then

$p_{k+1} = r_{k+1}$

else

$\beta_k = (|r_{k+1}|^2 - r_{k+1}^T r_k) / \mu_k$,

$p_{k+1} = r_{k+1} + \beta_k p_k$

if $\Delta_k \geq 0.75$ then

$\lambda_k = 1/4\ \lambda_k$

else if $\Delta_k < 0$ then

$\lambda_{\sim k} = \lambda_k$,

success = false.

8. If $\Delta_k < 0.25$ then

$\lambda_k = \lambda_k + (\delta_k (1 - \Delta_k) / |p_k|^2)$.

9. go to step 2.

3.4. Levenberg (Lev I) [Hagan95]

The next class of optimization methods is called Levenberg (Lev I). The Levenberg method is based on the Gauss-Newton method that was designed for sum of squares nonlinear function minimization. The Levenberg method,

$$x_{k+1} = x_k - [J^T(x_k) J(x_k) + \mu_k I]^{-1} J^T(x_k) v(x_k) \quad (11)$$

has a mixture of Gauss-Newton (GN) and Steepest-Descent (SD) properties. The Levenberg method guarantees convergence, as does steepest descent, and has a linear convergence rate as does Gauss-Newton [Hagan95]. The drawback of the Levenberg method is the storage requirement. This method require the storage of the approximated Hessian matrix. The following are the steps for the Levenberg method:

1. Compute the sum of squared errors over all inputs.

$$F(x) = \Sigma (t_q - a_q)^T (t_q - a_q)$$

2. Compute the Jacobian matrix, J.

$$J(x) = \begin{pmatrix} \partial v_1(x)/\partial x_1 & \partial v_1(x)/\partial x_2 & \dots & \partial v_1(x)/\partial x_n \\ \partial v_2(x)/\partial x_1 & \partial v_2(x)/\partial x_2 & \dots & \partial v_2(x)/\partial x_n \\ \vdots & \vdots & & \vdots \\ \partial v_m(x)/\partial x_1 & \partial v_m(x)/\partial x_2 & \dots & \partial v_m(x)/\partial x_n \end{pmatrix}$$

where $v_1 \dots v_N$ (error vector) are the errors generated by each input-output pair.

3. Solve the following iterative equation:

$$x_{k+1} = x_k - [J^T (x_k) J(x_k) + \lambda_k I]^{-1} J^T (x_k) v(x_k) \qquad (13)$$

(Solve without actually inverting any matrix).

4. Recompute the sum of squared errors using $x_k + \Delta x_k$. If the new sum of squares is smaller than the one computed in step 1, then divide λ by μ, and go to step 1. If the new sum squares is not reduced, then multiply λ by μ and go to step 3.

Unfortunately, the levenberg method is not scale-invariant. That is, a change in the units of any component of the x vector can change completely the convergence of the

algorithm. Any [3] *algorithm* that is not scale-invariant is inherently inferior, although that is not to say that it will be slower in every case.

In order to obtain a Levenberg method that works for ANN, we need to modify the Jacobian computation in step 2 by redefining the error vector in each row of the Jacobian matrix as

$$V^T = [v_1\ v_2 \ldots v_N]$$

$$= [e_{1,1}\ e_{2,1} \ldots e_S m_{,1}\ e_{1,2} \ldots e_S m_{,2} \ldots e_{1,N} \ldots e_S m_{,N}].$$

Notice that v_1 is redefined as $e_{1,1}\ e_{2,1} \ldots e_S m_{,1}$, where the first element of the subscript denotes the error produced by neuron x in the final layer, and the second element of the subscript is the error produced by the individual data point (input-output pair). For instance, $e_{x,y}$ is the error produced by neuron x in the final layer of the y-th data point. Since the row of the Jacobian matrix involving the computation of the individual error produced by each neuron in the final layer, the sensitivity calculations for the Levenberg method have to be reconstructed to adapt to this procedure. The reconstruction can be made by modifying the existing method of backpropagation computation at the final layer. For Levenberg sensitivities at the final layer, we have

$$\mathbf{S}_q^{\sim M} = -\mathbf{F}^{\cdot M}(\mathbf{n}_q^M),$$

where M is the last layer, $\mathbf{F}^{\cdot M}$ is the derivative of the transfer function vector of the final layer (layer M), and $\mathbf{n}_q^M$ is the net output vector of the last layer for the q-th input-output pair. As for the columns of the Jacobian matrix, they can be propagated using

$$\mathbf{S}_q^{\sim m} = -\mathbf{F}^{\cdot m}(\mathbf{n}_q^m)(\mathbf{X}^{m+1})\,\mathbf{S}_q^{\sim m+1},$$

[3] The Gauss-Newton method is scale-invariant; steepest descent is not.

where m denotes the m-th layer and **X** is the parameters vector. The next step would be augmenting the individual sensitivity vectors using

$$S^{\sim m} = [S_1^{\sim m}|S_2^{\sim m}|\ldots|S_q^{\sim m}]$$

Then, we can compute the elements of the Jacobian matrix with

$$[J]_{h,l} = \partial v_h/dx_l = \partial e_{k,q}/\partial w_{i,j} = (\partial e_{k,q}/\partial n_{i,q}^{m}) (\partial n_{i,q}^{m}/\partial w_{i,j}^{m}) = s^{\sim m}_{i,h}\, a^{m-1}_{j,q} \qquad \text{(for weights)}$$

$$[J]_{h,l} = \partial v_h/dx_l = \partial e_{k,q}/\partial b_{i,j} = (\partial e_{k,q}/\partial n_{i,q}^{m}) (\partial n_{i,q}^{m}/\partial b_{i,j}^{m}) = s^{\sim m}_{i,h} \qquad \text{(for biases)}$$

where a^{m-1} is the output of layer m-1, $h = (q-1)S^M + k$, and $e_{k,q}$ is the error produced by neuron k in the final layer at the q-th input-output pair.

In Step 4, μ is set to a constant number that is large enough to increase λ. Here, we can notice that in Equation (6), if λ is decreased to zero the iterative equation becomes the Gauss-Newton step, and if λ is increased to a large number, then the equation become a steepest descent step.

3.5. Levenberg-Marquardt

The scaling of the variables in an optimization problem is often crucial to the successful functioning of the methods. A well-scaled [Marq59] problem means that similar changes in the variables lead to similar changes in the objective function. In a contour plot of a well-scaled function, the contour lines would not deviate too far from concentric circles [Marq59]. In this case, we would hope that the method of steepest descent would work satisfactorily. Note that when the contours are exactly concentric circles, the steepest descent method converges in one iteration.

Marquardt in his 1963 paper [Marq63] introduced a method that uses an approach similar to Levenberg, except that his method scaled the Hessian (A*) and the gradient vector (g*). As a result, Marquardt's method is scale-invariant.

The Levenberg method computed the solution for δ from

$$(J^TJ + \lambda I)\,\delta = -J^Tv \tag{13}$$

where J^TJ is the Hessian, J^Tv is the gradient, and δ is the vector of changes of parameters. The use of scaling parameters

$$x' = Dx, \tag{14}$$

where D is a diagonal matrix having positive diagonal elements, implies the transformation of the Jacobian as follows:

$$J' = J\,D^{-1} \tag{15}$$

and Equation (12) becomes

$$[(J')^TJ' + \lambda I]\,\delta = -(J')^T v \tag{16a}$$

$$= (D^{-1}J^TJ\,D^{-1} + \lambda I)\,D\delta = -D^{-1}J^Tv \tag{16b}$$

$$= (J^TJ + \lambda D^2)\,\delta = -J^Tv, \tag{16c}$$

where the $D^2_{ii} = (J^TJ)_{ii}$. By simplifying the equation, the scaling can be done implicitly by solving (16c) instead of (16a) [Nash79].

3.6. Incremental Levenberg [LuHa94]

The previously mentioned Levenberg method is the batch mode version of Levenberg, where all the inputs are presented to the network before the parameters (weights and biases) are updated. At this point, we want to generate another algorithm

that works incrementally, where the parameters are updated after each new data point is presented to the network. In order to do this, examine the parameter updating formula in the Gauss-Newton method (GN):

$$x_{k+1} = x_k + [J^T(x_k)\, J^T(x_k)]^{-1}\, J^T(x_k)\, v(x_k) \tag{17}$$

The next step is to convert Equation (16) into an iterative form of the GN algorithm, by rewriting the above equation to:

$$x_{k+1} = x_k + P^{-1}_{k+1}(x_k)\, \nabla V_{k+1}(x_k). \tag{18}$$

The subscript k determines the computation after the k-th data point is presented, $P^{-1}_{k+1}(x_k)$ represents $[J^T(x_k)\, J^T(x_k)]^{-1}$ (the Hessian), and $\nabla V_{k+1}(x_k)$ represents $J^T(x_k)\, v(x_k)$ (the gradient). Subsequently, we need to figure out the iterative form of P (Hessian) and ∇V (gradient) [LuHa94].

3.6.1. Iterative form of the Hessian

The iterative form of P can be written as:

$$P_{k+1}(x_k) = P_k(x_k) + [\nabla v_{k+1}(x_k)]\, [\nabla v_{k+1}(x_k)]^T, \tag{19}$$

where $\nabla v_{k+1}(x_k)$ is the gradient of the error evaluated at the (k+1)-th data point. Based on Equation (18), it would be impractical to compute $P_k(x_k)$, because this requires the recomputation of all derivatives at x_k. Therefore, we will use the term $P_k(x_{k-1})$ and a "forgetting factor" (exponential smoothing or filtering factor) will be appended to Equation (18):

$$P_{k+1}(x_k) = f_k\, P_k(x_{k-1}) + [\nabla v_{k+1}(x_k)]\, [\nabla v_{k+1}(x_k)]^T, \tag{20}$$

where $0<f_k<1$. For every iteration, the forgetting factor will be updated as follows:

$$f_k = \omega f_{k-1} + 0.01 \text{ [LuHa94]}. \quad (21)$$

3.6.2. Iterative form of the Gradient

In a similar way, the gradient term ($\nabla V_{k+1}(x_k)$) can be expressed as:

$$\nabla V_{k+1}(x_k) = \nabla V_k(x_k) + [\nabla v_{k+1}(x_k)]\, v_{k+1}(x_k). \quad (22)$$

Here we have to assume that V is approximately minimized at each step; on that account the value of $\nabla V_k(x_k)$ should be very close to zero. By omitting this term, we have

$$\nabla V_{k+1}(x_k) = [\nabla v_{k+1}(x_k)]\, v_{k+1}(x_k). \quad (23)$$

Now Equation (22) has become a sound iterative form of the gradient term in the Levenberg method.

The following are the complete steps for the incremental Levenberg method [LuHa94] :

1. Evaluate the steepest descent direction:

$$\Delta x_{sd} = \nabla V_{k+1}(x_k) = [\nabla v_{k+1}(x_k)]\, v_{k+1}(x_k) \quad (24)$$

2 Evaluate the Gauss-Newton direction:

$$P^{-1}(k+1) = 1/f\,[P^{-1}(k) - (P^{-1}(k) v_{k+1}\, v^T_{k+1}\, P^{-1}(k)) / (f + v^T_{k+1}\, P^{-1}(k) v_{k+1})] \quad (25)$$

$$\Delta x_{gn} = P^{-1}(k+1)\, \nabla V_{k+1}(x_k) \quad (26)$$

where f is the forgetting factor,

$p^{-1}(k+1) = p^{-1}{}_{k+1}(x_k)$,

and $v_{k+1} = \nabla v_{k+1}(x_k)$.

3. Solve the parameter update scheme in the incremental Levenberg method:

$$\Delta x = (1/(1+\lambda)^2)\, (\Delta x_{gn}) + (\lambda/(1+\lambda)^2)\, (\Delta x_{sd}) \quad (27)$$

Note that $P^{-1}(0)$ has to be initialized before the iteration. This parameter value

($P^{-1}(0)$) can be adjusted and the effect on performance can be investigated. Likewise, the variable λ is multiplied by μ when the squared error increases, and is divided by μ when the squared error decreases. The values of λ and μ are set as follow:

$0 < \lambda << 1$,

$0 < \mu < 1$.

A typical value for λ is between 0.001 and 0.1, and a typical value for μ is between 0.85 and 0.95.

CHAPTER IV

METHOD OF ANALYSIS AND DISCUSSION OF RESULTS

4.1. Univariate and Multivariate Data

There are two distinct data set classisfications based on the size of the input in an input-output pair. These two distinct classisfications are univariate and multivariate data. In multivariate data sets, each object(output) is described with more than one variable (inputs). On the other hand, univariate data set is only having a single variable (input) to describe each object (output).

4.2. Classification and Modeling Problem

4.2.1. Classification Problem

One of the most frequently used operations in handling multivariate data in neural network is the classification system. The classification system has a task of sorting objects into a simple sets of categories, or into subclasses within classes. One sinple example for classification problem is the XOR, where the objective of the problem is to classify correctly both of the inputs to the designated classes. This 2-variate problem is shown in Table 4.1.

1st Input	2nd Input	Output
0	0	0
1	0	1
0	1	1
1	1	0

Table 4.1. XOR Classification problem

In training ANN for classification problems, there will always be a question of what range of computed output value is to be considered *close enough* to the desired output. In other words the correctness criterion has to be defined. The comparison to be made later in this chapter employs a 40-20-40 criterion. This criterion says that if the computed output exceeds 40% beyond the desired output then it is treated as incorrect classification, if the computed output is in the range of 40% of the desired output then it is treated as correct classification. Figure 4.1 shows the 40-20-40 criterion for a

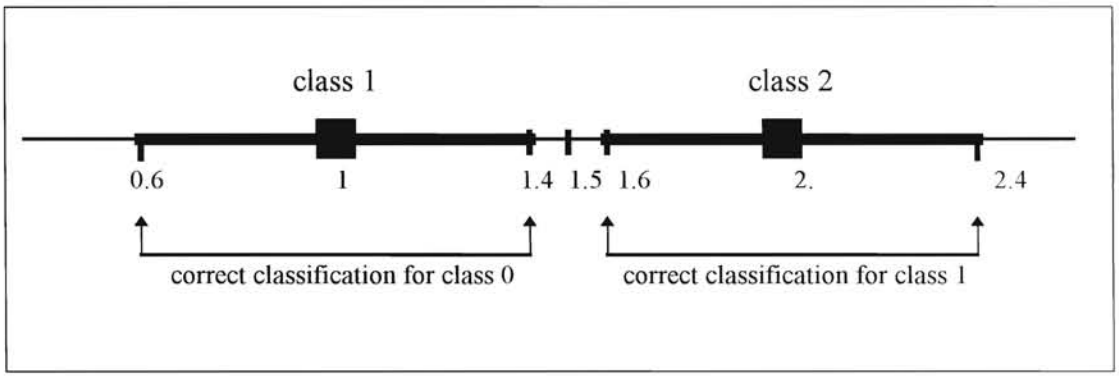

Figure 4.1. 40-20-40 correctness criterion for two classifiers problem, such as XOR.

classification problem with two distinct classes, where the thick line that lies between 0.6...1.4 shows a correct classification for the data pair that belongs to class 1, and the thick line lies between 1.6...2.4 exhibits a correct classification for the data pair that

belongs to class 2. This criterion is widely used [ZuGas93] because it does not require extreme accuracy in the output, but does require that the output be distinct enough for some amount of noise immunity.

The other adjustment that needs to be made for this classification problem (for most networks that employ a nonlinear transfer function in the final layer) is the scaling of the output. Due to the nonlinearity of the transfer function, it is better to scale the entire output to fall between zero and one. This help the reachability of the nonlinear transformation produced by the network. Zupan and Gasteiger [ZuGas93] suggested to scale the entire output to lie between 0.2 and 0.8.

4.2.2. Modeling Problem

A classification operation produces a discrete answer, such as a value identifying the input object with one of the several classes. However, modeling requires a system that produces a continuous answer for each set of input values. Curve fitting is classified into this class of modeling problems.

4.3. Utilization of a Validation Set to Overcome Overtraining Problem

Overtraining can be explained as a consequence of parameter redundancy; that is, when a network has more parameters than are needed for solution of the problem. This kind of network may be able to generalize to the training set but fails to handle a slightly different set of data.

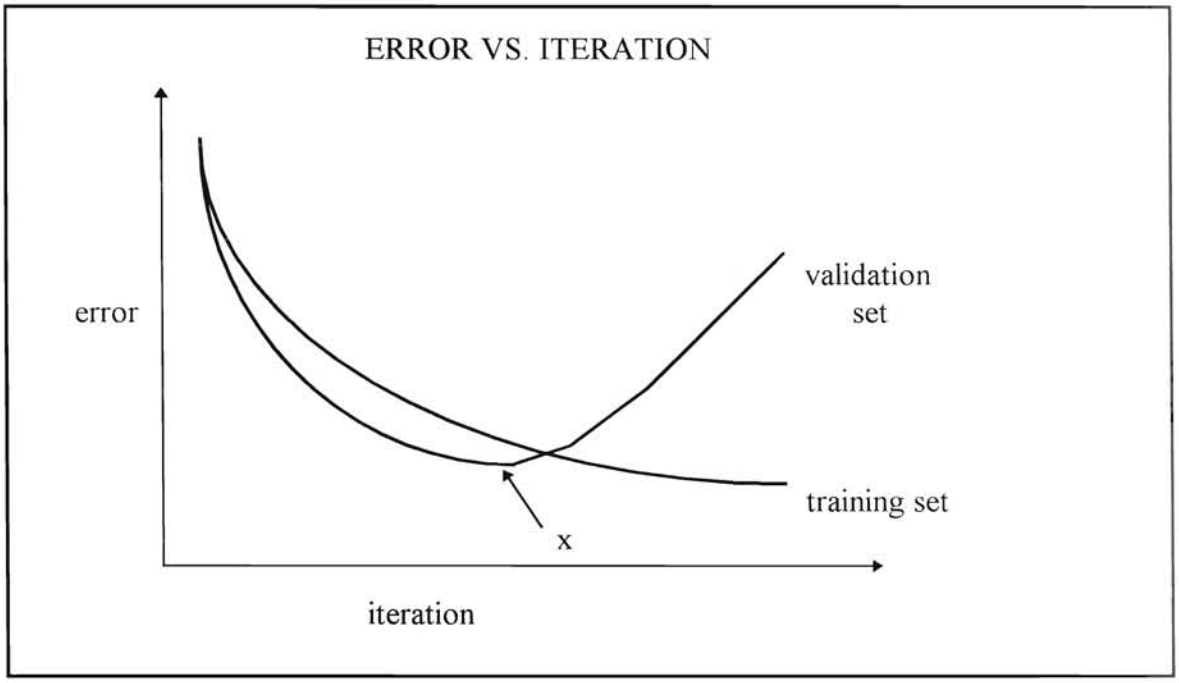

Figure 4.2. Training and validation error

In Figure 4.2, we can notice that at a point x the error in the training set reduces as the iteration increases, however, the error produced by the network using the different data (validation set) set starts to increase. This phenomenon is called overtraining. In order to overcome this problem, the network needs to be accomodated with a validation set, in which stopped training can be triggered when the above phenomenon occurs.

4.4. Selection of Data Sets

There are three kind of data sets that are used to test each optimization method, they are *lymphography*, *waveform* and *financial* data sets. The lymphography and the waveform data sets are classified into the area of *classification* problems, and the financial data is classified into the area of *modeling* problems. The number of instances for training, validating, testing the network and the rest of the specifications are defined in Table 4.2.

Name	Type	# Input	# Output	# Training Inst.	# Validating Inst.	# Testing Inst.
Lymphography	Classification	18	1	108	24	14
Waveform	Classification	21	1	188	187	125
Financial	Modeling	2	1	1000	0	0

Table 4.2. Data sets specifications.

Table 4.2. shows that the first two data sets are multivariate classification problems and the last data set is the time series data which is classified as a multivariate modeling problem.

4.5. Parameter Choice in the SDBP and Incremental Levenberg Method

The SDBP method (incremental and batch approaches) and the incremental Levenberg method must be supplied with a choice of parameters. This parameter assortment affects the performance of the methods. In the incremental SDBP method, for instance, the parameters required are the momentum constant and the learning rate, and batch SDBP only requires the momentum constant.

Method	parameter1	parameter2
Batch SDBP	Gamma = 0.1, 0.5, 0,9	-
Inc. SDBP	Gamma = 0.1, 0.5, 0.9	Eta = 0.1, 0.2, 0.5
Inc. Levenberg	p0 = 1, 1000, 100000	f = 0.88, 0.92, 0.95

Table 4.3. Parameters choice for Batch SDBP, Incremental SDBP, and Incremental Levenberg.

The reason for not having the learning rate as a parameter in the batch SDBP is because of the implementation of line search in this method (this method possesses a variable stepsize). The parameter choice in the incremental Levenberg method are the

initial value for the covariance matrix (p_0) and the forgetting factor (f). Table 4.3 exhibits the choices of reasonable parameters to be assigned to these methods for the experiments in the next section.

4.6. Results and Analysis

In this comparison study, several network architectures and [4] relevant parameters of each of the minimization methods were tested using three different data sets; they were lymphography, waveform and financial (time series) data. This study also showed how different data sets produce a different optimal [5] network structures. This is done by testing various combinations of network architectures using those three sets of data, and the effect of each network structure were recorded and analyzed. Appendix A contains the combinations of network structures used in solving all three data sets.

The first test data is the lymphography data set. In this experiment a validation set is used as the stopping criterion. Table 4.4 shows time consumed during training and percentage of correct classification by each method. The results shown in Table 4.4 are based on an average of ten runs for each method with a combination of network structures, and Table 4.5 shows the best case of each method.

---

[4] Parameter choices are only applied to both the incremental and batch SDBP and the Levenberg.

[5] Network structures are defined as various combination of network architectures and parameters (where applicable).

	Time			% Correct	% Correct
Method	Total	User	System	in Training	inTesting
Batch SDBP	1088.66	1088.50	0.16	82.10	63.02
Inc. SDBP	1250.91	1259.22	0.69	81.45	63.42
L-M	44.79	44.74	0.05	83.34	64.28
Batch Levenberg	43.74	43.72	0.02	81.95	64.28
Inc. Levenberg	474.60	463.10	0.25	83.52	65.71
LCG	103.57	103.49	0.08	81.48	67.86
SCG	117.17	116.90	0.27	80.56	64.28

(Time is in units of seconds)

Table 4.4. Results from the lymphography data set.

	TIME		
Method	Total	User	System
Batch SDBP	548.93	548.86	0.07
Inc. SDBP	535.66	535.49	0.17
LM	29.81	29.78	0.03
Levenberg	27.03	27.01	0.02
Inc. Levenberg	199.12	199.09	0.03
LCG	63.45	63.33	0.12
SCG	79.22	79.02	0.20

Table 4.5. Best case results from the lymphography data set.

