CHAPTER 3

DESIGN OF MODIFIED LEVENBERG MARQUARDT ALGORITHM

3.1 Introduction

Disk technology has improved steadily in multiple ways but it is difficult to relate the various physical improvements to the actual performance experienced by real workloads. Many optimization techniques have been invented to mask the slow mechanical nature of storage devices, most importantly disks. Because of the slow mechanical nature of many storage devices, the importance of optimizing I/O operations has been well recognized. Disk-Seek Algorithm is an important optimization technique to optimize the I/O operations by reducing the disk access time. The time required to satisfy a request depends on the state of the disk, specifically whether the requested data is present in the cache and where the disk head is relative to the requested data. First Come First Served (FCFS), Shortest Seek Time First (SSTF), and the SCAN algorithms were the most popular disk-seek algorithms. The operating system uses any one of the disk seek algorithm for a particular disk. Since the same algorithm is being used for all the disks present in the large disk storage system, the performance is poor. Hence the selection of the appropriate disk seek algorithm for each disk separately can be done by using the neural network.

3.2 Back Propagation Network (BPN)

A back propagation neural network uses a feed-forward topology, supervised learning, and back propagation learning algorithm. This algorithm was responsible in large part for the re-emergence of neural networks in the mid 1980s.
Back propagation is a general purpose learning algorithm. It is powerful but also expensive in terms of computational requirements for training. A back propagation network with a single hidden layer of processing elements can model any continuous function to any degree of accuracy (given enough processing elements in the hidden layer).

The basic back propagation algorithm consists of three steps:

- The input pattern is presented to the input layer of the network. These inputs are propagated through the network until they reach the output units. This forward pass produces the actual or predicted output pattern.
- Because back propagation is a supervised learning algorithm, the desired outputs are given as part of the training vector. The actual network outputs are subtracted from the desired outputs and an error signal is produced.
- This error signal is then the basis for the back propagation step, whereby the errors are passed back through the neural network by computing the contribution of each hidden processing unit and deriving the corresponding adjustment needed to produce the correct output. The connection weights are then adjusted and the neural network has just “learned” from an experience.

Two major learning parameters are used to control the training process of a back propagation network. The learning rate is used to specify whether the neural network is going to make major adjustments after each learning trial or if it is only going to make minor adjustments. Momentum is used to control possible oscillations in the weights, which could be caused by alternately signed error signals.

3.3 Neural Networks Learning Methods

Neural Networks training algorithms can be classified as deterministic or stochastic, according to the optimization algorithm used to minimize the cost function. The former type, which includes the various gradient-based algorithms,
such as Back propagation (BP) and Conjugate Gradient, is characterized by the use of deterministic search operators. In general, these algorithms determine, from a starting point in the search space, a direction which minimizes the error in that given point and steps toward this direction. The target of training algorithms is to minimize an error function by adjusting the corresponding weight matrix in the neural network. The Levenberg-Marquardt algorithm is popular to train the neural network because they are very suitable to minimize the error functions that arise from a squared-error criterion.

3.4 Levenberg Marquardt Algorithm

The rate at which the Gauss-Newton algorithm converges is not controlled. The second order approximate to the cost function does not always correspond well with the actual shape and taking a step which is too large can lead to worse results instead of improvement. Moreover, when at a point close to the minimum there is a danger of overshooting the minimal cost. The step $\delta \mathbf{p}$ taken at each iteration must have a correct length in order to avoid these problems. A method that is able to control the step size adaptively and effectively is the Levenberg-Marquardt (LM) method and it is this ability that made it very popular in solving least square problems. It is built upon the standard Gauss-Newton method of iterative solving of normal equations, however, it introduces a new variable called the damping term $\mu$, which is incorporated in the normal equation to create the augmented normal equation

$$(J^TJ+\mu I)\delta \mathbf{p}= J^T \mathbf{r} \quad (3.1)$$

Where $J$ is the Jacobian matrix, $I$ is the identity matrix, $\mu$ is the learning rate. The above equation allows to control the solution in terms of step-size and direction. When damping term is very high, then $(J^TJ+\mu I)$ is nearly diagonal, making $\delta \mathbf{p}$ approximately equal to $J^T \mathbf{r}$, which is the direction of gradient descent. Furthermore, the larger is the value of $\mu$, the smaller steps $\delta \mathbf{p}$ are taken, thus controlling the rate of convergence.
The strategy of the LM algorithm is as follows: at each step, if \( \delta p \) leads to reduction in the cost function, then the step is accepted and the damping term is decreased. If, however, the step leads to increase in the error, damping is increased and the augmented normal equations are solved again, until a suitable step, that decreases the cost function. This inner loop of finding acceptable deltas of parameters is considered as one iteration.

In practice, the LM algorithm continuously decreases the damping term as long as the updates are accepted, meaning that the step size is increased if the steps