Batch SDBP	L-M	Levenberg	Inc. Levenberg	LCG	SCG
1.2	27.9	28.6	2.6	12.1	10.7

Table 4.6. Comparison of the training speed of each method relative to the incremental SDBP

Table 4.6 shows the training speed of each method relative to the incremental SDBP. The L-M and Levenberg methods performed many times faster than the rest of the methods.

The second test data set is the waveform data set. In this experiment, the network is trained until it classified all the training set correctly. To do this, the network is trained

without employing the network validation. The methods involved in the experiments are the L-M, Levenberg, incremental Levenberg, LCG and SCG methods. Both batch and incremental SDBP are not included in these experiments, that is, because of the slow training process of these methods. The results of each method are shown in Table 4.7.

	Time		
Method	Total	User	System
L-M	767.63	766.42	1.21
Levenberg	772.26	771.23	1.03
Inc. Levenberg	39123.27	39121.22	2.05
LCG	5318.88	5312.33	6.55
SCG	5585.94	5578.29	7.65

Table 4.7. Results from the waveform data set.

In this experiment, the L-M and the Levenberg methods are still favorable in terms of their convergence rates. The incremental Levenberg, on the contrary, ran more than fifty times slower than the L-M and the Levenberg methods, and roughly seven times slower than the LCG and SCG methods. Table 4.8 contains the comparison of the training speed of the L-M, Levenberg, LCG, and SCG methods relative to the incremental Levenberg method.

L-M	Levenberg	LCG	SCG
51.0	50.7	7.4	7.0

Table 4.8. Comparison of the training speed of each method relative to the incremental Levenberg

The last test data set is the financial data set. Like the second experiment, this experiment does not include both the incremental and batch SDBP methods in the comparisons. Here, the network is trained until the mean squared error (MSE) reached $4x10^{-6}$. There are two stopping citeria assigned to the training of the network; these criteria are maximum iterations and error goal, that is, the network is trained until the MSE has reached the desired error goal, or the number of training iterations has reached the maximum allowable iterations. Table 4.9 shows the training time of different methods.

	TIME		
Method	Total	User	System
L-M	1318.06	1315.90	2.17
Levenberg	1334.15	1331.75	2.40
Inc. Levenberg	106862.78	106861.61	1.17
LCG	44781.97	44781.82	0.15
SCG	41725.25	41725.07	0.17

Table 4.9. Result from financial data set.

Once again the L-M and the Levenberg methods have shown their superiority in training time. Likewise, both the LCG and SCG appeared to have the same training speed, and the incremental Levenberg is still showing its its slow convergence. Figure 4.3 shows these results graphically.

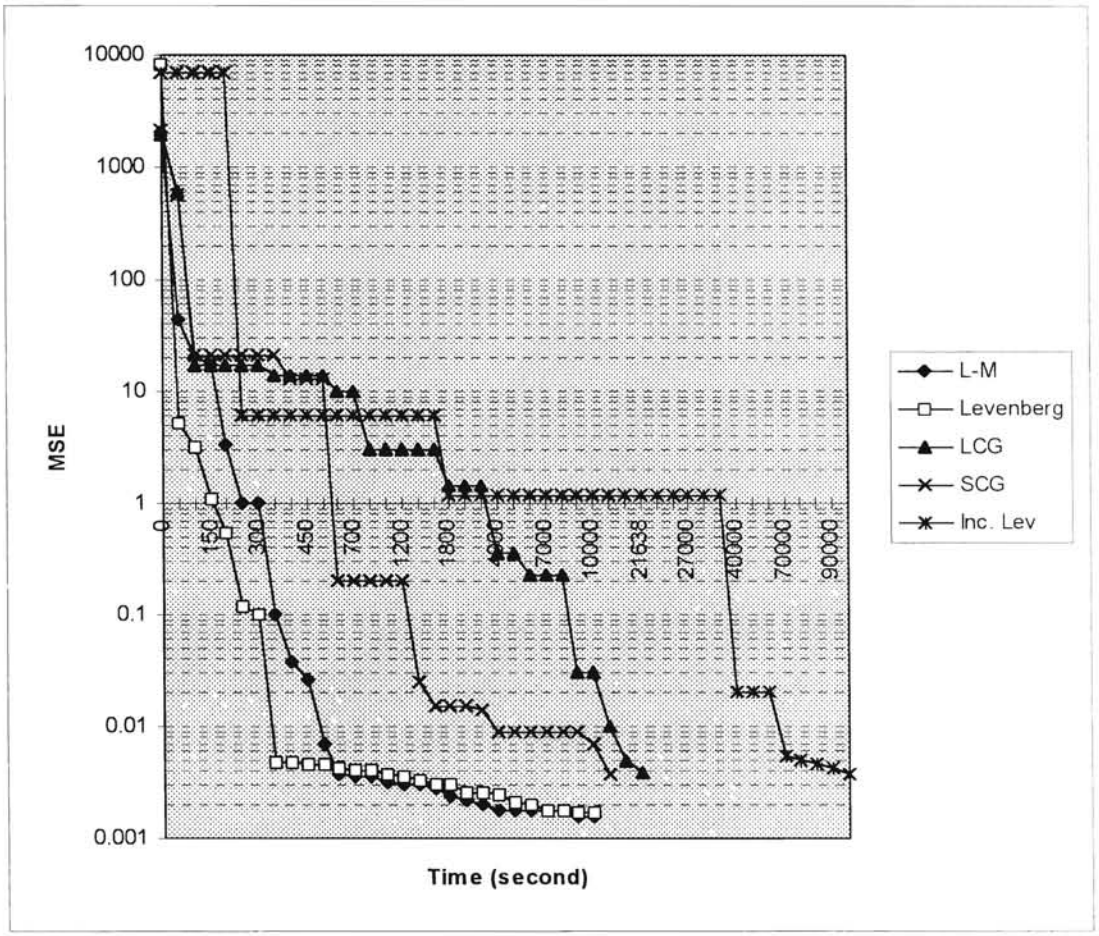

Figure 4.3. MSE versus training time in natural logarithmic scale.

As higher accuracy is needed, the performances of the L-M and Levenberg methods compared to the rest of the methods becomes more notable. This is illustrated in Figure 4.4, where the mean squared error goal is set between $4x10^{-4}$ to $4x10^{-6}$. Figure 4.4 exhibits the time required for training, as a function of the error goal. The curve shows that with an error goal of 0.0004 the L-M and Levenberg methods are just about twice as fast as the other three methods. The rates increase as the accuracy increases (error goal decreases)

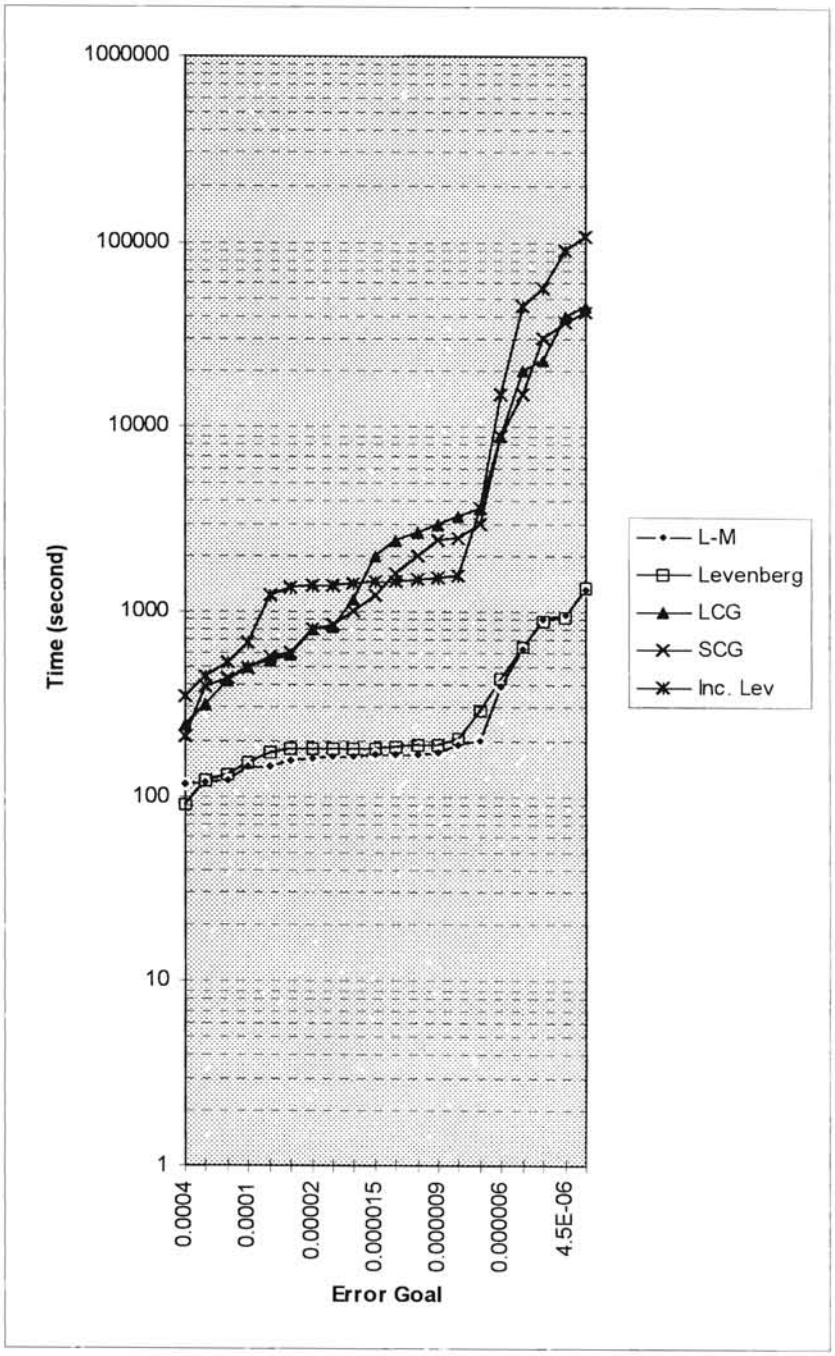

Figure 4.4. Training time versus error goal.

It has been shown that the L-M and batch Levenberg methods are the fastest methods in term of the convergence rates. However, L-M, batch and incremental Levenberg methods can require a large amount of storage to hold the inverse Hessian

matrix. The amount of storage required to hold this matrix is $O(n^2)$ floating point numbers (n is the number of weights). As the result, these methods are only limited to solve problems with several hundreds of weights.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

This paper compares several optimization methods to train feedforward neural networks. In this paper, the SDBP and the Levenberg methods were constructed to handle both batch and incremental approaches; the L-M, SCG, and LCG methods were limited to the batch approach. Three different data sets were used to train these methods. The study has shown that the batch L-M and batch Levenberg methods were the best methods in term of their training speed. The SCG and LCG methods were relatively similar in their convergence rates during training. There is no evidence to show that the incremental Levenberg method is superior than the batch Levenberg approach; on the contrary, the incremental Levenberg method was slower in convergence rate than the SCG or LCG methods.

In the third training task, the time series data set, the networks were trained to yield from low to high precision. The results showed that the superior convergence rates of the L-M and Levenberg methods with respect to the incremental Levenberg, SCG and LCG became more pronounced as higher accuracy was required.

The batch L-M and the batch Levenberg methods usually exhibit linear convergence. It would seem that something could be gained by applying a quasi-Newton algorithm with a backtracking algorithm [DenSchn83] as the line search. However, the L-M, the Levenberg, and the quasi-Newton methods are classified as *restricted methods* where second-order information must be computed and stored. Hence, these methods can only solve problems with a limited number of weights.

Simplicity and reliability of an algorithm may have a significant role in determining a suitable method for training feedforward neural networks. For that reason, Resilient Backprogation (Rprop) [Reid93] offers a simplicity and reliability. The basic principle of Rprop is the direct adaptation of the weight update, and it modifies the weight step directly by introducing the idea of a resilient update value. Rprop incurs only a slight expense in computation compared to the SDBP, and it exhibits much faster training time than the SDBP, but still slower than the restricted methods. Therefore, Rprop deserves further investigation.

REFERENCES

Barnard, E. (1992). 'Optimization for training neural nets,' *IEEE Transactions on Neural Networks* 3, pp. 232-240.

Beale, E. M. L., (1972). *A Derivation of Conjugate Gradients*, NY, Academic Press.

Brent, R. P., (1973). *Algorithms for Minimization Without Derivatives*, NJ, Prentice-Hall.

Charalambous, C. (1992). 'A conjugate gradient algorithm for efficient training of artificial neural networks,' *IEE Proc. Circ. Dev. Syst.* 139, (3), pp. 301-310.

Cheng, M. (1993). *A survey and comparison of conjugate gradient methods for optimization*. Oklahoma State University, Master of Computer Science thesis.

Dennis, J. E. and Schnabel, R. B. (1983): *Numerical methods for unconstrained optimization and nonlinear equations*, NJ, Prentice-Hall.

Fletcher, R. and Reeves, C. M. (1964): 'Function minimization by conjugate gradients,' *The Computer Journal* 7, pp. 149-154.

Hagan, M. T., Demuth, H. B., & Beale, M. (1995). *Neural Network Design*. MA: PWS Publishing Co.

Hagan, M. T. & Menhaj, M. B. (1994). 'Training Feedforward networks with the Marquardt algorithm,' *IEEE Transactions on Neural Networks* 5, pp. 989-993.

Jacobs, R. A. (1988). 'Increased rates of convergence through learning rate adaptation,' Neural Networks 1, pp. 296-307.

Kowalik, J., & Osborne, M. R. (1968). *Methods for Unconstrained Optimization Problems*. NY: American Elsevier Publishing Company, Inc.

Kung, S. Y. (1993). *Digital Neural Networks*. NJ: Prentice Hall.

Levenberg, K. (1944): 'A method for the solution of certain nonlinear problems in least squares,' *Q. Appl. Math*. 2, pp. 164-168.

Marquardt, D. W. (1959): 'Solution of nonlinear chemical engineering models,' *Chem. Eng. Progr*. 6, pp. 65-70.

Marquardt, D. W. (1963): 'An algorithm for least squares estimation of nonlinear parameters,' *J. Soc. Ind. Appl. Math*. 11, pp. 431-441.

McCulloch, W. and Pitts, W. (1943): 'A logical calculus of the ideas immanent in nervous activity,' *Bulletin of Mathematical Biophysics*. 5, pp. 115-133.

Miller, W. and Spooner, D.(1978). *ACM Trans. on Math Software* **4**, pp 369-387.

Minsky, M. and Papert, S., (1969). *Perceptron*, Cambridge, MA, MIT Press.

Moller, M. F. (1993): 'A scaled conjugate gradient algorithm for fast supervised learning,' *Neural Networks* **6**, pp. 525-533.

Nash, J. C. (1979). *Compact Numerical Methods for Computers: Linear Algebra and Function Minimization*. NY: John Wiley & Sons.

Pattichis. C. S., Charalambous, C., & Middleton, L. T. (1995). 'Efficient training of neural network models in classification of electromyographic data,' *Medical & Biological Engineering & Computing* 33, pp. 499-503.

Powell, M. J. D., (1977): 'Restart procedures for the conjugate gradient method,' *Mathematical Programming* 12, pp. 241-254.

Rosenblatt, F. (1958): 'The perceptron: A probabilistic model for information storage and organization in the brain,' *Psychological Review* 65, pp. 386-408.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986): 'Learning internal representation by error propagation,' *Nature* 323, pp. 533-536.

Shanno, D. F. and Phua, K. H. (1978): 'Matrix conditioning and nonlinear optimization,' *Mathematical Programming* 14, pp. 149-160.

Smagt, P. V. D. (1994). 'Minimization methods for training feedforward neural networks,' *Neural Networks* 7, pp.1-11.

Tollenaere, T. (1990). 'SuperSAB: Fast adaptive back propagation with good scaling properties,' *Neural Networks* 5, pp. 561-573.

Vemuri, V. R. (1992). *Artificial Neural Networks: Concepts and Control Application*. CA: IEEE Computer Society Press.

Villiers, J. D., & Barnard, E. (1992): 'Back propagation neural networks with one and two hidden layers,' *IEEE Transaction on Neural Networks* 4, No.1, pp.136-141.

Zhimin. L. (1993). *Recursive Algorithm for Feedforward Networks*. Oklahoma State University Electrical Engineering M.S. Thesis.

Zupan. J. & Gasteiger. J. (1993). Neural Network for Chemist. Weinheim: VCH publishers.

APPENDIX A

TABLE A-1.

INCREMENTAL SDBP WITH DIFFERENT
COMBINATIONS OF STRUCTURES
IN SOLVING LYMPHOGRAPHY CLASSIFICATION PROBLEM

			Time			% Correct	% Correct
Architecture	Eta	Mom.	Total	User	System	in Training	inTesting
18-2-1	0.1	0.1	606.80	606.48	0.32	82.41	64.20
		0.5	607.69	607.57	0.12	82.41	64.20
		0.9	913.20	912.43	0.77	82.41	57.14
	0.2	0.1	735.11	734.88	0.23	80.56	64.20
		0.5	670.38	670.21	0.17	80.56	64.20
		0.9	892.08	891.30	0.78	82.41	64.20
	0.5	0.1	592.56	592.33	0.23	82.41	64.20
		0.5	757.00	756.66	0.34	74.41	57.14
		0.9	1212.55	1211.23	1.32	80.56	64.20
18-3-1	0.1	0.1	1233.77	1233.23	0.54	82.41	64.20
		0.5	1347.79	1347.12	0.67	82.41	64.20
		0.9	1256.96	1256.34	0.62	82.41	64.20
	0.2	0.1	1288.90	1288.23	0.67	82.41	64.20
		0.5	1741.12	1740.10	1.02	82.41	64.20
		0.9	1905.97	1904.50	1.47	82.41	64.20
	0.5	0.1	2131.51	2130.30	1.21	80.56	64.20
		0.5	1944.67	1943.78	0.89	80.56	64.20
		0.9	2678.23	2677.19	1.04	82.41	64.20

TABLE A-2.

BATCH SDBP WITH COMBINATIONS OF STRUCTURES
IN SOLVING LYMPHOGRAPHY CLASSIFICATION PROBLEM

		Time			% Correct	% Correct
Architecture	Mom.	Total	User	System	in Training	inTesting
18-2-1	0.1	1270.477	1270.417	0.06	82.41	64.20
	0.5	843.666	843.65	0.02	82.41	64.20
	0.9	612.47	612.34	0.13	82.41	64.20
18-3-1	0.1	1288.50	1288.30	0.20	82.41	64.20
	0.5	1413.2	1412.79	0.41	80.56	64.20
	0.9	1103.63	1103.51	0.12	82.41	57.14

TABLE A-3.

INCREMENTAL LEVENBERG WITH COMBINATIONS
OF STRUCTURES IN SOLVING LYMPHOGRAPHY CLASSIFICATION PROBLEM

			Time			% Correct	% Correct
Architecture	po	ff	Total	User	System	in Training	inTesting
18-2-1	0.88	1	690.3	690.05	0.25	84.26	64.28
		1000	228.93	228.90	0.03	84.26	64.28
		100000	462.27	462.22	0.05	81.48	71.43
	0.92	1	513.16	512.49	0.67	84.26	64.28
		1000	229.38	229.35	0.03	84.26	64.28
		100000	617.16	503.91	0.73	84.26	64.28
	0.95	1	617.16	617.12	0.04	81.48	64.28
		1000	462.98	462.95	0.03	85.19	64.28
		100000	512.96	512.90	0.06	81.48	71.43
18-3-1	0.88	1000	411.71	411.14	0.57	84.26	64.28

TABLE A-4

LEVENBERG MARQUARDT WITH TWO DIFFERENT NETWORK ARCHITECTURES IN SOLVING LYMPHOGRAPHY

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
18-2-1	38.72	38.65	0.07	82.41	64.28
18-3-1	50.86	50.83	0.03	84.26	64.28

TABLE A-5.

LEVENBERG METHOD WITH DIFFERENT NETWORK ARCHITECTURES IN SOLVING LYMPHOGRAPHY

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
18-2-1	37.75	37.73	0.02	82.41	64.28
18-3-1	49.74	49.71	0.03	81.48	64.28

TABLE A-6

LCG METHOD WITH TWO DIFFERENT NETWORK ARCHITECTURES IN SOLVING LYMPHOGRAPHY

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
18-2-1	81.68	81.6	0.08	83.33	71.43
18-3-1	125.45	125.38	0.07	79.63	64.28

TABLE A-7.

SCG METHOD WITH TWO DIFFERENT NETWORK ARCHITECTURE IN SOLVING LYMPHOGPAHY

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
18-2-1	92.85	92.60	0.25	81.48	64.28
18-3-1	141.48	141.20	0.28	79.63	64.28

TABLE A-8.

INCREMENTAL LEVENBERG METHOD WITH DIFFERENT COMBINATIONS OF STRUCTURES IN SOLVING WAVEFORM

			Time			% Correct	% Correct
Architecture	po	ff	Total	User	System	in Training	inTesting
21-2-1	1	0.88	10832.79	10831.67	1.12	99.47	98.40
	1000	0.88	9121.71	9120.48	1.23	98.94	96.80
	100000	0.88	9305.29	9304.31	0.98	91.49	96.80
	1000	0.92	8014.15	8013.13	1.02	98.94	96.80
	1000	0.95	11529.07	11527.40	1.67	91 49	87.20
21-3-1	1000	0.92	23507.06	23505.12	1.94	98.94	96.80
21-5-1	1000	0.92	76345.98	76344.20	1.78	99.47	96.80

TABLE A-9.

L-M METHOD WITH COMBINATIONS OF NETWORK ARCHITECTURES IN SOLVING WAVEFORM

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
21-2-1	336.34	335.67	0.67	99.47	99.20
21-3-1	690.33	689.15	1.18	99.47	99.20
21-5-1	1426.29	1424.83	1.46	98.94	98.40

TABLE A-10.

LEVENBERG METHOD WITH COMBINATIONS OF NETWORK ARCHITECTURES IN SOLVING WAVEFORM

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
21-2-1	381.12	380.20	0.92	99.47	96.80
21-3-1	648.45	646.93	1.52	99.47	96.80
21-5-1	1441.67	1440.10	1.57	91.49	92.00

TABLE A-11.

LCG METHOD WITH COMBINATIONS OF NETWORK ARCHITECTURES IN SOLVING WAVEFORM

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
21-2-1	1128.47	1124.40	4.07	99.47	96.80
21-3-1	1214.56	1210.23	4.33	99.47	97.60
21-5-1	12177.62	12167.22	10.40	99.47	97.60

TABLE A-12.

SCG METHOD WITH COMBINATIONS OF NETWORK ARCHITECTURES IN SOLVING WAVEFORM

	Time			% Correct	% Correct
Architecture	Total	User	System	in Training	inTesting
21-2-1	1183.97	1178.25	5.72	97.60	97.60
21-3-1	1947.36	1937.31	10.05	99.47	97.60
21-5-1	11951.13	11941.63	9.50	99.47	97.60

TABLE A-13.

VARIOUS METHODS WITH COMBINATIONS OF NETWORK ARCHITECTURE IN SOLVING TIMESERIES (FINANCIAL DATA) PROBLEM

		TIME		
Method	Architecture	Total	User	System
L-M	2-6-1.	890.45	888.97	1.48
	2-9-1.	597.95	596.28	1.67
	2-12-1.	1053.26	1051.33	1.93
	2-24-1.	2730.58	2727.00	3.58
Levenberg	2-6-1.	1044.73	1043.67	1.07
	2-9-1.	755.00	753.08	1.92
	2-12-1.	1100.99	1099.10	1.89
	2-24-1.	2435.87	2431.15	4.72
Inc. Levenberg	2-6-1.	91053.32	91053.10	0.22
po=10^3, ff=0.88	2-9-1.	122672.23	122670.12	2.11
	*2-12-1.	-	-	-
	*2-24-1.	-	-	-
LCG	2-6-1.	21638.40	21638.27	0.13
	2-9-1.	20223.25	20223.10	0.15
	2-12-1.	36277.32	36277.12	0.20
	2-24-1.	100988.90	100988.80	0.12
SCG	2-6-1.	16889.95	16889.80	0.15
	2-9-1.	17820.27	17820.11	0.16
	2-12-1.	34780.46	34780.20	0.26
	2-24-1.	97410.32	97410.15	0.12

APPENDIX B

APPENDIX B

PROGRAM LISTING

```
PROGRAM DRIVER
C  THE DRIVER IS THE MAIN PROGRAM, THAT INVOKES ALL THE OPTIMIZATION
C  METHOD THAT INCORPORATED WITH BACKPROPAGATION OF ERRORS. THOSE
C  OPTIMIZATION METHODS ARE :
C     1. SDBP WITH GOLDEN SECTION SEARCH AND MOMENTUM.
C        IT INCLUDES BOTH INCREMENTAL AND BATCH VERSION.
C     2. SCALED CONJUGATE GRADIENT, SCG (MOLLER'S SCG).
C     3. LE'S CONJUGATE GRADIENT (LCG).
C     4. LEVENBERG I.
C        (BOTH BATCH AND INCREMENTAL VERSION)
C     5. LEVENBERG-MARQUARDT.
C-----------------------------------------------------------------
C THIS PROGRAM REQUIRES THE USER TO KEY-IN THE NUMBER OF HIDDEN
C NODE OF THE NETWORK, A FILE CONTAINS THE NETWORK I/O, SEED FOR
C RND, OPTIMIZATION OPTIONS.
C-----------------------------------------------------------------
C
C THE FOLLOWING ARE THE CONSTANT VALUES TO BE ASSIGNED FOR
C DIMENSIONING THE VARIABLES:
C MAXWSIZE INCLUDES BIASES
C (MAXWSIZE = (MAXHID*(MAXINP+1))
C   +(MAXOUT*(MAXHID+1)))
C (MAXA=MAXHID+MAXOUT+MAXINP)
C
C OTHER CONSTANT APPLIED :
C MAXITER=MAXIMUM ITERATIONS
C (SMIN=-40) OVERFLOW CONTROL
C  A(,) CONTAINS INPUT+OUTPUT OF HIDDEN LAYER+ACTUAL OUTPUT
C
C VARIABLES DECLARATION:
C W    : WEIGHTS + BIASES VECTOR
C G    : GRADIENT VECTOR
C ERR  : ERROR VECTOR
C A    : INPUT+HIDDEN_OUTPUT+COMPUTED OUTPUT+ DESIRED OUTPUT
C EGOAL : ERROR GOAL FOR TERMINATION CRITERION
C ETA  : STEPSIZE
C GAMMA : MOMENTUM CONSTANT
C LSEED : INTEGER FOR SEED RND
C MODE  : INTEGER FOR BATCH OR ONLINE MODE
C LP   : LOGICAL NUMBER FOR PRINTING PURPOSE
```

```
C NW    : NUMBER OF RESIDUALS
C OPT   : LOGICAL NUMBER FOR OPTIMIZATION OPTION
C-----------------------------------------------------------------
      DOUBLE PRECISION W(280),G(280),P(32),ERR(26),A(228.132),
     * EGOAL,ETA,GAMMA,AA(132),TOTERR,DIST,B(228,132),PO,RLAMDA
C
      INTEGER NPAT,NHID,NOUT,NINP,NW,SMIN,EPOCH,MODE,OPT,LP,
     * MAXITR,LSEED,LDIMA,MAXPAT,MAXJAC,MAXW,MAXOUT,LP2,CORR,
     * MODE2,OPTION,IOPTST,NPATTS,CLASS,LPX,LPY,LPZ
C CLOCK SET
      REAL DTIME,ETIME
      REAL TOTAL,TIMES(2),TMP(2)
C
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
      COMMON /FUNCT/ OPTION
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
      LPX=11
      LPY=21
      LPZ=31
      OPEN(UNIT=LPX,FILE='TRAIN.IN',STATUS='OLD')
      OPEN(UNIT=LPY,FILE='TEST.IN',STATUS='OLD')
      OPEN(UNIT=LPZ,FILE='VALID.IN',STATUS='OLD')
      OPEN(UNIT=41,FILE='T2',STATUS='NEW')
C
C***CONSTANT***
C
C (SMIN=-40) OVERFLOW CONTROL
C MAXPAT = MAXIMUM ALLOWABLE DATA SETS
C
      SMIN=-40
C LP=6 STDOUT
      LP=6
      LP2=41
      CORR=0
      MAXPAT=228
      MAXW=280
      MAXOUT=26
      MAXJAC=MAXPAT*MAXOUT
      EGOAL=.004D0
C SIGMOID CHOSEN
      OPTION=0
C
C  PROMPT USER FOR NUMBER OF NODES IN HIDDEN LAYER
C  AND ETA LEARNING CONSTANT
C
C
C  TOP OF THE LOOP FOR CONTINUATION OF DIFFERENT NET STRUC
C
22    CONTINUE
C
      WRITE(LP,1)
      READ*, OPT
      WRITE(LP,4)
```

```
      READ*. NHID
      WRITE(LP,5)
      READ *,I
      LSEED=I
      WRITE(LP,6)
      READ*,MODE2
      WRITE(LP,7)
      READ*,DIST
      WRITE(LP,8)
      READ*,IOPTST
      IF (OPT.EQ.1) THEN
       WRITE(LP,11)
       READ *, MODE
       WRITE(LP,12)
       READ *, GAMMA
       IF(MODE.EQ.0) THEN
        MAXITR=8000
        WRITE(LP,13)
        READ *,ETA
        WRITE(LP,14) ETA
       ELSE
        MAXITR=2000
       ENDIF
      ELSE IF (OPT.EQ.6) THEN
       WRITE(LP,23)
       READ*, PO
       WRITE(LP,24)
       READ*,RLAMDA
       WRITE(LP,25)PO,RLAMDA
      ENDIF
C
C
C  READ NUMBER OF PATTERN, INPUTS, OUTPUTS
C
      IF (MODE2.EQ.1) READ(LPX,*) NPAT,NINP,NOUT,CLASS
      IF (MODE2.NE.1) READ(LPX,*) NPAT,NINP,NOUT
      LDIMA=NINP+NHID+(2*NOUT)
      NW=(NINP+1)*NHID+(NHID+1)*NOUT
      CALL GETTD(A,MAXPAT,LDIMA,MODE2)
C
      IF (NW.GT.280) THEN
       WRITE(LP,15) NW
       GOTO 34
      ENDIF
C
C  INIT CLOCK
C
      TOTAL=ETIME(TIMES)
      TOTAL=DTIME(TIMES)
C
C***CALL FOR RANDOMIZE WEIGHTS AND BIASES
C  THE RANDOMIZE WEIGHTS AND BIASES CONTAINS VALUES BETWEEN -0.5...0.5
C
C  W(1)=UNI(LSEED)-5.0D-1
```

```
      W(1)=UNI(LSEED)
      DO 10 I=1,NW
       W(I)=(2.D0*UNI(0)-1.D0)*DIST
10    CONTINUE
C
C STEEPEST DESCENT
C
      IF (OPT.EQ.1) THEN
       CALL SD(EGOAL,ETA,MODE,A,W,ERR,MAXITR,GAMMA,
     *  NOUT,NW,LDIMA,MAXPAT,MODE2)
C
C SCALED CG
C
      ELSE IF (OPT.EQ.2) THEN
       CALL SCG(EGOAL,W,A,NOUT,NW,LDIMA,MAXPAT,MODE2)
C
C LEVENBERG-MARQUARDT
C
      ELSE IF (OPT.EQ.3) THEN
       CALL LM(EGOAL,W,A,NOUT,NW,LDIMA,MAXPAT,MAXJAC,MAXW,OPT,MODE2)
C
C LEVENBERG
C
      ELSE IF (OPT.EQ.4) THEN
       CALL LM(EGOAL,W,A,NOUT,NW,LDIMA,MAXPAT,MAXJAC,MAXW,OPT,MODE2)
C
C LE'S CG
C
      ELSE IF (OPT.EQ.5) THEN
       CALL LCG(EGOAL,W,A,NHID,NOUT,NW,LDIMA,MAXPAT,MODE2)
C
C INCREMENTAL LEVENBERG
C
      ELSE IF (OPT.EQ.6) THEN
       CALL ILEV(EGOAL,W,A,NW,NOUT,MAXPAT,LDIMA,PO,RLAMDA,MODE2)
      ENDIF
C
      WRITE(LP,26)
C
      WRITE(LP,27) (W(KK),KK=1,NW)
C
      TOTERR=0.D0
      CALL FCN(W,A,TOTERR,NOUT,NW,LDIMA,MAXPAT)
      WRITE(LP,28) TOTERR
C
      TMP(1)=TIMES(1)
      TMP(2)=TIMES(2)
      TOTAL=TOTAL+DTIME(TIMES)
      TIMES(1)=TMP(1)+TIMES(1)
      TIMES(2)=TMP(2)+TIMES(2)
      WRITE(LP,29) TIMES(1),TIMES(2),TOTAL
C
      WRITE(LP,30) NINP,NHID,NOUT,LSEED,-DIST,DIST
C
```

```
      MM=LCLSF(W,A,NOUT,NW,LDIMA,MAXPAT)
      IF ((IOPTST.EQ.1).OR.(IOPTST.EQ.3)) THEN
       NPATTS=NPAT
       READ(21,*) NPAT
       CALL GETTS(B,NINP,NHID,NOUT,NPAT,NW,CLASS,MODE2,LPY)
       CALL TEST(W,B,NINP,NHID,NOUT,NW,LDIMA,MAXPAT,NPAT,LP)
       NPAT=NPATTS
      ENDIF
      REWIND(UNIT=LPY)
      REWIND(UNIT=LPX)
      GOTO 22
C
C FORMAT
C
1     FORMAT(///'ENTER THE SELECTION FOR EACH OPTIMIZATION METHOD:'/,
     *  'BP 1, SCG 2, LEV 3, LM 4, LCG 5, INCLEV 6')
4     FORMAT ('ENTER #NODES IN HIDDEN LAYER')
5     FORMAT ('ENTER SEED')
6     FORMAT ('CLASSIFICATION PROB=1,OTHERS=0')
7     FORMAT ('WEIGHTS DISTRIBUTION, KEY IN 0.5 FOR -.5<W<.5')
8     FORMAT ('ENTER 0 FOR TRAINING ONLY'/,
     *       'ENTER 1 FOR TRAINING AND TESTING'/,
     *       'ENTER 2 FOR TRAINING AND VALIDATING'/,
     *       'ENTER 3 FOR TRAINING, VALIDATING AND TESTING')
11    FORMAT ('ENTER MODE (BATCH=1, SQUENTIAL=0)')
12    FORMAT ('ENTER MOMENTUM CONSTANT')
13    FORMAT ('ENTER ETA')
14    FORMAT ('ETA=',F6.4)
15    FORMAT(/'ALLOWABLE NUMBER OF WEIGHTS IS:280',/'CURRENTLY',
     *  'YOU ARE USING:',I3)
23    FORMAT ('PO=')
24    FORMAT ('LAMBDA=')
25    FORMAT(/'PO AND RLAMBDA IS',F9.2,F9.2/)
26    FORMAT (///'THE PARAMETERS AFTER TRAINING ARE:')
27    FORMAT(G15.7,G15.7,G15.7)
28    FORMAT(/'FINAL SSE BASED ON THE ABOVE WEIGHTS ARE: ',G15.7//)
29    FORMAT('USER_TIME  :',G15.7,/'SYSTEM_TIME:',G15.7,/'TOTAL_TIME :',
     *  G15.7//)
30    FORMAT(/'NETWORK          : ',I2,'-',I2,'-',I2,
     *       /'SEED             : ',I2,
     *       /'WEIGHT DISTRIBUTION: ', F5.2,'..',F4.2/)
C***END OF DRIVER****
34     CONTINUE
      CLOSE(UNIT=LPX,STATUS='KEEP')
      CLOSE(UNIT=LPY,STATUS='KEEP')
      CLOSE(UNIT=LPZ,STATUS='KEEP')
C
C END OF DRIVER
C
      END
      SUBROUTINE TEST(W,A,NINP,NHID,NOUT,NW,LDIMA,MAX,NPAT,LP)
C
C SUBROUTINE TEST
C THIS SUBROUTINE RUN A TEST DATA SET. IT RUN A 40-20-40 CORRECTNESS
```

```
C CRITERION. THIS ROUTINE CALLS A FPASS TO COMPUTE THE ERROR GENERATED
C BY THE NETWORK. FOR A PATTERN CLASSIFICATION MODE THE ROUTINE
C GENERATE AN OUTPUT, SUCH AS, THE NUMBER CORRECT CLASSIFICATION.
C
      INTEGER NINP,NHID,NOUT,NPAT,NW,LP,LP2,LDIMA,
     *  MAX,CORR,CNT,CLASS,IOPTST,LPX,LPY.LPZ
      DOUBLE PRECISION W,A,AA,ERR,TOTERR
      DIMENSION W(NW),A(MAX.LDIMA),AA(LDIMA),ERR(NOUT)
      REAL DTIME,ETIME
      REAL TOTAL,TIMES(2)
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
      WRITE(LP,1)
1     FORMAT(///'TESTING THE NETWORK'/)
      TOTERR=0.D0
      CORR=0
      DO 198 I=1,NPAT
       DO 201 J=1,NINP+NHID+(2*NOUT)
        AA(J)=A(I,J)
201     CONTINUE
       CALL FPASS(W,AA,ERR,TOTERR,NOUT,NW,LDIMA)
C
C CLASSIFICATION UNTIL EACH SQUARE ERR IS LESS THAN 20%
C
       CNT=0
       DO 299 J=1,NOUT
        IF (DABS(ERR(J)).LT.(.4D0*.6D0/(REAL(CLASS)-1.D0))) THEN
         CNT=CNT+1
        ENDIF
299     CONTINUE
       IF (CNT.EQ.NOUT) CORR=CORR+1
198    CONTINUE
      WRITE(LP,70) TOTERR,CORR,NPAT
70     FORMAT('TOTAL ERROR=',G15.7,/'NUMBER OF CORRECT CLASSIFICATION='.
     *     I4,/'NUMBER OF DATA SET TESTED=',I4)
      RETURN
C
C  END OF TEST
C
      END
      DOUBLE PRECISION FUNCTION VALID(W,MNOUT,MNW,
C
C SUBROUTINE VALID()
C
C THIS ROUTINE RUN A VALIDATION SET IN ORDER TO TOGGLE ON THE
C TOP TRAINING PROCESS. THIS ROUTINE CALL FPASS() WITH THE GIVEN
C VALIDATION SET AND THE COMPUTED WEIGHTS BY THE OPT().
C THIS ROUTINE RETURN A TOTERR, WHICH IS THE SUM OF SQUARE ERROR
C GENERATED BY THE NETWORK. THIS ROUTINE IS CALLED BY EACH OPT()
C AND THE RETURN VALUE OF THIS ROUTINE (TOTERR) IS OBSERVED, TO
C SEE IF IT IS LARGER THAN THE PREVIOUS PASS. THIS IS MAINLY TO
C CONTROL OVERTRAINING.
C
```

```
      INTEGER FUNCTION LCLSF(W,A,MNOUT,MNW,LDIMA,MAX)
C
C FUNCTION LCLSF()
C
C THIS ROUTINE RETURN THE NUMBER OF CORRECT CLASSIFICATION
C THAT THE NETWORK OBTAIN. THIS FUNCTION IS CALLED BY OPT()
C FOR CLASSIFICATION PROBLEM ONLY.
C
      INTEGER NINP,NHID,NOUT,NPAT,NW,LP,LP2,MNOUT.MNW,LDIMA,
     *  MAX,CORR,CNT,CLASS,IOPTST,LPX,LPY,LPZ
      DOUBLE PRECISION W,A,AA,ERR,TOTERR
      DIMENSION W(MNW),A(MAX,LDIMA).AA(LDIMA).ERR(MNOUT)
      REAL DTIME,ETIME
      REAL TOTAL,TIMES(2),TMP(2)
      COMMON /DRIVE/ NINP,NHID,NOUT.NPAT,NW,LP,LP2
      COMMON /DCLASS/ CLASS,TIMES,TOTAL.IOPTST,LPX,LPY.LPZ
C
      TOTERR=0.D0
      CORR=0
      DO 198 I=1,NPAT
       DO 201 J=1,NINP+NHID+(2*NOUT)
        AA(J)=A(I,J)
201     CONTINUE
       CALL FPASS(W,AA,ERR,TOTERR,NOUT,NW,LDIMA)
C
       CNT=0
       DO 299 J=1,NOUT
C ERROR TEST TO BE LESS THAN 40% OF CLASSIFICATION
        IF (DABS(ERR(J)).LT.(.4D0*.6D0/(REAL(CLASS)-1.D0))) THEN
C        WRITE*,(AA(K),K=1,NINP),(AA(K+NINP+NHID+NOUT),K=1,NOUT)
         CNT=CNT+1
        ENDIF
299     CONTINUE
       IF (CNT.EQ.NOUT) CORR=CORR+1
198    CONTINUE
      WRITE(LP,80) TOTERR,CORR,NPAT
80     FORMAT(//'TOTAL ERROR=',G15.7,/'NUMBER OF CORRECT CLASSI'
     *     ,'FICATION=',I4,/'NUMBER OF DATA SET TRAINED=',I4)
      TMP(1)=TIMES(1)
      TMP(2)=TIMES(2)
      TOTAL=TOTAL+DTIME(TIMES)
      TIMES(1)=TMP(1)+TIMES(1)
      TIMES(2)=TMP(2)+TIMES(2)
      WRITE(LP,29) TIMES(1),TIMES(2),TOTAL
29     FORMAT('USER_TIME  :',G15.7,/'SYSTEM_TIME:',G15.7,/'TOTAL_TIME :',
     *  G15.7//)
      LCLSF=CORR
      RETURN
C  END OF LCLSF
      END
C
C  DERIVATIVE OF TRANSFER FUNCTION FUNCTION 1
C  D(LOGSIG(X))/DX
      DOUBLE PRECISION FUNCTION DF1(IN)
```

```
     * LDIMA,MAX,MODE2)
      INTEGER NINP,NHID,NOUT,NPAT,NW,LP,LP2,LDIMA,NPATV,
     * MAX,CORR,CNT,CLASS,IOPTST,LPX,LPY,LPZ,MNOUT,MNW,MODE2
      DOUBLE PRECISION W,A,AA,ERR,TOTERR
      DIMENSION W(MNW),A(MAX,LDIMA),AA(LDIMA),ERR(MNOUT)
      REAL DTIME,ETIME
      REAL TOTAL,TIMES(2)
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
C     WRITE(LP,1)
C1    FORMAT(//'VALIDATING THE NETWORK'/)
      TOTERR=0.D0
      CORR=0
C
      NPATV=NPAT
      READ(LPZ,*) NPAT
      DO 10 I=1,NPAT
       READ(LPZ,*) (A(I,K),K=1,NINP),(A(I,J+NINP+NHID+NOUT),J=1,NOUT)
       IF(MODE2.EQ.1) THEN
        DO 20 J=1,NOUT
         A(I,J+NINP+NHID+NOUT)=.2D0+(A(I,J+NINP+NHID+NOUT)-1.D0)*
     *      (.6D0/(REAL(CLASS)-1.D0))
20       CONTINUE
       ENDIF
10    CONTINUE
C
      DO 198 I=1,NPAT
       DO 201 J=1,NINP+NHID+(2*NOUT)
        AA(J)=A(I,J)
201     CONTINUE
       CALL FPASS(W,AA,ERR,TOTERR,NOUT,NW,LDIMA)
C
C CLASSIFICATION UNTIL EACH SQUARE ERR IS LESS THAN 20%
C
       CNT=0
       DO 299 J=1,NOUT
        IF (DABS(ERR(J)).LT.(.4D0*.6D0/(REAL(CLASS)-1.D0))) THEN
         CNT=CNT+1
        ENDIF
299     CONTINUE
       IF (CNT.EQ.NOUT) CORR=CORR+1
198    CONTINUE
C      WRITE(LP,70) TOTERR,CORR,NPAT
C70    FORMAT('TOTAL ERROR=',G15.7,/'NUMBER OF CORRECT CLASSIFICATION=',
C     *     I4,/'NUMBER OF DATA SET VALIDATED=',I4)
       VALID=TOTERR
       NPAT=NPATV
       REWIND(LPZ)
       RETURN
C
C  END OF VALIDATION
C
       END
```

```
      DOUBLE PRECISION IN
C       DF1=(1.0D0 - F1(IN)) * F1(IN)
      DF1=(1.0D0 - F1(IN)) * .5D0*F1(IN)
      RETURN
      END
C
C  DERIVATIVE OF TRANSFER FUNCTION FUNCTION 2
C  D(LOGSIG(X))/DX
      DOUBLE PRECISION FUNCTION DF2(IN)
      DOUBLE PRECISION IN
      INTEGER OPTION
      COMMON /FUNCT/ OPTION
      IF (OPTION.EQ.0) THEN
C       DF2=(1.0D0 - F2(IN)) * F2(IN)
       DF2=(1.0D0 - F2(IN)) * .5D0*F2(IN)
      ELSE
       DF2=1.D0
      ENDIF
      RETURN
      END

C LOGSIG TRANSFER FUNCTION 1
      DOUBLE PRECISION FUNCTION F1(IN)
      DOUBLE PRECISION IN
C  SMIN TO CONTROL OVERFLOW
      PARAMETER(SMIN=-40)
      IF(IN.LT.SMIN) THEN
       F1=0.0D0
      ELSEIF (IN.GT.(-SMIN)) THEN
       F1=1.0D0
      ELSE
C       F1=1.0D+00/(1.0D+00 + DEXP(-IN))
       F1=1.0D+00/(1.0D+00 + DEXP(-.5D+00 *IN))
      ENDIF
       RETURN
      END

C LOGSIG TRANSFER FUNCTION 2
      DOUBLE PRECISION FUNCTION F2(IN)
      DOUBLE PRECISION IN
      INTEGER OPTION
      COMMON /FUNCT/ OPTION
C  SMIN TO CONTROL OVERFLOW
      PARAMETER(SMIN=-40)
      IF (OPTION.EQ.0) THEN
       IF(IN.LT.SMIN) THEN
        F2=0.0D0
       ELSEIF (IN.GT.(-SMIN)) THEN
        F2=1.0D0
       ELSE
C        F2=1.0D+00/(1.0D+00 + DEXP(-IN))
       F2=1.0D+00/(1.0D+00 + DEXP(-.50D+00 *IN))
       ENDIF
      ELSE
```

```
C PURELIN
      F2=IN
     ENDIF
      RETURN
     END
C
C ROUTINE THAT READIN THE DATA SETS
C FOR TRAINING
C
     SUBROUTINE GETTD(A,MAX,NDIMA,MODE2)
     DOUBLE PRECISION A(MAX,NDIMA)
     INTEGER NPAT,NINP,NOUT,NHID,NW,LP,LP2,CLASS,MODE2,MAX,NDIMA
     INTEGER IOPTST,LPX,LPY,LPZ
     REAL TOTAL,TIMES(2)
     COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
     COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
     DO 10 I=1,NPAT
      READ(LPX,*) (A(I,K),K=1,NINP),(A(I,J+NINP+NHID+NOUT),J=1,NOUT)
      IF(MODE2.EQ.1) THEN
      DO 20 J=1,NOUT
       A(I,J+NINP+NHID+NOUT)=.2D0+(A(I,J+NINP+NHID+NOUT)-1.D0)*
    *      (.6D0/(REAL(CLASS)-1.D0))
20     CONTINUE
      ENDIF
10    CONTINUE
     RETURN
     END
C
C ROUTINE THAT READIN THE DATA SETS
C FOR TESTING
C
     SUBROUTINE GETTS(A,NINP,NHID,NOUT,NPAT,NW,CLASS,MODE2,LPY)
     DOUBLE PRECISION A(228,132)
     INTEGER NPAT,NINP,NOUT,NHID,NW,LP,LP2,CLASS,MODE2,LPY
     DO 10 I=1,NPAT
      READ(LPY,*) (A(I,K),K=1,NINP),(A(I,J+NINP+NHID+NOUT),J=1,NOUT)
      IF (MODE2.EQ.1) THEN
      DO 20 J=1,NOUT
       A(I,J+NINP+NHID+NOUT)=.2D0+(A(I,J+NINP+NHID+NOUT)-1.D0)*
    *      (.6D0/(REAL(CLASS)-1.D0))
20     CONTINUE
      ENDIF
10    CONTINUE
     RETURN
     END
     SUBROUTINE FCN(W,A,FX,MNOUT,MNW,NDIMA,MNPAT)
C
C  SUBROUTINE THAT RETURN FX: ERROR EVALUATED BY A GIVEN PARAMETERS
C
     DOUBLE PRECISION W,A,ERR,FX,AA
     INTEGER NINP,NHID,NOUT,NPAT,NW,LP,MNOUT,MNW,NDIMA,MNPAT,
    * LP2
     DIMENSION W(MNW),A(MNPAT,NDIMA),ERR(MNOUT),AA(NDIMA)
     COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
```

```
C
      FX=0.D0
      DO 10 I=1,NPAT
       DO 20 J=1,NINP
        AA(J)=A(I,J)
20     CONTINUE
       DO 30 J=1,NOUT
        AA(J+NINP+NHID+NOUT)=A(I,J+NINP+NHID+NOUT)
30     CONTINUE
      CALL FPASS(W,AA,ERR,FX,MNOUT,MNW,NDIMA)
10    CONTINUE
      RETURN
      END
C
C
C  SUBROUTINE THAT EVALUATED THE GRADIENT OF A GIVEN
C  PARAMETERS. G CONTAINS THE GRADIENTS EVALUATED.
C
      SUBROUTINE FGRAD(G,W,A,MNOUT,MNW,NDIMA,MNPAT)
C
      DOUBLE PRECISION W,A,G,ERR,FX,AA,GTOT
      INTEGER NINP,NHID,NOUT,NPAT,NW,LP,MNOUT,MNW,NDIMA,MNPAT,LP2
      DIMENSION W(MNW),A(MNPAT,NDIMA),G(MNW),ERR(MNOUT),AA(NDIMA),
     *  GTOT(MNW)
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
C     NW=(NHID*(NINP+1))+((NHID+1)*NOUT)
C
      FX=0.D0
C
C  ITERATE AND FIND THE RESULTANT OF GRADIENT (BATCHING)
C
      DO 5 J=1,NW
       GTOT(J)=0.D0
5     CONTINUE
      DO 10 I=1,NPAT
       DO 20 J=1,NINP
        AA(J)=A(I,J)
20     CONTINUE
       DO 30 J=1,NOUT
        AA(J+NINP+NHID+NOUT)=A(I,J+NINP+NHID+NOUT)
30     CONTINUE
      CALL GRAD(G,W,AA,ERR,FX,NHID,MNOUT,MNW,NDIMA)
C
C  SUM UP ALL THE GRADIENT OF EACH PATTERN
C
      DO 40 J=1,NW
       GTOT(J)=GTOT(J)+G(J)
40    CONTINUE
10    CONTINUE
C
C  SET THE GRADIENT RESULTANT (BATCHING)
C
      DO 60 J=1,NW
```

```
C      G(J)=GTOT(J)/NPAT
       G(J)=GTOT(J)
60     CONTINUE
C
C  END OF FGRAD()
C
       RETURN
       END

       SUBROUTINE GRCHEK (AA,X,MASK,GRAD,GRADIF,RLDFMX,MNW,NDIMA,MNPAT)
C
C  GRCHEK 1.0        SEPTEMBER 1992
C
C  CHECK THE ANALYTICAL GRADIENT VECTOR USING FINITE
C  DIFFERENCES.
C
C  J. P. CHANDLER, COMPUTER SCIENCE DEPARTMENT,
C    OKLAHOMA STATE UNIVERSITY
C
C
C
       IMPLICIT REAL*8 (A-H,O-Z)
C
       INTEGER MASK,NW,LP,J,IOPT,LP2
       INTEGER NINP,NHID,NOUT,NPAT,NDIMA,MNPAT,MNW
C
       DOUBLE PRECISION AA
       DOUBLE PRECISION X,GRAD,GRADIF,RLDFMX,RELDFK,RZERO,
      *  XSAVE,DX,RLDF,DENOM,DABS,DMAX1,
      *  FXA,FXB
C
       COMMON /NP/NPROB
       COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
       DIMENSION AA(MNPAT,NDIMA)
       DIMENSION X(MNW),MASK(MNW),GRAD(MNW),GRADIF(MNW)
C
       RELDFK=1.0D-5
C
       RZERO=0.0D0
C
       NFV=0
       NGV=0
C
       DO 10 J=1,NW
C
         IF(MASK(J).NE.0) THEN
           GRADIF(J)=RZERO
           GO TO 10
         ENDIF
C
         XSAVE=X(J)
C
C  ADD AN ABSDFK PARAMETER FOR DIFFERENCING IF DESIRED:
```

```
C     DX=RELDFK*DABS(X(J))+ABSDFK
C  (THE VALUE OF ABSDFK WILL DEPEND ON THE SCALING OF X(J).)
C
      DX=RELDFK*DABS(X(J))
      IF(DX.EQ.RZERO) DX=RELDFK
      X(J)=XSAVE+DX
      IOPT=-1
      CALL FCN (X,AA,FXA,NOUT,NW,NDIMA,MNPAT)
      X(J)=XSAVE-DX
      CALL FCN (X,AA,FXB,NOUT,NW,NDIMA,MNPAT)
      X(J)=XSAVE
      GRADIF(J)=(FXA-FXB)/(DX+DX)
   10   CONTINUE
C
     WRITE(LP,20)(GRADIF(J),J=1,NW)
   20 FORMAT(/' NUMERICAL GRADIENT ='/(1X,1PG15.7,4G15.7))
C
     IOPT=0
     CALL FCN (X,AA,FX,NOUT,NW,NDIMA,MNPAT)
     CALL FGRAD(GRAD,X,AA,NOUT,NW,NDIMA,MNPAT)
C
     WRITE(LP,30)(GRAD(J),J=1,NW)
   30 FORMAT(/' ANALYTICAL GRADIENT ='/(1X,1PG15.7,4G15.7))
C
     RLDFMX=RZERO
     DO 40 J=1,NW
       IF(MASK(J).NE.0) GO TO 40
       DENOM=DMAX1(DABS(GRAD(J)),DABS(GRADIF(J)))
       IF(DENOM.EQ.RZERO) DENOM=1.0D0
       RLDF=(GRAD(J)-GRADIF(J))/DENOM
       RLDFMX=DMAX1(RLDFMX,DABS(RLDF))
   40   CONTINUE
C
     WRITE(LP,50)RLDFMX
   50 FORMAT(//' MAXIMUM RELATIVE DIFFERENCE =',1PG15.7)
C
     RETURN
C
C  END GRCHEK
C
     END
     SUBROUTINE GRAD(G,W,A,ERR,TOTERR,MNHID,MNOUT,MNW,NDIMA)
C  SUBROUTINE GRAD()
C  HENDRA TIO @ OKLAHOMA STATE UNIVERSITY, COMPUTER SCIENCE DEPT.
C
C  THIS ROUTINE COMPUTE THE GRADIENT OF ERROR FUNCTION WHICH
C  PUT INTO AN ARRAY G. THE GRADIENT COMPUTED IS ESTABLISHED
C  BY BACKPROPAGATING THE COMPUTED ERROR AT THE FINAL LAYER
C  AND COMPUTE THE SENSITIVITIES BY BACKWARD COMPUTATION (BACK-
C  WARD PASS). IN OTHER HANDS, THE PROCESS IS THE FORWARD PASS
C  WHERE THE ERROR IS COMPUTED AT THE FINAL LAYER, AND THE
C  BACKWARD PASS WHERE THE SENSITIVITIES IS OBTAINED FROM FINAL
C  LAYER BACK TO THE FIRST LAYER.
C
```

```
C  VARIABLES DECLARATION.
C   G[]   GRADIENT ARRAY
C   A[]   INPUT/OUTPUT OF A SINGLE DATA SET
C   W[]   WEIGHTS + BIASES
C   ERR[] ERROR VECTOR
C   S1[]  SENSITIVITY OF HIDDEN LAYER
C   S2[]  SENSITIVITY OF FINAL LAYER
C
C  FUNCTION CALL:
C   FPASS()
C
      DOUBLE PRECISION A,G,W,ERR,SUMS,S1,S2,NET,TOTERR
      INTEGER NINP,NOUT,NHID,NPAT,LP,NW,MNHID,MNOUT,MNW,NDIMA,
     * STEP,LP2
      DIMENSION A(NDIMA),G(MNW),W(MNW),ERR(MNOUT),S1(MNHID),S2(MNOUT)
C
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
C--CALL SUBROUTINE FPASS TO COMPUTE ERROR IN THE FIRST PASS
C
      CALL FPASS(W,A,ERR,TOTERR,MNOUT,MNW,NDIMA)
C
C     COMPUTE SENSITIVITIES OF EACH LAYER
C COMPUTE SENSITIVITIES OF LAYER2
C
      STEP=((NINP+1)*NHID)+1
      DO 300 K=1,NOUT
       NET=0.0D0
       DO 310 J=(NINP+1),(NINP+NHID)
        NET=NET+(W(STEP)*A(J))
        STEP=STEP+1
310     CONTINUE
       NET=NET+W((NHID*(NINP+1))+(NOUT*NHID)+K)
       S2(K)=-2.0D0*DF2(NET)*ERR(K)
300   CONTINUE
C
C COMPUTE SENSITIVITIES OF LAYER1
C COMPUTE OUTPUT OF LAYER1
C
      STEP=1
      DO 400 K=1,NHID
       NET=0.0D0
       DO 410 J=1,NINP
        NET=NET+(W(STEP)*A(J))
        STEP=STEP+1
410     CONTINUE
       SUMS=0.0D0
       DO 420 I=1,NOUT
        SUMS=SUMS+((W( (NHID*(NINP+1))+((K*NOUT)-(NOUT-I))) )
     &   * S2(I))
420     CONTINUE
       NET=NET+W((NHID*NINP)+K)
       S1(K)=DF1(NET)*SUMS
400   CONTINUE
```

```
C
C***COMPUTE NEW WEIGHTS & BIASES***
C COMPUTE WEIGHTS LAYER 2
C
    CNTS=1
    CNTA=1
    DO 500 K=1,(NHID*NOUT)
     G((NHID*(NINP+1))+K)= (S2(CNTS) * A(NINP+CNTA))
     CNTA=CNTA+1
     IF (MOD(K,NHID).EQ.0) THEN
       CNTA=1
       CNTS=CNTS+1
     ENDIF
500   CONTINUE
C
C COMPUTE BIASES LAYER2
C
    DO 510 K=1,NOUT
     G( (NHID*(NINP+1))+(NHID*NOUT)+K) =  S2(K)
510   CONTINUE
C
C COMPUTE WEIGHTS LAYER1
C
    CNTA=1
    CNTS=1
    DO 520 K=1,(NINP*NHID)
     G(K)= (S1(CNTS)*A(CNTA))
     CNTA=CNTA+1
     IF (MOD(K,NINP).EQ.0) THEN
      CNTA=1
      CNTS=CNTS+1
      ENDIF
520    CONTINUE
C
C COMPUTE BIASES LAYER1
C
     DO 530 K=1,NHID
      G((NINP*NHID)+K)=(S1(K))
530    CONTINUE
     RETURN
C
C END OF GRAD()
C
    END
    SUBROUTINE FPASS(W,A,ERR,TOTERR,MNOUT,MNW,NDIMA)
C
C  SUBROUTINE FPASS()
C  HENDRA TIO @ O.S.U.
C  THIS ROUTINE DOES FORWARD PASS, WHERE THE COMPUTATION OF
C  ERROR IS DONE. FPASS() MAKING A FORWARD CALCULATION FROM
C  INPUT LAYER UP TO FINAL LAYER. THE OUTPUT OF EACH LAYER
C  IS INSERTED TO ARRAY A[]. WHERE A[] CONSISTS OF [INP+HIDOUT
C  +NETOUT+ACTUAL_OUTPUT], INP=NETWORK INPUT, HIDOUT= HIDDEN
C  OUTPUT GENERATED BY THIS FPASS, NETOUT=NETWORK OUTPUT BASED
```

```
C  ON OUTPUT GENERATED BY NETWORK, ACTUAL_OUTPUT= DESIRED NETWORK
C  OUTPUT (GIVEN IN A DATA SET)
C
      INTEGER STEP,NINP,NHID,NOUT,NPAT,NW,LP,MNOUT,MNW,NDIMA,LP2
      DOUBLE PRECISION W,A,ERR,SUM,TOTERR
      DIMENSION W(MNW),A(NDIMA),ERR(MNOUT)
C
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
C  COMPUTE OUTPUT OF LAYER1
C
      STEP=1
      DO 110 K=1,NHID
       SUM=0.0D0
       DO 120 J=1,NINP
        SUM=SUM+(W(STEP)*A(J))
        STEP=STEP+1
120    CONTINUE
C
C  COMPUTE THE NET INPUT
C  TOTAL WP + BIAS
C
       SUM=SUM+W((NHID*NINP)+K)
C
C  COMPUTE THE OUTPUT OF LAYER1
C
       A(NINP+K)=F1(SUM)
110   CONTINUE
C
C COMPUTE OUTPUT OF LAYER2
C
      STEP=((NHID*(NINP+1))+1)
      DO 130 K=1,NOUT
       SUM=0.0D0
       DO 140 J=(NINP+1),(NINP+NHID)
        SUM=SUM+(W(STEP)*A(J))
        STEP=STEP+1
140    CONTINUE
C
C  COMPUTE THE NET INPUT
C  TOTAL WP + BIAS
C
       SUM=SUM+W((NHID*(NINP+1))+(NOUT*NHID)+K)
C
C  COMPUTE THE OUTPUT OF LAYER2
C
       A(NINP+NHID+K)=F2(SUM)
130   CONTINUE
C
C***ACTUAL_OUTPUT GENERATED IS IN A(J)***
C***ERROR IS IN ERR(J)***
C
      DO 150 J=1,NOUT
```

```
      ERR(J)=(A(NINP+NHID+NOUT+J) - A(NINP+NHID+J))
      TOTERR=TOTERR+(ERR(J)**2.0D0)
150   CONTINUE
      RETURN
      END
C
C END OF FPASS()
C
      SUBROUTINE SD(EGOAL,ETA,MODE,A,W,ERR,MAXITR,GAMMA,
     *  MNOUT,MNW,NDIMA,MNPAT,MODE2)
C
C SUBROUTINE SD
C HENDRA TIO @ OKLAHOMA STATE UNIVERSITY, COMPUTER SCIENCE DEPT.
C  SD ( STEEPEST DESCENT)
C
C THIS PROGRAM IS THE BACKPROPAGATION OF ERROR BY STEEPEST DESCENT.
C THE METHOD USES THE NEGATIVE OF THE GRADIENT TIMES THE STEP SIZE
C FOR UPDATING THE PARAMETERS. THE STEP SIZE IS DETERMINE BY FINDING
C THE MINIMUM POINT ALONG THE SEARCH DIRECTION (-GRADIENT). THIS IS
C DONE BY GOLDEN SECTION SEARCH.
C
C THIS ROUTINE IS CALLED BY :
C    DRIVER()
C
C THIS ROUTINE CALLS :
C    LSRCH() AT THIS FILE
C    FCN() AT DRIVER.F
C    GRAD() AT NN.F
C    FPASS() AT NN.F
C
C THIS PROGRAM DEALS WITH BOTH BATCHING AND INCREMENTAL MODE. THE
C FOLLOWING ARE THE INFORMATION NEEDED FOR CALLING THIS FUNCTION SD()
C  W        : WEIGHTS
C  A        : DATA POINTS (THE WHOLE DATA POINTS INCLUDED)
C  EGOAL    : ERROR GOAL
C  GAMMA    : MOMENTUM FACTOR
C  MODE     : MODE FOR BATCHING=1 OR INCREMENTAL=0
C  MAXITR   : MAXIMUM ITERATION ASSIGNED
C  NW       : NUMBER OF PARAMETERS INVOLVED
C VARIABLES USED INSIDE THE SD()
C  G        : GRADIENT
C  GTOT     : TOTAL GRADIENT FOR BATCHING
C  DWOLD    : PREVIOUS PARAMETERS CHANGE
C  DW       : CURRENT PARAMETERS CHANGE
C
      DOUBLE PRECISION ETA,W,G,A,AA,ERR,GTOT,DWOLD,DW,GAMMA,P,
     * EGOAL,TOTERR,LSRCH
      INTEGER BATCH,MODE,MAXITR,EPOCH,NINP,NHID,NOUT,NPAT,NW,
     * LP,NDIMA,MNW,MNPAT,MNOUT,LP2,MODE2
      DOUBLE PRECISION DOLDR,DNEWR
      INTEGER CLASS,IOPTST,LPX,LPY,LPZ,LCNT,LCNV
      REAL TIMES(2),TOTAL
C
      DIMENSION W(MNW),G(MNW),A(MNPAT,NDIMA),AA(NDIMA),ERR(MNOUT),
```

```
     * GTOT(MNW),DWOLD(MNW),DW(MNW),P(MNW)
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
      BATCH=1
      DOLDR=9.9D9
      IF(MODE.EQ.BATCH) LCNV=1
      IF(MODE.NE.BATCH) LCNV=3
      LCNT=0
C
      DO 15 J=1,NW
       DWOLD(J)=0.D0
15    CONTINUE
      EPOCH=0
C
C***ITERATE THROUGH EACH PATTERN***
C
20     CONTINUE
       DO 25 J=1,NW
        GTOT(J)=0.0D0
25     CONTINUE
C
       DO 100 I=1,NPAT
C
C  STORE ACTUAL OUTPUT (T) AT THE END ARRAY 'A'
C
        DO 110 K=1,NINP
         AA(K)=A(I,K)
110      CONTINUE
        DO 120 J=1,NOUT
         AA(J+NINP+NHID+NOUT)=A(I,J+NINP+NHID+NOUT)
120      CONTINUE
       CALL GRAD(G,W,AA,ERR,TOTERR,NHID,NOUT,NW,NDIMA)
C
C--IF MODE IS INCREMENTAL
C
       IF(MODE.NE.BATCH) THEN
        DO 140 JJ=1,NW
         P(JJ)=-G(JJ)
140      CONTINUE
C        ETA=LSRCH(W,A,P,MODE,I,NOUT,NW,NDIMA,MNPAT)
        DO 215 J=1,NW
         DW(J)=GAMMA*DWOLD(J)-((1.D0 - GAMMA) * ETA * G(J))
         W(J)=W(J)+DW(J)
         DWOLD(J)=DW(J)
215      CONTINUE
C
C--IF MODE IS BATCH
C
       ELSE
        DO 225 J=1,NW
         GTOT(J)=GTOT(J)+G(J)
225      CONTINUE
       ENDIF
```

```
100   CONTINUE
C
C  COMPUTE BATCHING GRADIENT
C
     IF(MODE.EQ.BATCH) THEN
      DO 230 JJ=1,NW
       GTOT(JJ)=GTOT(JJ)/REAL(NPAT)
       P(JJ)=-GTOT(JJ)
230    CONTINUE
      ETA=LSRCH(W,A,P,MODE,I,NOUT,NW,NDIMA,MNPAT)
      DO 235 J=1,NW
       DW(J)=GAMMA*DWOLD(J)-((1.D0 - GAMMA)*ETA*GTOT(J))
       W(J)=W(J)+DW(J)
       DWOLD(J)=DW(J)
235    CONTINUE
     ENDIF
C
C  END OF BATCHING
C
     EPOCH=EPOCH+1
C
C--IF MODE IS INCREMENTAL
C
     TOTERR=0.D0
     CALL FCN(W,A,TOTERR,NOUT,NW,NDIMA.MNPAT)
       WRITE(LP2,299)EPOCH,TOTERR
299      FORMAT(/'ITER=' I5, ', SSE=',G15.7)
     IF (MOD(EPOCH,50).EQ.0) THEN
       WRITE(LP,301)EPOCH,TOTERR
301      FORMAT(/'ITER=' I5, ', SSE=',G15.7)
     ELSE IF(TOTERR.LT. EGOAL) THEN
       WRITE(LP,302)EPOCH,TOTERR
302      FORMAT(/'ITER=' I5, ', SSE=',G15.7)
     ENDIF
C***END OF EACH ITERATION***
C UNTIL TOTERR < EGOAL & EPOCH =< MAXITER
     IF (EPOCH.GT.MAXITR)  GOTO 1000
     IF ((MODE2.EQ.1) .AND. (MOD(EPOCH,50).EQ.0))THEN
      IF(NPAT.EQ.LCLSF(W,A,NOUT,NW,NDIMA,MNPAT)) GOTO 1000
     ENDIF
C
C  VALIDATION
C
     IF((IOPTST.GE.2).AND.(MOD(EPOCH,50).EQ.0)) THEN
      DNEWR=VALID(W,MNOUT,NW,NDIMA,MNPAT,MODE2)
      IF ((DNEWR.GT.DOLDR).AND.(LCNT.GE.LCNV)) THEN
         GOTO 1000
      ELSEIF ((DNEWR.GT.DOLDR).AND.(LCNT.LT.LCNV)) THEN
         LCNT=LCNT+1
      ELSE
         LCNT=0
      ENDIF
      DOLDR=DNEWR
     ENDIF
```

```
C
      GOTO 20
1000  CONTINUE
      RETURN
      END
C
      DOUBLE PRECISION FUNCTION LSRCH(W,A,P,MODE,INDEX,MNOUT,
     *  MNW,NDIMA,MNPAT)
C
C  FUNCTION LSRCH
C  HENDRA TIO. @ O.S.U.
C
C  CALLED BY: SD() AT THIS FILE (SD.F).
C
C  FUNCTION THAT PERFORM THE GOLDEN SECTION SEARCH
C  ADOPTED FROM 'NEURAL NETWORK DESIGN' BY HAGAN, DEMUTH, BEALE
C  , PWS PUBLISHING (1995), CHAPTER 12, PP 15-19.
C  THIS ROUTINE PERFORMS BOTH BRACKETING THE MINIMUM POINT
C  AND SEARCHING THAT PARTICULAR POINT WITH A GIVEN SEARCH
C  DIRECTION, P. THIS ROUTINE DONE BOTH LINE SEARCH IN BATCH
C  AND ONLINE MODE. THE DIFFERENT BETWEEN BOTH THAT APPLIED
C  HERE IS THAT THE BATCHING CALLS FCN() FOR THE FUNCTION EVAL
C  (BATCH FORM OF FPASS), AND THE ONLINE CALLS FPASS() (WHICH
C  FED IN THE INPUT (A(,)) ONE AT A TIME BASED ON THE VALUE
C  OF INDEX.
C
      INTEGER NW,NINP,NHID,NOUT,NPAT,LP,MODE,INDEX,MNOUT,MNW,
     * MNPAT,NDIMA,LP2
      DOUBLE PRECISION W,P,ETA,X(100),Y(100),Z(100),
     * ZZ(100),FX,FY,FYOLD,FZ,FZZ,RGOLD,WRES,TOL,A,
     * AA,DUMMY
      DIMENSION W(MNW),P(MNW),WRES(MNW),A(MNPAT,NDIMA),AA(NDIMA),
     * DUMMY(MNOUT)
C
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
C INITIALIZE TOL,ETA AND GOLDEN FRACTION
C ETA IS ASSIGNED TO .075. THIS VALUE CAN BE ASSIGNED A SMALLER
C VALUE, BUT IT COULD CAUSE MORE FUNCTION EVALUATIONS.
C
      TOL=1.D-6
      ETA=0.075D0
      RGOLD=0.618D0
C
C EVALUATE MINIMUM POINT AT SEVERAL INTERVAL
C
      I=0
100   CONTINUE
       I=I+1
       K=2**(I-1)
       Y(I)=ETA*REAL(K)
       DO 105 J=1,NW
        WRES(J)=W(J)+Y(I)*P(J)
105    CONTINUE
```

```
      FYOLD=FY
      FY=0.D0
C
C  BATCHING
C
      IF (MODE.EQ.1) THEN
       CALL FCN(WRES,A,FY,NOUT,NW,NDIMA,MNPAT)
      ELSE
C
C  INCREMENTAL
C
       DO 106 KK=1,NINP
        AA(KK)=A(INDEX,KK)
106     CONTINUE
       DO 107 JJ=1,NOUT
        AA(JJ+NINP+NHID+NOUT)=A(INDEX,JJ+NINP+NHID+NOUT)
107     CONTINUE
       CALL FPASS(WRES,AA,DUMMY,FY,NOUT,NW,NDIMA)
      ENDIF
C
      IF (I.GT.1) THEN
       IF (FY.GT.FYOLD) THEN
        Y(1)=Y(I)
        IF (I.EQ.2) THEN
C         LSRCH=ETA
C         RETURN
          X(1)=ETA/2.D0
        ENDIF
        GOTO 110
       ELSE
        IF (I.GT.2) THEN
         X(1)=Y(I-2)
        ELSE
         X(1)=Y(I-1)
        ENDIF
       ENDIF
      ENDIF
      GOTO 100
110   CONTINUE
C
C  BRACKETING THE MINIMUM POINT
C
      Z(1)=X(1)+(1.0D0-RGOLD)*(Y(1)-X(1))
      ZZ(1)=Y(1)-(1.0D0-RGOLD)*(Y(1)-X(1))
      DO 120 J=1,NW
       WRES(J)=W(J)+Z(1)*P(J)
120   CONTINUE
      FZ=0.D0
C
C  BATCHING
C
      IF (MODE.EQ.1) THEN
       CALL FCN(WRES,A,FZ,MNOUT.MNW,NDIMA,MNPAT)
      ELSE
```

```
C
C  INCREMENTAL
C
      DO 121 KK=1,NINP
       AA(KK)=A(INDEX,KK)
121    CONTINUE
      DO 122 JJ=1,NOUT
       AA(JJ+NINP+NHID+NOUT)=A(INDEX,JJ+NINP+NHID+NOUT)
122    CONTINUE
      CALL FPASS(WRES,AA,DUMMY,FZ,NOUT,NW,NDIMA)
     ENDIF
C
     DO 130 J=1,NW
      WRES(J)=W(J)+ZZ(1)*P(J)
130   CONTINUE
     FZZ=0.D0
C  BATCHING
     IF (MODE.EQ.1) THEN
      CALL FCN(WRES,A,FZZ,MNOUT,MNW,NDIMA,MNPAT)
     ELSE
C
C  INCREMENTAL
C
      DO 131 KK=1,NINP
       AA(KK)=A(INDEX,KK)
131    CONTINUE
      DO 132 JJ=1,NOUT
       AA(JJ+NINP+NHID+NOUT)=A(INDEX,JJ+NINP+NHID+NOUT)
132    CONTINUE
      CALL FPASS(WRES,AA,DUMMY,FZZ,NOUT,NW,NDIMA)
     ENDIF
C
     K=0
200   CONTINUE
      K=K+1
      IF (FZ.LT.FZZ) THEN
       X(K+1)=X(K)
       Y(K+1)=ZZ(K)
       ZZ(K+1)=Z(K)
       Z(K+1)=X(K+1)+(1.0D0-RGOLD)*(Y(K+1)-X(K+1))
       FZZ=FZ
       DO 210 J=1,NW
        WRES(J)=W(J)+Z(K+1)*P(J)
210     CONTINUE
       FZ=0.D0
C
C  BATCHING
C
       IF (MODE.EQ.1) THEN
        CALL FCN(WRES,A,FZ,MNOUT,MNW,NDIMA,MNPAT)
       ELSE
C
C  INCREMENTAL
C
```

```
      DO 211 KK=1,NINP
       AA(KK)=A(INDEX,KK)
211     CONTINUE
      DO 212 JJ=1,NOUT
       AA(JJ+NINP+NHID+NOUT)=A(INDEX,JJ+NINP+NHID+NOUT)
212     CONTINUE
      CALL FPASS(WRES,AA,DUMMY,FZ,NOUT,NW,NDIMA)
     ENDIF
C
    ELSE
     X(K+1)=Z(K)
     Y(K+1)=Y(K)
     Z(K+1)=ZZ(K)
     ZZ(K+1)=Y(K+1)-(1.D0-RGOLD)*(Y(K+1)-X(K+1))
     FZ=FZZ
     DO 220 J=1,NW
      WRES(J)=W(J)+ZZ(K+1)*P(J)
220     CONTINUE
     FZZ=0.D0
C
C  BATCHING
C
     IF (MODE.EQ.1) THEN
      CALL FCN(WRES,A,FZZ,MNOUT,MNW,NDIMA,MNPAT)
     ELSE
C
C  INCREMENTAL
C
      DO 221 KK=1,NINP
       AA(KK)=A(INDEX,KK)
221     CONTINUE
      DO 222 JJ=1,NOUT
       AA(JJ+NINP+NHID+NOUT)=A(INDEX,JJ+NINP+NHID+NOUT)
222     CONTINUE
      CALL FPASS(WRES,AA,DUMMY,FZZ,NOUT,NW,NDIMA)
     ENDIF
    ENDIF
    IF ((Y(K+1)-X(K+1)).LT.TOL) THEN
     GOTO 300
    ENDIF
    GOTO 200
300  CONTINUE
   LSRCH=X(K+1)
   RETURN
   END
   SUBROUTINE LM(EGOAL,W,A,MNOUT,MNW,NDIMA,MNPAT,MAXJ,MAXW,OPT,MODE2)
C  HENDRA TIO @ OKLAHOMA STATE UNIVERSITY,
C  COMPUTER SCIENCE DEPT.
C
C  THIS ROUTINE IS THE LEVENBERG AND LEVENBERG-MARQUARDT
C  IMPLEMENTED ALGORITHM. THE PASSSED LOGICAL NUMBER, OPT,
C  SPECIFIED WHICH ALGORITHM WILL BE USED. FOR E.Q. OPT=3
C  TELLS THE ROUTINE TO RUN THE LEVENBERG METHOD, AND OPT=4
C  RUN THE LEVENBERG-MARQUARDT METHOD.
```

```
C  THE FORMULA FOR THIS METHOD IS:
C    DXI=INV(JTJ+LAMBDA*I) JTV,
C  WHERE DXI IS THE CHANGE OF PARAMETERS, JTJ IS THE HESSIAN,
C  LAMBDA IS THE SCALING FACTOR, I IS THE IDENTITY MATRIX, AND
C  JTV IS THE GRADIENT.
C  THIS ROUTINE CALLS :
C    JACT()          - THE ROUTINE THAT EVALUATE THE JACOBIAN
C    MATRIX AND TRANSPOSED IT. THE JACOBIAN MATRIX IS EVALUATED
C    BY CALLING THE SSTY(), WHERE THE SENSITIVITIES OF THE EACH
C    LAYER IS COMPUTED.
C    GTJTJ()         - THE ROUTINE THAT EVALUATE THE HESSIAN
C    BY MULTIPLYING THE JACOBIAN TRANSPOSED AND THE JACOBIAN.
C    GTGRAD()        - THE ROUTINE THAT COMPUTE THE GRADIENT
C    TERM OF THE METHOD. THE GRADIENT CONSISTS OF THE MULTIPLICATION
C    OF THE JACOBIAN AND THE ERROR VECTOR. THIS GRADIENT IS ALSO
C    CALLED THE SD TERM OF THE METHOD.
C    LSOLV()         - A LINEAR SYSTEM SOLVER FOR COMPUTING THE
C    AX=B, WHERE X IS THE SOLUTION, A IS THE HESSIAN AND B IS THE
C    GRADIENT.
C
C  THE ROW OF A IS THE INPUT+NHID_OUT+NET_OUT+DESIRED
C  THE COLUMN IS THE SERIES OF I/O PATTERNS
C  W = WEIGHTS + BIASES
C  NINP NUMBER OF INPUT,NHID NUMBER OF HIDDEN NODE/S
C  NOUT NUMBER OF OUTPUT, NPAT NUMBER OF PATTERN I/O
C  JT = TRANSPOSED OF JACOBIANS AND JTJ IS HESSIAN.
C
C
      INTEGER NINP,NHID,NOUT,NPAT,ITER,NW,MNOUT,MNW,NDIMA,MNPAT,
     * MAXJ,LP,MAXW,NRANK,OPT,LP2,MAXITER,MODE2
      INTEGER CLASS,IOPTST,LPX,LPY,LPZ
      REAL TIMES(2),TOTAL
      DOUBLE PRECISION A,W,WNEW,TOL,INC,DEC,MAXLAM,LAMBDA,PSMAL,
     * SSE,SSENEW,AA,JT,JTJ,JTV,JTJ2,JTV2,ERROR,DUMMY,ERR,
     * DNEWR,DOLDR,EGOAL
C
      DIMENSION A(MNPAT,1),
     * W(MNW),
     * WNEW(MNW),
     * AA(NDIMA),JT(MAXJ,MNW),JTJ(MNW,MNW),JTV(MNW),
     * JTJ2(280,280),JTV2(MAXW),
     * ERROR(MNPAT,MNOUT),DUMMY(MNOUT),ERR(MNOUT)
C
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
C  INITIALIZATION
C
      TOL=1.D-6
      INC=1.D1
      DEC=1.D-1
      MAXLAM=1.0D10
      LAMBDA=1.D-3
      ITER=0
```

```
      MAXITER=100
      DOLDR=9.9D9
C
C  TOP OF ITERATION FOR SUCCESSFUL IN SSE REDUCTION
C
56    CONTINUE
      SSE=0.D0
      DO 5 I=1,NW
       JTV(I)=0.D0
5     CONTINUE
C
      DO 100 I=1,NPAT
       DO 110 J=1,NINP+NHID+(2*NOUT)
        AA(J)=A(I,J)
110    CONTINUE
C
C  GET SSE
C
       CALL FPASS(W,AA,ERR,SSE,MNOUT,MNW,NDIMA)
       DO 111 K=1,NOUT
        ERROR(I,K)=ERR(K)
111    CONTINUE
C
C  COMPUTE JT
C
       CALL JACT(W,AA,NHID,MNOUT,MNW,NDIMA,MAXJ,JT,I)
C
C  COMPUTE JTV
C
       CALL GTGRAD(NINP,NHID,NOUT,NPAT,
     &  JT,JTV,I,ERROR,MAXJ,MNW,MNPAT,MNOUT)
100    CONTINUE
C
C  COMPUTE JTJ
C
       CALL GTJTJ(NINP,NHID,NOUT,NPAT,JT,JTJ,MAXJ,MNW)
C
C  TOP OF ITERATION FOR UNSUCCESSFUL IN SSE REDUCTION
C  INCLUDE LAMBDA*I TO JTJ
C
156    CONTINUE
      DO 120 I=1,NW
       DO 130 J=1,NW
C
C  EVALUATE ONLY THE DIAGONAL
C
        IF (I EQ.J) THEN
C SWITCH FROM LEV1 TO LEV2
C  LEV I
         IF (OPT.EQ.3) THEN
          JTJ2(I,J)=JTJ(I,J)+LAMBDA
C  LEVII
         ELSE
          JTJ2(I,J)=JTJ(I,J)+((1.D0+JTJ(I,J))*LAMBDA)
```

```
C       JTJ2(I,J)=JTJ(I,J)*(LAMBDA+1.D0)
        ENDIF
       ELSE
        JTJ2(I,J)=JTJ(I,J)
       ENDIF
C
C END OF DIAGONAL EVALUATION
C
      JTJ2(I,J)=1.D0*JTJ2(I,J)
130    CONTINUE
      JTV2(I)=1.0D0*JTV(I)
120    CONTINUE
C
C  COMPUTE DELX = -(JTJ +LAMBDA*I)^-1 + JTV
C
      CALL LSOLV(JTJ2,JTV2,NW,MAXW,NRANK,PSMAL)
      IF (NRANK.LT.NW) THEN
       WRITE(LP,131) NRANK
131    FORMAT(/' SINGULAR SYSTEM, RANK=',I3)
      ENDIF
C
C  UPDATE NEW PARAMETERS
C
      DO 160 I=1,NW
       WNEW(I)=W(I)-JTV2(I)
160    CONTINUE
C
C  COMPUTE NEW SSE
C
      SSENEW=0.D0
      DO 200 I=1,NPAT
       DO 210 J=1,NINP+NHID+(2*NOUT)
        AA(J)=A(I,J)
210    CONTINUE
C
C  GET SSENEW
C
      CALL FPASS(WNEW,AA,DUMMY,SSENEW,MNOUT,MNW,NDIMA)
200    CONTINUE
C
C       WRITE(LP,*) 'SSE=',SSE, 'SSENW=',SSENEW,'ITER=',ITER
C
C  EVALUATE NEW SSE
C
      IF (SSENEW .LT. SSE) THEN
        LAMBDA=LAMBDA*DEC
C
C  UPDATE PARAMETERS
C
        DO 230 I=1,NW
         W(I)=WNEW(I)
230      CONTINUE
        ITER=ITER+1
        WRITE(LP,231)ITER-1,SSE,LAMBDA
```

```
231    FORMAT(/'ITER=' I5, ', SSE=',G15.7,', LAMBDA',G15.4)
      WRITE(LP2,232)ITER-1,SSE,LAMBDA
232    FORMAT(/'ITER=' I5, ', SSE=',G15.7,', LAMBDA',G15.4)
      IF (ITER.GE.MAXITER) GOTO 1000
      IF (MODE2.EQ.1) THEN
       IF(NPAT.EQ.LCLSF(W,A,NOUT,NW,NDIMA,MNPAT)) GOTO 1000
      ENDIF
C  VALIDATION
      IF(IOPTST.GE.2) THEN
       DNEWR=VALID(W,MNOUT,NW,NDIMA,MNPAT,MODE2)
       IF (DNEWR.GT.DOLDR) THEN
          GOTO 1000
       ENDIF
       DOLDR=DNEWR
      ENDIF
      IF(SSE.LE.EGOAL) GOTO 1000
C
     ELSEIF (SSENEW .GE. SSE) THEN
      IF (LAMBDA.EQ.0) THEN
       LAMBDA=TOL
      ENDIF
      IF (LAMBDA.LE.MAXLAM) THEN
       LAMBDA=LAMBDA*INC
      ELSE
       GOTO 1000
      ENDIF
      GOTO 156
     ENDIF
     GOTO 56
1000  RETURN
     END
C
C  END OF SUBROUTINE LM
C
     SUBROUTINE JACT(W,AA,MNHID,MNOUT,MNW,NDIMA,MAXJ,JT,IND)
C  ROUTINE TO COMPUTE TRANSPOSED JACOBIANS FOR
C  A SINGLE COLUMN, EACH CALL CREATE A COLUMN
C  OF JACOBIAN THAT CONTAINS A SINGLE PATTERN
C
     DOUBLE PRECISION W,AA,JT,S1,S2
     INTEGER NINP,NHID,NOUT,NPAT,IND,NODE
     INTEGER CNTA,CNTS,MNHID,MNOUT,MNW,NDIMA,MAXJ,LP,NW
     DIMENSION W(MNW),AA(NDIMA),JT(MAXJ,MNW),S1(MNHID),S2(MNOUT)
     COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP
C
     DO 5 NODE=1,NOUT
C
C  CALL SSTY() TO COMPUTE S1 & S2
C
     CALL SSTY(S1,S2,W,AA,NINP,NHID,NOUT,MNW,NDIMA,NODE)
C
C  WEIGHTS LAYER1
C
     CNTA=1
```

```
      CNTS=1
      DO 10 K=1,NINP*NHID
       JT(((IND-1)*NOUT)+NODE,K)=S1(CNTS)*AA(CNTA)
       CNTA=CNTA+1
       IF (MOD(K,NINP).EQ.0) THEN
        CNTA=1
        CNTS=CNTS+1
       ENDIF
10     CONTINUE
C
C  BIASES LAYER1
C
      DO 20 K=1,NHID
       JT(((IND-1)*NOUT)+NODE,(NINP*NHID)+K)=S1(K)
20     CONTINUE
C
C  WEIGHTS LAYER2
C
       CNTS=1
       CNTA=1
       DO 30 K=1,NHID*NOUT
        JT(((IND-1)*NOUT)+NODE,(NHID*(NINP+1))+K)=S2(CNTS)*AA(NINP+CNTA)
        CNTA=CNTA+1
        IF (MOD(K,NHID).EQ.0) THEN
         CNTA=1
         CNTS=CNTS+1
        ENDIF
30      CONTINUE
C
C  BIASES LAYER2
C
       DO 40 K=1,NOUT
        JT(((IND-1)*NOUT)+NODE,(NHID*(NINP+1))+(NHID*NOUT)+K) = S2(K)
40     CONTINUE
5      CONTINUE
       RETURN
       END
C
       SUBROUTINE GTGRAD(NINP,NHID,NOUT,
      &    NPAT,JT,JTV,IND,ERR,MAXJ,MNW,MNPAT,MNOUT)
C      GTGRAD()        - THE ROUTINE THAT COMPUTE THE GRADIENT
C      TERM OF THE METHOD. THE GRADIENT CONSISTS OF THE MULTIPLICATION
C      OF THE JACOBIAN AND THE ERROR VECTOR. THIS GRADIENT IS ALSO
C      CALLED THE SD TERM OF THE METHOD.
       DOUBLE PRECISION JT,JTV,ERR
       INTEGER NINP,NHID,NOUT,NPAT,IND,NW,NODE,MAXJ,MNW,MNPAT,MNOUT
       DIMENSION ERR(MNPAT,MNOUT),JTV(MNW),JT(MAXJ,MNW)
C
       CNTA=1
       NW=((NINP+1)*NHID)+(NOUT*(NHID+1))
       DO 10 I=1,NW
        DO 5 NODE=1,NOUT
         JTV(I)=JTV(I)+(JT(((IND-1)*NOUT)+NODE,I)*ERR(IND,NODE))
5       CONTINUE
```

```
10  CONTINUE
    RETURN
    END
C
C234567890
C
C*************************************************************************
     SUBROUTINE LSOLV (A,BX,N,LDIM,NRANK,PSMAL)
C
C LSOLV 1.6       APRIL 1992
C
C J. P. CHANDLER, COMPUTER SCIENCE DEPARTMENT,
C   OKLAHOMA STATE UNIVERSITY
C
C LSOLV SOLVES A SYSTEM OF LINEAR EQUATIONS USING GAUSSIAN
C ELIMINATION WITH PARTIAL PIVOTING.
C IF THE MATRIX OF COEFFICIENTS IS SINGULAR, LSOLV COMPUTES
C THE SOLUTION THAT WOULD RESULT FROM MULTIPLYING A RAO
C PSEUDOINVERSE OF THE COEFFICIENT MATRIX TIMES THE VECTOR OF
C CONSTANTS.
C C. R. RAO AND S. K. MITRA, -GENERALIZED INVERSE OF MATRICES
C   AND ITS APPLICATIONS- (WILEY, 1971), PAGE 212
C
C N IS THE NUMBER OF EQUATIONS IN THE LINEAR SYSTEM.
C ON ENTRY, A(*,*) CONTAINS THE MATRIX OF COEFFICIENTS AND
C BX(*) CONTAINS THE VECTOR OF CONSTANTS (THE RIGHTHAND
C SIDES).
C ON EXIT, BX(*) CONTAINS THE SOLUTION VECTOR AND A(*,*)
C CONTAINS GARBAGE.
C LDIM IS THE VALUE OF THE DIMENSIONS OF THE ARRAYS A AND BX.
C THE VALUE OF N MUST NOT EXCEED THE VALUE OF LDIM.
C NRANK RETURNS THE RANK OF THE MATRIX A.
C IF NRANK .LT. N THEN THE MATRIX A WAS SINGULAR.
C PSMAL RETURNS THE PIVOT, IF ANY, THAT HAD THE SMALLEST
C NONZERO MAGNITUDE.
C*************************************************************************
C
C
     DOUBLE PRECISION A,BX,ZABS,ARG,BIGA,TEMP,EM,SUM,PSMAL,
    *  RZERO
C
     DIMENSION A(LDIM,N),BX(N)
C
     ZABS(ARG)=DABS(ARG)
C
C CHECK FOR AN INVALID VALUE OF N OR LDIM.
C
     RZERO=0.0D0
     NRANK=-1
     PSMAL=RZERO
     IF(N.LT.1 .OR. N.GT.LDIM) GO TO 2495
C
C TRIANGULARIZE THE MATRIX A.
C
```

```
      NRANK=0
      IF(N.LT.2) GO TO 2470
      NMU=N-1
      DO 2460 J=1,NMU
C
C  SEARCH COLUMN J FOR THE PIVOT ELEMENT.
C
            BIGA=RZERO
            DO 2410 K=J,N
              TEMP=ZABS(A(K,J))
              IF(TEMP.LE.BIGA) GO TO 2410
              BIGA=TEMP
              JPIV=K
 2410       CONTINUE
            IF(BIGA.LE.RZERO) GO TO 2460
            IF(JPIV.LE.J) GO TO 2430
C
C  INTERCHANGE EQUATIONS J AND JPIV.
C
            DO 2420 L=J,N
              TEMP=A(J,L)
              A(J,L)=A(JPIV,L)
              A(JPIV,L)=TEMP
 2420       CONTINUE
            TEMP=BX(J)
            BX(J)=BX(JPIV)
            BX(JPIV)=TEMP
 2430       JPU=J+1
            DO 2450 K=JPU,N
C
C  PERFORM ELIMINATION ON EQUATION K.
C
              EM=A(K,J)/A(J,J)
              IF(EM.EQ.RZERO) GO TO 2450
              DO 2440 L=JPU,N
                A(K,L)=A(K,L)-EM*A(J,L)
 2440         CONTINUE
              BX(K)=BX(K)-EM*BX(J)
 2450       CONTINUE
 2460     CONTINUE
C
C  DO THE BACK SOLUTION.
C
 2470 DO 2490 JINV=1,N
            J=N+1-JINV
            TEMP=A(J,J)
            IF(TEMP.NE.RZERO) GO TO 2475
            BX(J)=RZERO
            GO TO 2490
 2475       NRANK=NRANK+1
            IF(PSMAL.EQ.RZERO .OR. ZABS(TEMP).LT.ZABS(PSMAL))
     *      PSMAL=TEMP
            SUM=RZERO
            IF(J.GE.N) GO TO 2485
```

```
      JPU=J+1
      DO 2480 K=JPU,N
       SUM=SUM+A(J,K)*BX(K)
2480   CONTINUE
C
2485  BX(J)=(BX(J)-SUM)/TEMP
2490  CONTINUE
C
2495 RETURN
C
C END LSOLV
C
      END
      SUBROUTINE GTJTJ(NINP,NHID,NOUT,NPAT,JT,JJ,MAXJ,MNW)
C INPUT:    TRANSPOSED MATRIX JT()() ,ROW,COLUMN
C OUTPUT:   MATRIX JJ()() SQUARE SIZE
C NOTE:     THE INPUT MATRIX JT HAS TO BE TRANSPOSITIONED
C           BEFORE CALLING THIS ROUTINE.
C-------------------------------------------------------------
      DOUBLE PRECISION JT,JJ,RJJ
      INTEGER NINP,NHID,NOUT,NPAT,MAXJ,MNW
      INTEGER ROW,COL,NW
      DIMENSION JT(MAXJ,MNW),JJ(MNW,MNW), RJJ(MAXJ)
C
      ROW=(NHID*(NINP+1))+ (NOUT*(NHID+1))
      COL=NOUT*NPAT
C INITIALIZE JJ
      DO 40 I=1,ROW
       DO 30 J=1,ROW
        JJ(J,I)=0.D0
30     CONTINUE
40    CONTINUE
C
C SET THE SIZE OF SQUARE MATRIX
C
      DO 100 I=1,ROW
       DO 200 J=1,COL
        RJJ(J)=JT(J,I)
        DO 400 L=1,ROW
         JJ(I,L)=JJ(I,L)+JT(J,L)*RJJ(J)
400     CONTINUE
200    CONTINUE
100   CONTINUE
      RETURN
      END
C
C**********************************************************************
      SUBROUTINE SSTY(S1,S2,W,A,NINP,NHID,NOUT,MNW,NDIMA,NODE)
C**********************************************************************
C     ROUTINE THAT COMPUTE THE SENSITIVITIES OF EACH LAYER
C     IN THE FULLY CONNECTED NETWORK.
C     THESE SENSITIVITIES COMPUTATION IS DESIGNED ONLY FOR
C     LEVENBERG & LEV-MARQ, WHICH IS SLIGHTLY DIFFERENT THAN
C     BP. THE DIFFERENCE IS ONLY ON THE OUTPUT LAYER SENSITIVITIES
```

```
C    IMPLEMENTATION.
C
C    TASK  : COMPUTE S1 AND S2
C       S2=-F(N), WHERE F() IS THE DERIVATIVE OF THE ACTIVATION
C          FUNCTION OF LAYER 2, AND N IS THE WP+B.
C          WHEN K=NODE S2(K)=-F(N)
C          O.W. S2(K)=0
C       S1=F(N)(WM+1)S2.
C
C    FUNCTION CALL : NONE
C-----------------------------------------------------------------
     DOUBLE PRECISION A,W,SUMS,S1,S2,NET
     INTEGER NINP,NOUT,NHID,STEP,NODE,NDIMA,MNW
     DIMENSION S2(NOUT),S1(NHID),A(NDIMA),W(MNW)
C
C    COMPUTE SENSITIVITIES OF EACH LAYER
C COMPUTE SENSITIVITIES OF LAYER2
C
     STEP=((NINP+1)*NHID)+1
     DO 300 K=1,NOUT
      NET=0.0D0
      DO 310 J=(NINP+1),(NINP+NHID)
       NET=NET+(W(STEP)*A(J))
       STEP=STEP+1
310    CONTINUE
C
C COMPUTE NETOUTPUT WP+B
C
      NET=NET+W((NHID*(NINP+1))+(NOUT*NHID)+K)
C
C SET THE S2 THAT CORRESPONDED TO THE NODE TO THE ACTUAL SINGLE
C NODE SENSITIVITY COMPUTATION
C AND SET THE REST OF THE SENSITIVITIES THAT NODE REFFERED TO 0
C
      IF (K.EQ.NODE) THEN
       S2(K)=-1.0D0*DF2(NET)
      ELSE
       S2(K)=0.D0
      ENDIF
300   CONTINUE
C
C COMPUTE SENSITIVITIES OF LAYER1
C COMPUTE OUTPUT OF LAYER1
C
     STEP=1
     DO 400 K=1,NHID
      NET=0.0D0
      DO 410 J=1,NINP
       NET=NET+(W(STEP)*A(J))
       STEP=STEP+1
410    CONTINUE
      SUMS=0.0D0
      SUMS=SUMS+((W( (NHID*(NINP+1))+((K*NOUT)-(NOUT-NODE))) )
     &   * S2(NODE))
```

```
C
C COMPUTE NETOUTPUT WP+B
C
     NET=NET+W((NHID*NINP)+K)
     S1(K)=DF1(NET)*SUMS
400  CONTINUE
     RETURN
     END
     SUBROUTINE ILEV(EGOAL,W,A,MNW,MNOUT,MAXPAT,NDIMA,
    *          PO,RLAMDA,MODE)
C
C  THIS SUBROUTINE PERFORM THE INCREMENTAL LEVENBERG
C  TO TRAIN FEEDFORWARD ANNS. THE PO IS THE VALUE THAT
C  FILLED IN THE INITIAL DIAGONAL ELEMENT OF P^-1. THIS
C  VALUE IS PROVIDED BY USER AS PARAMETER CHOICE. THE OTHER
C  PARAMETER CHOICE IS THE RLAMDA (FORGETTING FACTOR).
C  THE FIRST OPERATION IN THIS ROUTINE IS TO INITILIZE
C  THE MATRIX P^-1. SECOND IS TO COMPUTE THE ERROR GENERATED.
C  THEN, THE GRADIENT OF THE ERRORS ARE COMPUTED, BY BACKPROP
C  (CALLED GTALP()). ALPHA CONTAINS THE GRADIENT OF THESE
C  ERROR. MMUL() IS THE MATRIX MULTIPLICATION ROUTINE. WITH
C  THIS ROUTINE PVVP IS COMPUTED, WHERE P STANDS FOR P^-1,
C  V IS THE COMPUTED GRADIENT. IN STEP 5, THE PARAMETER X() IS
C  UPDATED. TRPS() IS THE MATRIX TRANSPOSITION OPERATION.
C
     DOUBLE PRECISION W,A,PO,RLAMDA,AA,SSE,RMU,RBETA,WNEW
     DOUBLE PRECISION PINV,ALPHA,ALPHAT,VP,VVP,PVVP,VPV,RES,
    * DELTV,ERR,ERRNEW,ERROLD,ERRPREV,VRGS,VRSD
     INTEGER NINP,NHID,NOUT,NPAT,NW,NODE,ITER,LP,LP2
     INTEGER MAXITR,MAXPAT,MNW,MNOUT,NDIMA,MODE,CNT
     DOUBLE PRECISION DOLDR,DNEWR,EGOAL
     INTEGER CLASS,IOPTST,LPX,LPY,LPZ,LCNT,LCNV
     REAL TIMES(2),TOTAL
     DIMENSION W(MNW),A(MAXPAT,NDIMA),AA(NDIMA)
     DIMENSION PINV(MNW,MNW),ALPHA(1,MNW),ALPHAT(MNW,1),
    * VP(MNW,1),VVP(MNW,MNW),PVVP(MNW,MNW),VPV(1,1),
    * RES(MNW,MNW),DELTV(MNW),ERR(MNOUT),
    * WNEW(MNW),DW(MNW)
     COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
     COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
C  INIT SCALING PARAMETER
C
     RMU=1.D-3
     RBETA=9.5D-1
     MAXITR=100
     DOLDR=9.9D9
     LCNT=0
     LCNV=1
C
C  STEP:1
C  INIT P^1(0) BY FILLING THE DIAGONAL ELEMENT OF P^1 W/ PO
C
     DO 10 I=1,NW
```

```
      DO 20 J=1,NW
       IF (I.EQ.J) THEN
        PINV(I,J)=PO
       ELSE
        PINV(I,J)=0.D0
       ENDIF
20     CONTINUE
10    CONTINUE
C
C  TOP OF ITERATION
C
      ITER=0
56    CONTINUE
      ITER=ITER+1
      DO 100 I=1,NPAT
C
C  INIT DELTV
C
       DO 115 JK=1,NW
        DELTV(JK)=0.D0
115    CONTINUE
C
C  STEP:2A
C  COMPUTE ERROR
C
       DO 110 J=1,NINP+NHID+(2*NOUT)
        AA(J)=A(I,J)
110    CONTINUE
       CALL FPASS(W,AA,ERR,SSE,NOUT,NW,NDIMA)
C
C  STEP:2B
C  COMPUTE THE GRADIENT OF THE ERROR THROUGH A BACKPROP
C  ITERATIVELY COMPUTING EACH ERROR GENERATED FOR EACH OUTPUT
C  NODE.
C
      DO 120 NODE=1,NOUT
       CALL GTALP(W,AA,NINP,NHID,NOUT,NW,NPAT,NDIMA,ALPHA,NODE)
      DO 333 KL=1,NW
       ALPHA(1,KL)=ALPHA(1,KL)
333    CONTINUE
C
C  STEP:3
C  UPDATE P^-1()
C
C  COMPUTE PVVP
C
       CALL TRPS(ALPHA,ALPHAT,1,NW)
       CALL MMUL(ALPHAT,PINV,VP,NW,1,NW,NW)
       CALL MMUL(ALPHA,VP,VVP,1,NW,NW,1)
       CALL MMUL(PINV,VVP,PVVP,NW,NW,NW,NW)
C
C  COMPUTE VPV
C  VPV IS SCALAR
C
```

```
      CALL MMUL(VP,ALPHA,VPV,NW,1,1,NW)
      VPV(1,1)=VPV(1,1)+RLAMDA
C
      DO 220 J=1,NW
       DO 230 K=1,NW
       PINV(J,K)=PINV(J,K)-PVVP(J,K)/VPV(1,1)
       PINV(J,K)=PINV(J,K)*(1.D0/RLAMDA)
230     CONTINUE
220    CONTINUE
C
C  STEP:4
C  CALCULATE DELTV
C
      DO 300 J=1,NW
       DELTV(J)=DELTV(J)+ALPHA(1,J)*ERR(NODE)
300    CONTINUE
120    CONTINUE
C
C  END OF ITERATION OF EACH NODE IN THE OUTPUT LEVEL
C
C  STEP:5
C  UPDATE PARAMETERS ESTIMATE X(I)
C
      ERROLD=0.D0
      DO 330 J=1,NOUT
       ERROLD=ERROLD+ERR(J)**2.D0
330    CONTINUE
      ERRNEW=0.D0
      CNT=0
356    CONTINUE
      VRGS=(1.D0/((1.0D0+RMU)**2.D0))
      VRSD=(RMU/((1.D0+RMU)**2.D0))
      DO 400 J=1,NW
       DW(J)=0.D0
       DO 410 K=1,NW
        DW(J)=DW(J)+(PINV(K,J)*DELTV(K))
410     CONTINUE
       DW(J)=(VRGS*DW(J))+(VRSD*DELTV(J))
       WNEW(J)=W(J)-DW(J)
400    CONTINUE
      ERRPREV=ERRNEW
      ERRNEW=0.D0
      CALL FPASS(WNEW,AA,ERR,ERRNEW,NOUT,NW,NDIMA)
C
      IF (ERRNEW.GT.ERROLD) THEN
       RMU=RMU/RBETA
       CNT=CNT+1
       GOTO 100
C
      ELSEIF (ERRNEW.LT.ERROLD) THEN
       RMU=RMU*RBETA
       IF (RMU.LT.1.D-40) RMU=1.D-40
       DO 510 J=1,NW
        W(J)=WNEW(J)
```

```
510   CONTINUE
C
C  UPDATE LAMBDA (FORGETTING FACTOR)
C
      RLAMDA=(RLAMDA*9.9D-1)+1.D-2
     ENDIF
C
C  END OF ITERATION THROUGH ALL PATTERN
C
100   CONTINUE
C
C  GOTO TOP OF ITERATION
C
     SSE=0.0D0
     CALL FCN(W,A,SSE,NOUT,NW,NDIMA,MAXPAT)
     PRINT*,'SSE=',SSE,', EPOCH=',ITER,'RMU= ',RMU
     IF (SSE.LE.EGOAL)  GOTO 1000
     IF (ITER.GE.MAXITR) GOTO 1000
     IF (MODE.EQ.1) THEN
      IF(NPAT.EQ.LCLSF(W,A,NOUT,NW,NDIMA,MAXPAT))  GOTO 1000
     ENDIF
C
C  VALIDATION
C
     IF(IOPTST.GE.2) THEN
      DNEWR=VALID(W,MNOUT,NW,NDIMA,MAXPAT,MODE)
      IF((DNEWR.GT.DOLDR).AND.(LCNT.GE.LCNV)) THEN
         GOTO 1000
      ELSEIF((DNEWR.GT.DOLDR).AND.(LCNT.LT.LCNV)) THEN
         LCNT=LCNT+1
      ELSE
         LCNT=0
      ENDIF
      DOLDR=DNEWR
     ENDIF
     GOTO 56
1000  CONTINUE
     RETURN
C
C END OF ILEV
C
     END
     SUBROUTINE MMUL(A,B,C,COLA,ROWA,COLB,ROWB)
C
C    MATRIX MULTIPLICATION
C    INPUT: MATRIX A AND B
C    OUTPUT: MATRIX C(COLB,ROWA)
C
     INTEGER COLA,ROWA,COLB,ROWB
     DOUBLE PRECISION C,A,B
     DIMENSION C(COLB,ROWA),A(COLA,ROWA),B(COLB,ROWB)
C
     IF (COLA.NE.ROWB) THEN
      PRINT*, ' MATRIX MULTIPLICATION CANNOT BE PERFORMED'
```

```
C    PRINT*, ' CHECK THE MATRICES AND MAKE SURE IT IS CORRECT!!!'
     ELSE
C    DO 10 K=1,COLB
      DO 20 I=1,ROWA
       C(K,I)=0.D0
       DO 30 J=1,COLA
        C(K,I)=C(K,I)+A(J,I)*B(K,J)
30     CONTINUE
20    CONTINUE
10   CONTINUE
     ENDIF
     RETURN
C
C END OF MMUL
C
     END
     SUBROUTINE TRPS(A,B,COL,ROW)
C
C  MATRIX TRANSPOSITION OPERATION
C  TAKE A AS AN INPUT MATRIX W/ SIZE OF(COL,ROW)
C  RETURN B W/ SIZE OF (ROW,COL)
C
     INTEGER COL,ROW
     DOUBLE PRECISION A,B
     DIMENSION A(COL,ROW)
     DIMENSION B(ROW,COL)
C
     DO 5 I=1,ROW
      DO 10 J=1,COL
       B(I,J)=A(J,I)
10     CONTINUE
5    CONTINUE
     RETURN
C
C END OF TRPS
C
     END
     SUBROUTINE GTALP(W,AA,NINP,NHID,NOUT,NW,NPAT,NDIMA,JT,NODE)
C
C  THIS ROUTINE COMPUTE THE GRADIENT OF ERROR THAT
C  GENERATED AT EACH OUTPUT NODE IF THERE ARE MORE THAN
C  ONE OUTPUT, OTHERWISE ITS ONLY CALLED ONCE.
C  THIS ROUTINE CALLS SSTY TO COMPUTE THE SENSITIVITIES
C  COMPUTATION SIMILAR TO BP.
C
     DOUBLE PRECISION W
     DOUBLE PRECISION AA
     DOUBLE PRECISION JT
     INTEGER NINP,NHID,NOUT,NPAT,NODE
     INTEGER CNTA,CNTS
     DOUBLE PRECISION S1(NHID)
     DOUBLE PRECISION S2(NOUT)
     DIMENSION W(NW),AA(NDIMA),JT(1,NW)
```

```
C
C  CALL SSTY() TO COMPUTE S1 & S2
     CALL SSTY2(S1,S2,W,AA,NINP,NHID,NOUT,NODE)
C  WEIGHTS LAYER1
     CNTA=1
     CNTS=1
     DO 10 K=1,NINP*NHID
      JT(1,K)=S1(CNTS)*AA(CNTA)
      CNTA=CNTA+1
      IF (MOD(K,NINP).EQ.0) THEN
       CNTA=1
       CNTS=CNTS+1
      ENDIF
10    CONTINUE
C  BIASES LAYER1
     DO 20 K=1,NHID
      JT(1,(NINP*NHID)+K)=S1(K)
20    CONTINUE
C  WEIGHTS LAYER2
      CNTS=1
      CNTA=1
      DO 30 K=1,NHID*NOUT
       JT(1,(NHID*(NINP+1))+K)=S2(CNTS)*AA(NINP+CNTA)
       CNTA=CNTA+1
       IF (MOD(K,NHID).EQ.0) THEN
        CNTA=1
        CNTS=CNTS+1
       ENDIF
30     CONTINUE
C  BIASES LAYER2
      DO 40 K=1,NOUT
       JT(1,(NHID*(NINP+1))+(NHID*NOUT)+K) = S2(K)
40     CONTINUE
      RETURN
C
C END OF GTALP
C
      END
     SUBROUTINE SSTY2(S1,S2,W,A,NINP,NHID,NOUT,NODE)
C     ROUTINE THAT COMPUTE THE SENSITIVITIES OF EACH LAYER
C     IN THE FULLY CONNECTED NETWORK.
C     THESE SENSITIVITIES COMPUTATION IS DESIGNED ONLY FOR
C     LEVENBERG & LEV-MARQ, WHICH IS SLIGHTLY DIFFERENT THAN
C     BP. THE DIFFERENCE IS ONLY ON THE OUTPUT LAYER SENSITIVITIES
C     IMPLEMENTATION.
C
C     TASK  : COMPUTE S1 AND S2
C          S2=-F(N), WHERE F() IS THE DERIVATIVE OF THE ACTIVATION
C                FUNCTION OF LAYER 2, AND N IS THE WP+B.
C                WHEN K=NODE S2(K)=-F(N)
C                O.W. S2(K)=0
C          S1=F(N)(WM+1)S2.
C
C     FUNCTION CALL : NONE
```

```
C
      DOUBLE PRECISION A,W,S1,S2,NET,SUMS
      INTEGER STEP,NODE
      INTEGER NINP,NOUT,NHID
      DIMENSION A(1),W(1),S1(NHID),S2(NOUT)
C
C COMPUTE SENSITIVITIES OF EACH LAYER
C
C COMPUTE SENSITIVITIES OF LAYER2
C
      STEP=((NINP+1)*NHID)+1
      DO 300 K=1,NOUT
       NET=0.0D0
       DO 310 J=(NINP+1),(NINP+NHID)
        NET=NET+(W(STEP)*A(J))
        STEP=STEP+1
310    CONTINUE
C
C COMPUTE NETOUTPUT WP+B
C
       NET=NET+W((NHID*(NINP+1))+(NOUT*NHID)+K)
C
C SET THE S2 THAT CORRESPONDED TO THE NODE TO THE ACTUAL SINGLE
C NODE SENSITIVITY COMPUTATION
C AND SET THE REST OF THE SENSITIVITIES THAT NODE REFFERED TO 0
C
       IF (K.EQ.NODE) THEN
        S2(K)=-1.0D0*DF2(NET)
       ELSE
        S2(K)=0.D0
       ENDIF
300    CONTINUE
C
C COMPUTE SENSITIVITIES OF LAYER1
C COMPUTE OUTPUT OF LAYER1
C
      STEP=1
      DO 400 K=1,NHID
       NET=0.0D0
       DO 410 J=1,NINP
        NET=NET+(W(STEP)*A(J))
        STEP=STEP+1
410    CONTINUE
       SUMS=0.0D0
       SUMS=SUMS+((W( (NHID*(NINP+1))+((K*NOUT)-(NOUT-NODE))) )
     &   * S2(NODE))
C
C COMPUTE NETOUTPUT WP+B
C
       NET=NET+W((NHID*NINP)+K)
       S1(K)=DF1(NET)*SUMS
400    CONTINUE
      RETURN
      END
```

```
C
C LECG.CNTL        TEST CG ROUTINE OF D. LE USING OSB1
C
      SUBROUTINE LCG(EGOAL,X,A,MNHID,MNOUT,N,NDIMA,MNPAT,LMODE2)
      IMPLICIT REAL*8 (A-H,O-Z)
C
      INTEGER IER,J,LP,MASK,MAXFN,N,NCALLS,NPFAC,EPOCH,LMODE2
      INTEGER NINP,NHID,NOUT,NPAT,LP2,NW,MNHID,MNOUT,NDIMA,MNPAT
C
      DOUBLE PRECISION X,WORK,FBEST,FACNP,EPS,STEP,RLDFMX,YN,
     *  GRAD,GRADIF
      DOUBLE PRECISION A,EGOAL
C
      EXTERNAL FUN
C
      DIMENSION X(N),WORK(N*3+3)
      DIMENSION A(MNPAT,NDIMA)
      DIMENSION MASK(N),GRAD(N),GRADIF(N)
C
      COMMON /CLECG/ FBEST,FACNP,NCALLS,NPFAC,EPOCH
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
      MAXFN=90000
      EPS=1.0D-6
      STEP=0.04D0
C
      FBEST=1.0D30
      FACNP=1.5D0
      NCALLS=0
      EPOCH=0
      NPFAC=1
C
      DO 99 J=1,N
        MASK(J)=0
   99 CONTINUE
C
      CALL GRCHEK (A,X,MASK,GRAD,GRADIF,RLDFMX,N,NDIMA,MNPAT)
C
      CALL LMINN (EGOAL,A,FUN,N,X,YN,MAXFN,EPS,STEP,WORK,IER,
     *  MNHID,MNOUT,NDIMA,MNPAT,LMODE2)
C
      WRITE(LP,1)YN,IER,(X(J),J=1,N)
    1 FORMAT(/' YN =',1PG15.7,5X,'IER =',I3/
     *  ' X(*) =',5G15.7/(7X,5G15.7))
C
      WRITE(LP,2)WORK(1),WORK(2),WORK(3)
    2 FORMAT(/' WORK(1) =',1PG15.7,5X,'WORK(2) =',G15.7/
     *  5X,'WORK(3) =',G15.7)
C
      WRITE(LP2,3)YN,IER,(X(J),J=1,N)
    3 FORMAT(/' YN =',1PG15.7,5X,'IER =',I3/
     *  ' X(*) =',5G15.7/(7X,5G15.7))
C
C END MAIN (LECG)
```

```
C
      RETURN
      END
      SUBROUTINE FUN (A,N,X,FOBJ,GRAD,IOPT,WNF,WNG,
     * MNHID,MNOUT,NDIMA,MNPAT)
C
C IOPT = -1 TO RETURN ONLY FOBJ,
C IOPT =  0 TO RETURN BOTH FOBJ AND GRAD(*),
C IOPT = +1 TO RETURN ONLY GRAD(*)
C
      IMPLICIT REAL*8 (A-H,O-Z)
C
      INTEGER N,IOPT,NCALLS,NPFAC,J,K,KPL,NEXPS,NPTS,
     *  NPSAVE,EPOCH
C
      INTEGER NINP,NHID,NOUT,NPAT,LP2,NW,MNHID,MNOUT,NDIMA,MNPAT
C
      DOUBLE PRECISION A
      DOUBLE PRECISION DFIT,DSUM,DTEMP,EXPAB,FAC,FBEST,FIT,
     *   FOBJ,GRAD,RESNEG,RESNG2,TJ,WNF,WNG,X,YY1,ZEXP,
     *   FACNP,PRODUC
C
      DIMENSION X(1),GRAD(1)
C
      DIMENSION A(MNPAT,NDIMA)
C
      COMMON /CLECG/ FBEST,FACNP,NCALLS,NPFAC,EPOCH
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
      IF (IOPT.LE.0) THEN
       CALL FCN(X,A,FOBJ,MNOUT,NW,NDIMA,MNPAT)
       WNF=WNF+1.0D0
       IF (FOBJ.LT.FBEST) FBEST=FOBJ
      ENDIF
C
      IF(IOPT.GE.0) THEN
       CALL FGRAD(GRAD,X,A,MNOUT,NW,NDIMA,MNPAT)
       WNG=WNG+1.0D0
      ENDIF
C
C     PRINT THE FUNCTION VALUE OCCASIONALLY.
C
      NCALLS=NCALLS+1
      IF(NCALLS.GE.NPFAC) THEN
       PRINT 99,NCALLS,FBEST,EPOCH
   99  FORMAT(/' NCALLS =',I9,8X,'FBEST =',1PG20.12,'EPOCH =',I9,8X)
       NPSAVE=NPFAC
       NPFAC=NPFAC*FACNP
       IF(NPFAC.LE.NPSAVE) NPFAC=NPSAVE+1
      ENDIF
      RETURN
C
C END FUN (LECG)
C
```

```
C   END
C
    SUBROUTINE LMINN (EGOAL,A,FUN,N,XN,YN,MAXFN,EPS,STEP,WORK,IER,
   * MNHID,MNOUT,NDIMA,MNPAT,LMODE2)
C
C  UNCONSTRAINED MINIMIZATION BY CONJUGATE GRADIENTS
C
C  D. LE, "IMPLEMENTATION AND ASSESSMENT OF THE NEW NONLINEAR
C  PROGRAMMING CODE LMINN", UNSW REPORT 1982/OR/1, AUGUST 1982,
C  SCHOOL OF MECHANICAL AND INDUSTRIAL ENGINEERING,
C  UNIVERSITY OF NEW SOUTH WALES, KENSINGTON, NSW 2033, AUSTRALIA
C
C  D. LE, "A FAST AND ROBUST UNCONSTRAINED OPTIMIZATION METHOD
C  REQUIRING MINIMUM STORAGE", MATHEMATICAL PROGRAMMING 32 (1985) 41-68.
C
C  FUN  :  SUBROUTINE SUBPROGRAM SUPPLIED BY THE USER, WHICH MUST HAVE
C          THE FORM  SUBROUTINE FUN(N,X,Y,G,IOPT,WNF,WNG), WHERE
C             IOPT=-1 :  EVALUATE ONLY THE FUNCTION VALUE Y,
C                =+1 :  EVALUATE ONLY THE GRADIENT VECTOR G (G IS
C                       DIMENSIONED N),
C                = 0 :  EVALUATE BOTH Y AND G
C             WNF IS THE NUMBER OF FUNCTION EVALUATIONS AND SHOULD
C              BE SET  WNF=WNF+1  WHEN IOPT=-1 OR IOPT=0
C             WNG IS THE NUMBER OF GRADIENT EVALUATIONS AND SHOULD
C              BE SET  WNF=WNF+1  WHEN IOPT=1 OR IOPT=0
C
C  N    :  THE DIMENSION OF THE PROBLEM
C
C  XN   :  (DIMENSIONED N) ON ENTRY, THE STARTING POINT,
C                  ON EXIT, THE FINAL POSITION OF THE MINIMUM
C
C  YN   :  ON EXIT, THE FINAL MINIMUM FUNCTION VALUE
C
C  MAXFN:  THE MAXIMUM NUMBER OF FUNCTION EVALUATIONS TO BE DONE
C
C  EPS  :  TEST VALUE ON GRADIENT FOR CONVERGENCE CRITERION
C
C  STEP :  INITIAL STEPSIZE
C
C  WORK :  (DIMENSIONED 3*N+3) WORKING VECTOR.
C        ON EXIT, WORK(1), WORK(2), AND WORK(3) ARE THE NUMBER OF
C        ITERATIONS, NUMBER OF FUNCTION EVALUATIONS, AND NUMBER OF
C        GRADIENT EVALUATIONS, RESPECTIVELY.  WORK(4),...,WORK(N+3)
C        ARE RECENT GRADIENT COMPONENTS.
C
C  IER  :  RETURNED =0 FOR A NORMAL TERMINATION,
C       RETURNED =1 IF MAXFN FUNCTION EVALUATIONS WERE NOT ENOUGH
C
    IMPLICIT REAL*8 (A-H,O-Z)
C
    INTEGER N,IER,N1,N11,N2,N22,N3,N33,I,NF1,NFAIL,NQUAD1,
   *  NQUAD,MAXFN,IOPT,EPOCH,LMODE2
C
    INTEGER NINP,NHID,NOUT,NPAT,LP2,NW,MNHID,MNOUT,NDIMA,MNPAT
```

```
C
      DOUBLE PRECISION A,EGOAL
C
      DOUBLE PRECISION X,XN,YN,EPS,STEP,WORK,
     *  RZERO,UNITR,SIZE,SIZE1,Y3,STEPI,SIZE2,DUM,ULMINN,XSTEP
      DOUBLE PRECISION DOLDR,DNEWR
      INTEGER CLASS,IOPTST,LPX,LPY,LPZ,LCNT,LCNV
      REAL TIMES(2),TOTAL
C
      EXTERNAL FUN
C
      DIMENSION XN(N),WORK(1)
C
      DIMENSION A(MNPAT,NDIMA)
      COMMON /CLECG/ FBEST,FACNP,NCALLS,NPFAC,EPOCH
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
      RZERO=0.0D0
      UNITR=1.0D0
      DOLDR=9.9D9
      LCNT=0
      LCNV=3
C
      N1=3
      N11=N1+1
      N2=N1+N
      N22=N2+1
      N3=N2+N
      N33=N3+1
C
      IER=0
      WORK(1)=RZERO
      WORK(2)=RZERO
      WORK(3)=RZERO
C
C     CALL FUN (N,XN,YN,WORK(N11),0,WORK(2),WORK(3))
      CALL FUN (A,N,XN,YN,WORK(N11),0,WORK(2),WORK(3),
     *  MNHID,MNOUT,NDIMA,MNPAT)
C
      SIZE=ULMINN(A,DUM,WORK(N11),WORK(N33),WORK(N33),N,0,FUN,DUM,
     *  MNHID,MNOUT,NDIMA,MNPAT)
C
      DO 10 I=1,N
        WORK(N1+I)=-WORK(N1+I)/SIZE
   10 CONTINUE
C
      WORK(N33)=-SIZE
C
C  RESTART.
C
   20 DO 30 I=1,N
        WORK(N2+I)=WORK(N1+I)
   30 CONTINUE
```

```
C
      SIZE1=RZERO
      IOPT=2
      NF1=WORK(1)
      NFAIL=0
      NQUAD1=0
      GO TO 90
C
C  START OF ITERATION.  ADJUSTING STEP.
C
   40 NFAIL=0
      NQUAD1=0
   50 Y3=YN
      EPOCH=EPOCH+1
C
      IF(FBEST.LE.EGOAL) RETURN
      IF(LMODE2.EQ.1) THEN
       IF(NPAT.EQ.LCLSF(XN,A,NOUT,NW,NDIMA,MNPAT)) RETURN
      ENDIF
C
C  VALIDATION
C
      IF(IOPTST.GE.2) THEN
         DNEWR=VALID(XN,MNOUT,NW,NDIMA,MNPAT,LMODE2)
         IF ((DNEWR.GT.DOLDR).AND.(LCNT.GE.LCNV)) THEN
            RETURN
         ELSEIF ((DNEWR.GT.DOLDR).AND.(LCNT.LT.LCNV)) THEN
            LCNT=LCNT+1
         ELSE
            LCNT=0
         ENDIF
         DOLDR=DNEWR
      ENDIF
C
      CALL LMIN1 (A,FUN,STEP,DUM,Y3,WORK(2),XN,WORK(N11),WORK(N33),
     X    N,MAXFN,3,NQUAD,MNHID,MNOUT,NDIMA,MNPAT)
C
      DO 60 I=1,N
       WORK(N1+I)=XN(I)+STEP*WORK(N1+I)
   60 CONTINUE
C
C     CALL FUN (N,WORK(N11),Y3,WORK(N33),-1,WORK(2),WORK(3))
      CALL FUN (A,N,WORK(N11),Y3,WORK(N33),-1,WORK(2),WORK(3),
     *  MNHID,MNOUT,NDIMA,MNPAT)
      IF(WORK(2).GE.MAXFN) GO TO 120
C
C  SEARCH PARALLEL DIRECTION.
C
      STEPI=STEP
      CALL LMIN1 (A,FUN,STEPI,XSTEP,Y3,WORK(2),WORK(N11),WORK(N22),
     X   WORK(N33),N,MAXFN,4,NQUAD,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WORK(2).GE.MAXFN) GO TO 120
C
      NQUAD1=NQUAD1+NQUAD
```

```
C
      DO 70 I=1,N
        WORK(N1+I)=WORK(N1+I)+XSTEP*WORK(N2+I)
        WORK(N2+I)=WORK(N1+I)-XN(I)
   70 CONTINUE
C
      SIZE1=ULMINN(A,DUM,WORK(N22),WORK(N33),WORK(N33),N,0,FUN,DUM,
     *  MNHID,MNOUT,NDIMA,MNPAT)
      DO 80 I=1,N
        IF (SIZE1.EQ.0.D0) RETURN
        WORK(N2+I)=WORK(N2+I)/SIZE1
   80 CONTINUE
C
C  SEARCH CONJUGATE DIRECTION.
C
      WORK(N33)=SIZE1
      WORK(N33+1)=Y3
      IOPT=1
   90 STEPI=STEP
      Y3=YN
      CALL LMIN1 (A,FUN,STEPI,SIZE2,Y3,WORK(2),XN,WORK(N22),WORK(N33),
     X  N,MAXFN,IOPT,NQUAD,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WORK(2).GE.MAXFN) GO TO 120
C
      NQUAD1=NQUAD1+NQUAD
C
      DO 100 I=1,N
        XN(I)=XN(I)+SIZE2*WORK(N2+I)
  100 CONTINUE
C
      YN=Y3
      WORK(1)=WORK(1)+UNITR
C
C  GRADIENT AT XN
C
C     CALL FUN (N,XN,YN,WORK(N11),1,WORK(2),WORK(3))
      CALL FUN (A,N,XN,YN,WORK(N11),1,WORK(2),WORK(3),
     *  MNHID,MNOUT,NDIMA,MNPAT)
      SIZE=ULMINN(A,DUM,WORK(N11),WORK(N33),WORK(N33),N,0,FUN,DUM,
     *  MNHID,MNOUT,NDIMA,MNPAT)
C
C  TEST CONVERGENCE CRITERION.
C
      IF(SIZE.LE.EPS) RETURN
C
      DO 110 I=1,N
        WORK(N1+I)=-WORK(N1+I)/SIZE
  110 CONTINUE
      WORK(N33)=-SIZE
C
C  CHECK WHETHER OR NOT RESTART IS NEEDED.
C
      IF(SIZE2.GT.SIZE1) GO TO 40
C
```

```
      NFAIL=NFAIL+1
      IF(NFAIL.LT.2) GO TO 50
C
      IF(NQUAD1.LT.4 .OR.
     X  (NQUAD1.GE.4 .AND. WORK(1).GE.NF1+N+1)) GO TO 20
C
      GO TO 50
C
C  MAXIMUM NUMBER OF FUNCTION EVALUATIONS EXCEEDED.
C
  120 IER=1
      RETURN
C
C  END LMINN (CONJUGATE GRADIENT MINIMIZATION BY D. LE)
C
      END
      DOUBLE PRECISION FUNCTION ULMINN (A,XSTEP,X,D,T,N,IOPT,FUN,WNF,
     *          MNHID,MNOUT,NDIMA,MNPAT)
C
C  UTILITY FUNCTION SUBPROGRAM FOR SUBROUTINE LMINN BY D. LE.
C
      IMPLICIT REAL*8 (A-H,O-Z)
C
      INTEGER N,IOPT,I
C
      INTEGER NINP,NHID,NOUT,NPAT,LP,LP2,NW,MNHID,MNOUT,NDIMA,MNPAT
C
      DOUBLE PRECISION A
      DOUBLE PRECISION XSTEP,X,D,T,WNF,DSQRT,DUM
C
      DIMENSION X(N),D(N),T(N)
C
      DIMENSION A(MNPAT,NDIMA)
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
      IF(IOPT.EQ.0) GO TO 20
C
C  IOPT.LT.ZERO:  CALCULATE  T(*)=X(*)+XSTEP*D(*) AND THE VALUE OF F(T).
C
      DO 10 I=1,N
        T(I)=X(I)+XSTEP*D(I)
   10 CONTINUE
C
C     CALL FUN (N,T,ULMINN,D,-1,WNF,DUM)
      CALL FUN (A,N,T,ULMINN,D,-1,WNF,DUM,
     *  MNHID,MNOUT,NDIMA,MNPAT)
      RETURN
C
C  IOPT.EQ.ZERO:  CALCULATE THE MAGNITUDE OF A VECTOR X(*).
C
   20 ULMINN=0.0D0
      DO 30 I=1,N
        ULMINN=ULMINN+X(I)*X(I)
   30 CONTINUE
```

```
C
      ULMINN=DSQRT(ULMINN)
      RETURN
C
C  END ULMINN (CALLED BY LMINN)
C
      END
      SUBROUTINE LMIN1 (AA,FUN,STEPI,XMIN,YMIN,WNF,X,D,T,N,MAXFN,
     X   IOPT,NQUAD,MNHID,MNOUT,NDIMA,MNPAT)
C
C  LINE SEARCH SUBROUTINE FOR SUBROUTINE LMINN BY D. LE.
C
      IMPLICIT REAL*8 (A-H,O-Z)
C
      INTEGER N,MAXFN,IOPT,NQUAD,LP
      INTEGER NINP,NHID,NOUT,NPAT,LP2,NW,MNHID,MNOUT,NDIMA,MNPAT
C
      DOUBLE PRECISION AA
      DOUBLE PRECISION STEPI,XMIN,YMIN,WNF,X,D,T,
     *   ZABS,ZSIGN,ARG,ARG2,TOLA,TOLB,TOLC,RZERO,RHALF,
     *   UNITR,RTWO,RFOUR,X2,Y2,Y0,X1,Y1,X2TEMP,GXN,ULMINN,
     *   X2SAVE,XSTEMP,X2FAC,STEP,Z,A,B,C,DEL,XQMIN,
     *   YQMIN,YQQ,DABS,DMIN1,DSIGN,DSQRT
C
      EXTERNAL FUN
C
      DIMENSION X(N),D(N),T(N)
      DIMENSION AA(MNPAT,NDIMA)
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
C
      ZABS(ARG)=DABS(ARG)
      ZSIGN(ARG,ARG2)=DSIGN(ARG,ARG2)
C
      TOLA=1.0D-10
      TOLB=1.0D-4
      TOLC=1.0D-15
C
      RZERO=0.0D0
      RHALF=0.5D0
      UNITR=1.0D0
      RTWO=2.0D0
      RFOUR=4.0D0
C
      NQUAD=0
      IF(IOPT.NE.1) GO TO 30
C
C  IOPT=1:  LINE SEARCH FOR CONJUGATE DIRECTION (DETERMINE ALPHA(K)).
C
      XMIN=RZERO
      X2=T(1)
      Y2=T(2)
      Y0=DMIN1(YMIN,Y2)
      IF(YMIN.GT.Y2) GO TO 110
C
```

```
      X1=XMIN
      Y1=YMIN
      GO TO 20
C
   10 X2=XMIN
      Y2=YMIN
   20 XMIN=X2/RFOUR
C
      X2TEMP=UNITR+X2
      IF(X2TEMP.EQ.UNITR .OR. X2TEMP.EQ.X2) THEN
        YMIN=Y1
        WNF=WNF+UNITR
        GO TO 130
      ENDIF
C
      YMIN=ULMINN(AA,X2,X,D,T,N,-1,FUN,WNF,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
      IF(YMIN-Y1)130,130,10
C
   30 GXN=T(1)
      XMIN=RZERO
      Y0=YMIN
      X2=STEPI
      Y2=ULMINN(AA,X2,X,D,T,N,-1,FUN,WNF,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
C
      IF(IOPT-3)40,60,90
C
C  IOPT=2:  LINE SEARCH FOR RESTART (DETERMINE ALPHA(0)).
C
   40 IF(Y2.LT.YMIN) GO TO 100
C
C          50 X2=0.5*GXN*X2*X2/(YMIN-Y2+GXN*X2)
C
   50 CONTINUE
      X2SAVE=X2
      XSTEMP=UNITR+X2SAVE
      X2FAC=RHALF*GXN*X2/(YMIN-Y2+GXN*X2)
      X2=X2*X2FAC
C
      X2TEMP=UNITR+X2
      IF(X2TEMP.EQ.UNITR) THEN
        IF(ZABS(X2SAVE).GT.ZABS(X2)) X2=ZSIGN(X2SAVE,X2)
        Y2=Y0
        WNF=WNF+UNITR
        GO TO 55
      ENDIF
C
      Y2=ULMINN(AA,X2,X,D,T,N,-1,FUN,WNF,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
C
      IF(Y2.GT.Y0) GO TO 50
C
   55 XMIN=X2
```

```
      YMIN=Y2
      RETURN
C
C  IOPT=3:  ADJUST STEPSIZE (DETERMINE BETA(K)).
C
C
   60 STEP=STEPI*RFOUR
      STEPI=GXN*X2*X2/(YMIN-Y2+GXN*X2)
      IF(STEPI.GT.XMIN .AND. STEPI.LE.STEP) RETURN
   70 X1=XMIN
      Y1=YMIN
      XMIN=X2
      YMIN=Y2
      X2=STEP
      STEP=STEP*RFOUR
      Y2=ULMINN(AA,X2,X,D,T,N,-1,FUN,WNF,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
C
      Z=(Y1-YMIN)/(X1-XMIN)
      A=(Z-(Y2-YMIN)/(X2-XMIN))/(X1-X2)
      IF(A.LE.RZERO) GO TO 70
C
      B=Z-A*(X1+XMIN)
      C=Y1-X1*(B+X1*A)-Y0
      DEL=DSQRT(B*B-RFOUR*A*C)
      STEPI=(-B+DEL)/(RTWO*A)
      IF(STEPI.GT.XMIN) GO TO 80
C
      STEPI=(-B-DEL)/(RTWO*A)
   80 IF(STEPI.LE.STEP) RETURN
C
      GO TO 70
C
C  IOPT=4:  LINE SEARCH FOR PARALLEL DIRECTION (DETERMINE GAMMA(K)).
C
   90 IF(YMIN.GE.Y2) GO TO 100
C
      STEPI=-STEPI
      X1=X2
      Y1=Y2
      X2=XMIN+STEPI
      GO TO 120
C
C  GENERAL SITUATION WHEN THREE POINTS ARE KNOWN...
C
  100 STEPI=STEPI*RTWO
  110 X1=XMIN
      Y1=YMIN
      XMIN=X2
      YMIN=Y2
      X2=X2+STEPI
  120 Y2=ULMINN(AA,X2,X,D,T,N,-1,FUN,WNF,MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
C
```

```
      IF(YMIN.LE.Y2) GO TO 130
C
C  QUADRATIC FIT TO X1,XMIN,X2 WITH YMIN.LT.Y1 AND YMIN.GT.Y2 ...
C
      Z=(Y1-YMIN)/(X1-XMIN)
      A=(Z-(Y2-YMIN)/(X2-XMIN))/(X1-X2)
      IF(A.LE.RZERO) GO TO 100
C
      B=Z-A*(X1+XMIN)
      C=Y1-X1*(B+X1*A)
      XQMIN=-B/(RTWO*A)
      IF((XQMIN-X2-RTWO*STEPI)*(XQMIN-X2).GE.RZERO) GO TO 100
C
      YQMIN=ULMINN(AA,XQMIN,X,D,T,N,-1,FUN,WNF,
     *  MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
C
      IF(YQMIN.LE.Y2) GO TO 150
C
      X1=XMIN
      Y1=YMIN
      XMIN=X2
      YMIN=Y2
      X2=XQMIN
      Y2=YQMIN
C
C  QUADRATIC FIT TO X1,XMIN,X2 WITH YMIN.LT.Y1 AND YMIN.LT.Y2 ...
C
  130 Z=(Y1-YMIN)/(X1-XMIN)
      A=(Z-(Y2-YMIN)/(X2-XMIN))/(X1-X2)
      B=Z-A*(X1+XMIN)
      C=Y1-X1*(B+X1*A)
C
      IF(A.EQ.0.0D0) THEN
        XQMIN=XMIN
      ELSE
        XQMIN=-B/(RTWO*A)
      ENDIF
C
      IF(ZABS(XQMIN-XMIN).LE.TOLA*(ZABS(XMIN)+UNITR)) RETURN
C
      YQMIN=ULMINN(AA,XQMIN,X,D,T,N,-1,FUN,WNF,
     *  MNHID,MNOUT,NDIMA,MNPAT)
      IF(WNF.GE.MAXFN) RETURN
C
      IF(YQMIN.LE.YMIN) GO TO 150
C
      IF(YMIN.LT.Y0) RETURN
C
      IF((X1-XMIN)*(XQMIN-XMIN).LT.RZERO) GO TO 140
C
      X1=XQMIN
      Y1=YQMIN
      GO TO 130
```

```
C
 140 X2=XQMIN
     Y2=YQMIN
     GO TO 130
C
 150 XMIN=XQMIN
     YMIN=YQMIN
     YQQ=(A*XMIN+B)*XMIN+C
     IF(ZABS(YQQ-YMIN).LE.TOLB*ZABS(YMIN)+TOLC) NQUAD=1
     RETURN
C
C  END LMIN1 (CALLED BY LMINN)
C
     END
     SUBROUTINE SCG(EGOAL,X,A,MNOUT,MNW,NDIMA,MNPAT,LMODE2)
C
C  SCG 1.0      A.N.S.I. STANDARD FORTRAN 77      APRIL 1996
C
C  UNCONSTRAINED MINIMIZATION BY A SCALED CONJUGATE GRADIENT ALGORITHM
C
C  HENDRA TIO AND J. P. CHANDLER,
C     COMPUTER SCIENCE DEPARTMENT,
C     OKLAHOMA STATE UNIVERSITY
C
C  "A SCALED CONJUGATE GRADIENT ALGORITHM FOR
C     FAST SUPERVISED LEARNING" BY MARTIN FODSLETTE MOLLER,
C     NEURAL NETWORKS 6 (1993) 525-533
C
C  INPUT QUANTITIES....  N,X(*),RELTOL,ABSTOL,NRESTR,MAXIT,NTRACE,LP
C
C  OUTPUT QUANTITIES...  X(*)
C
C     N      -- THE NUMBER OF PARAMETERS, X(J)
C
C     X(*)   -- THE VECTOR OF PARAMETERS.
C               ON ENTRY, X(*) SHOULD CONTAIN THE BEST
C               AVAILABLE STARTING POINT FOR THE MINIMIZATION.
C               ON EXIT, X(*) WILL CONTAIN THE BEST POINT
C               THAT SUBROUTINE SCG COULD FIND.
C
C     RELTOL -- RELATIVE CONVERGENCE TOLERANCE
C
C     ABSTOL -- ABSOLUTE CONVERGENCE TOLERANCE
C
C     NCONVG -- NUMBER OF SUCCESSIVE ITERATIONS THAT MUST PASS
C               THE CONVERGENCE TEST
C
C     NRESTR -- THE CONJUGATE GRADIENT ITERATION WILL BE
C               RESTARTED AFTER EVERY NRESTR ITERATIONS
C
C     MAXIT  -- UPPER LIMIT ON THE NUMBER OF ITERATIONS
C               TO BE PERFORMED
C
C     NTRACE -- =+1 TO PRINT AT THE END OF EACH CYCLE,
```

```
C     =+2 TO PRINT AT EACH ITERATION,
C     =+3 TO PRINT AT EACH STEP OF EACH ITERATION,
C     = 0 TO PRINT ONLY INITIAL AND FINAL QUANTITIES,
C     =-1 NOT TO PRINT AT ALL
C   LP  -- THE LOGICAL UNIT NUMBER FOR OUTPUT
C
C
      IMPLICIT REAL*8(A-H,O-Z)
C
C
      DOUBLE PRECISION RELTOL,ABSTOL,EGOAL
      DOUBLE PRECISION X,R,P,GRADX,GRSIGP,S,RX
      DOUBLE PRECISION DELTAK,SIGMA,RLAMBD,BARLAM,PLENG,
     *  SIGMAK,DELSAV,BARSAV,RLAMSV,RMU,ALPHA,CAPDEL,
     *  RDOT,RLENSQ,BETA,RZERO,UNITR,XPLUS,CRITER,
     *  FX,FRX,FBEST,
     *  DABS,DSQRT,A,ERR
      DOUBLE PRECISION DOLDR,DNEWR
      INTEGER CLASS,IOPTST,LPX,LPY,LPZ
      REAL TIMES(2),TOTAL
      INTEGER NINP,NOUT,NHID,NPAT,NW,LP,MNOUT,MNW,NDIMA,MNPAT
      INTEGER MAXIT,NCONVG,NTRACE,NFEVAL,NGEVAL,LCNT
      INTEGER N,NRESTR, ITER,L,JSUCCS,JCONVG,LP2
     *  KNVGOK,LMODE2,LCNV
     *  MOD
C
      DIMENSION X(MNW),R(MNW),P(MNW),A(MNPAT,NDIMA),ERR(MNOUT),
     *  GRADX(MNW),GRSIGP(MNW),S(MNW),RX(MNW)
C
      COMMON /DRIVE/ NINP,NHID,NOUT,NPAT,NW,LP,LP2
      COMMON /DCLASS/ CLASS,TIMES,TOTAL,IOPTST,LPX,LPY,LPZ
C
      MAXIT=8000
      NTRACE=1
      NCONVG=3
      ABSTOL=0.D0
      RELTOL=1.D-6
      LCNT=0
      LCNV=3
      N=((NINP+1)*NHID) + ((NHID+1)*NOUT)
C     NRESTR=1
      NRESTR=N+1
      RZERO=0.0D0
      UNITR=1.0D0
      DOLDR=9.9D9
C
      JCONVG=0
C
C NRESTR = NUMBER OF ITERATIONS BETWEEN RESTARTS.
C
      IF(NRESTR.LE.0) NRESTR=N+3
C
      ITER=1
```

```
      DELTAK=RZERO
C
C  STEP 1 ... INITIALIZE.
C
C  CHOOSE SCALARS SIGMA IN (0,1.0D-4) AND RLAMBD IN (0,1.0D-6)
C  AND BARLAM=0.
C
      SIGMA=1.0D-4
      RLAMBD=1.0D-6
      BARLAM=RZERO
C
      CALL FCN(X,A,FX,NOUT,NW,NDIMA,MNPAT)
      NFEVAL=1
C
      CALL FGRAD(GRADX,X,A,NOUT,NW,NDIMA,MNPAT)
      NGEVAL=1
C
      DO 10 L=1,N
        P(L)=-GRADX(L)
        R(L)=P(L)
   10 CONTINUE
C
      JSUCCS=1
C
C  STEP 2 ...
C
C  IF SUCCESS=TRUE, THEN CALCULATE SECOND ORDER INFORMATION.
C
   20 PLENG=RZERO
      DO 30 L=1,N
        PLENG=PLENG+P(L)**2
   30 CONTINUE
      PLENG=DSQRT(PLENG)
C
      DO 35 L=1,N
        P(L)=P(L)/PLENG
   35 CONTINUE
C
      PLENG=UNITR
C
      IF(JSUCCS.EQ.1) THEN
        SIGMAK=SIGMA/PLENG
C
        KNVGOK=1
        DO 40 L=1,N
          RX(L)=X(L)+SIGMAK*P(L)
          CRITER=RELTOL*DABS(X(L))+ABSTOL
          IF(DABS(RX(L)-X(L)).GT.CRITER) KNVGOK=0
   40   CONTINUE
C
        CALL FGRAD(GRSIGP,RX,A,NOUT,NW,NDIMA,MNPAT)
        NGEVAL=NGEVAL+1
C
        DO 50 L=1,N
```

```
      S(L)=(GRSIGP(L)-GRADX(L))/SIGMAK
   50 CONTINUE
C
   * DELTAK=RZERO
   * DO 60 L=1,N
      DELTAK=DELTAK+P(L)*S(L)
   60 CONTINUE
C
   IF(NTRACE.GE.3) WRITE(LP,70)ITER,SIGMA,PLENG,SIGMAK,DELTAK
   70 FORMAT(/' STEP 2 ... ITER =',I6,5X,'SIGMA =',1PG13.5,5X,
   *  'PLENG =',G13.5/
   *  5X,'SIGMAK =',G13.5,5X,'DELTAK =',G13.5)
C
   ENDIF
C
C STEP 3 ... SCALE DELTA(K).
C
   DELSAV=DELTAK
   DELTAK=DELTAK+(RLAMBD-BARLAM)*PLENG**2
C
   IF(NTRACE.GE.3) WRITE(LP,80)ITER,DELSAV,RLAMBD,BARLAM,PLENG,
   *  DELTAK
   80 FORMAT(/' STEP 3 ... ITER =',I6,5X,'DELSAV =',1PG13.5,5X,
   *  'RLAMBD =',G13.5/
   *  5X,'BARLAM =',G13.5,5X,'PLENG =',G13.5,5X,'DELTAK =',G13.5)
C
C STEP 4 ... IF DELTAK .LE. ZERO THEN MAKE THE
C            HESSIAN MATRIX POSITIVE DEFINITE.
C
   IF(DELTAK.LE.0) THEN
     DELSAV=DELTAK
     BARSAV=BARLAM
     RLAMSV=RLAMBD
     BARLAM=2.0D0*(RLAMBD-DELTAK/PLENG**2)
     DELTAK=-DELTAK+RLAMBD*PLENG**2
     RLAMBD=BARLAM
C
     IF(NTRACE.GE.3) WRITE(LP,90)ITER,DELSAV,BARSAV,RLAMSV,PLENG,
   *   BARLAM,DELTAK
   90  FORMAT(/' STEP 4 ... ITER =',I6,5X,'DELSAV =',1PG13.5,5X,
   *   'BARSAV =',G13.5/
   *   5X,'RLAMSV =',G13.5,5X,'PLENG =',G13.5/
   *   5X,'PLENG =',G13.5,5X,'BARLAM =',G13.5/
   *   5X,'DELTAK =',G13.5)
C
   ENDIF
C
C STEP 5 ... CALCULATE THE STEP SIZE.
C
   RMU=RZERO
   DO 100 L=1,N
     RMU=RMU+P(L)*R(L)
  100 CONTINUE
   ALPHA=RMU/DELTAK
```

```
C
      IF(NTRACE.GE.3) WRITE(LP,110)ITER,RMU,DELTAK,ALPHA
  110 FORMAT(/' STEP 5 ...  ITER =',I6,5X,'RMU =',1PG13.5,5X,
     *  'DELTAK =',G13.5/
     *  5X,'ALPHA =',G13.5)
C
C  STEP 6 ...  CALCULATE THE COMPARISON PARAMETER, CAPDEL.
C
      DO 120 L=1,N
        RX(L)=X(L)+ALPHA*P(L)
  120 CONTINUE
C
      CALL FCN(RX,A,FRX,NOUT,NW,NDIMA,MNPAT)
      NFEVAL=NFEVAL+1
C
      CAPDEL=2.0D0*DELTAK*(FX-FRX)/RMU**2
C
      IF(NTRACE.GE.3) WRITE(LP,130)ITER,DELTAK,FX,FRX,RMU,CAPDEL
  130 FORMAT(/' STEP 6 ...  ITER =',I6,5X,'DELTAK =',1PG13.5/
     *  5X,'FX =',G13.5,5X,'FRX =',G13.5/
     *  5X,'RMU =',G13.5,5X,'CAPDEL =',G13.5)
C
C  STEP 7 ...  IF THE COMPARISON PARAMETER CAPDEL IS .GE. ZERO
C              THEN A SUCCESSFUL REDUCTION IN THE FUNCTION
C              CAN BE MADE.
C
      IF(CAPDEL.GE.RZERO) THEN
C
        CALL FGRAD(GRADX,RX,A,NOUT,NW,NDIMA,MNPAT)
        NGEVAL=NGEVAL+1
C
        RDOT=RZERO
        RLENSQ=RZERO
        DO 150 L=1,N
          X(L)=RX(L)
          RDOT=RDOT-GRADX(L)*R(L)
          R(L)=-GRADX(L)
          RLENSQ=RLENSQ+R(L)**2
  150   CONTINUE
C
        FX=FRX
C
        BARLAM=RZERO
        JSUCCS=1
C
        IF(NTRACE.GE.3) WRITE(LP,170)ITER,CAPDEL
  170   FORMAT(/' STEP 7 ...  ITER =',I6,5X,'CAPDEL =',1PG13.5,
     *    5X,'F DECREASED.')
C
        IF(MOD(ITER,NRESTR).EQ.0) THEN
C
          DO 180 L=1,N
            P(L)=R(L)
  180       CONTINUE
```

```
C  *
  250   IF(NTRACE.GE.1) WRITE(LP,190)ITER,NRESTR
 190     FORMAT(/' STEP 7 ... ITER =',I9,5X,'NRESTR =',I7/
   *      5X,'RESTART WITH P(*)=R(*).')
C
      ELSE
C
        BETA=(RLENSQ-RDOT)/RMU
        DO 210 L=1,N
          P(L)=R(L)+BETA*P(L)
 210      CONTINUE
C
        IF(NTRACE.GE.3) WRITE(LP,220)ITER,RLENSQ,RDOT,RMU,BETA
 220      FORMAT(/' STEP 7 ... ITER =',I6,
   *       5X,'RLENSQ =',1PG13.5,5X,'RDOT =',G13.5/
   *       5X,'RMU =',G13.5,5X,'BETA =',G13.5)
C
      ENDIF
C
      IF (CAPDEL.GE.0.75D0) THEN
        RLAMSV=RLAMBD
C
C  DO NOT ALLOW RLAMBD TO DECREASE MUCH BELOW MACHINE EPSILON.
C
        XPLUS=UNITR+RLAMBD
        IF(XPLUS.GT.UNITR) RLAMBD=0.25D0*RLAMBD
C
        IF(NTRACE.GE.3) WRITE(LP,230)ITER,CAPDEL,RLAMSV,RLAMBD
 230      FORMAT(/' STEP 7 ... ITER =',I6,5X,'CAPDEL =',1PG13.5/
   *       5X,'RLAMSV =',G13.5,5X,'RLAMBD =',G13.5)
C
      ENDIF
C
    ELSE
      BARSAV=BARLAM
      BARLAM=RLAMBD
      JSUCCS=0
C
      IF(NTRACE.GE.3) WRITE(LP,240)ITER,BARSAV,BARLAM
 240    FORMAT(/' STEP 7 ... ITER =',I6/
   *     5X,'BARSAV =',1PG13.5,5X,'BARLAM =',G13.5/
   *     5X,'NO REDUCTION CAN BE MADE IN THE FUNCTION VALUE.'/
   *     5X,'SET SUCCESS=FALSE.')
C
    ENDIF
C
C  STEP 8 ... IF THE COMPARISON PARAMETER CAPDEL IS .LT. 0.25,
C            THEN INCREASE THE SCALE PARAMETER DELTAK.
C
    IF(CAPDEL.LT.0.25D0) THEN
      RLAMSV=RLAMBD
      RLAMBD=RLAMBD+DELTAK*(UNITR-CAPDEL)/PLENG**2
C
      IF(NTRACE.GE.3) WRITE(LP,250)ITER,CAPDEL,RLAMSV,DELTAK,
```

```
     *    PLENG,RLAMBD
  250  FORMAT(/' STEP 8 ...  ITER =',I6,5X,'CAPDEL =',1PG13.5/
     *    5X,'RLAMSV =',G13.5,5X,'DELTAK =',G13.5/
     *    5X,'PLENG =',G13.5,5X,'RLAMBD =',G13.5)
C
     ENDIF
C
C  STEP 9 ...  IF THE ITERATION HAS CONVERGED,
C           SET K=K+1 AND GO TO STEP 2,
C           ELSE TERMINATE AND RETURN X(*) AS
C           THE DESIRED MINIMUM POINT.
C
C  THE CONVERGENCE CRITERION OF MOLLER WAS TO TEST FOR
C  THE GRADIENT BEING EXACTLY ZERO.
C  THIS IS IMPRACTICAL, AND HAS BEEN REPLACED.
C
     IF(KNVGOK.EQ.1) THEN
       JCONVG=JCONVG+1
     ELSE
       JCONVG=0
     ENDIF
C
     FBEST=FX
     IF(FRX.LT.FBEST) FBEST=FRX
C
     IF(NTRACE.GE.2 .OR. (NTRACE.GE.1 .AND. MOD(ITER,N).EQ.0))
     *  WRITE(LP,261) ITER,RLAMBD,DELTAK,BARLAM,ALPHA,
     *    FBEST,CAPDEL,JCONVG
  261  FORMAT(/' ITER =',I6,5X,'RLAMBD =',1PG13.5,5X,
     *    'DELTAK =',G13.5/
     *    5X,'BARLAM =',G13.5,5X,'ALPHA =',G13.5/
     *    5X,'FBEST =',G13.5,5X,'CAPDEL =',G13.5,5X,'JCONVG =',I2)
C
     IF(JCONVG.GE.NCONVG) GO TO 290
C
  270 ITER=ITER+1
C
     IF(FBEST.LE.EGOAL)  GOTO 290
C
     IF(LMODE2.EQ.1) THEN
      IF(NPAT.EQ.LCLSF(X,A,NOUT,NW,NDIMA,MNPAT)) GOTO 290
     ENDIF
     IF(ITER.GT.MAXIT) THEN
C
       IF(NTRACE.GE.0) WRITE(LP,280)MAXIT
  280   FORMAT(/' MORE THAN MAXIT =',I9,
     *    ' ITERATIONS IN SCG.  CONVERGENCE FAILURE.')
C
       GO TO 290
     ENDIF
C
C  VALIDATION
C
     IF(IOPTST.GE.2) THEN
```

```
      DNEWR=VALID(X,MNOUT,NW,NDIMA,MNPAT,LMODE2)
      IF ((DNEWR.GT.DOLDR).AND.(LCNT.GE.LCNV)) THEN
         GOTO 290
      ELSEIF ((DNEWR.GT.DOLDR).AND.(LCNT.LT.LCNV)) THEN
         LCNT=LCNT+1
      ELSE
         LCNT=0
      ENDIF
      DOLDR=DNEWR
    ENDIF
C
    GO TO 20
C
 290 CONTINUE
C
    RETURN
C
C  END SCG
C
    END
```

VITA

Hendra Tio

Candidate for the Degree of

Master of Science

Thesis: OPTIMIZATION METHODS FOR TRAINING FEEDFORWARD NEURAL NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Jakarta, Indonesia, June 25th 1969, the son of Mr. Thjung Sau Thin and Mrs. Tio Sui Kim.

Education: Graduated from Sekolah Menengah Atas 2, Jakarta, Indonesia in May 1988; received the Bachelor of Science from Oklahoma State University, Stillwater, July, 1994; completed the requirements for the degree of Master of Science at Oklahoma State University in December, 1996.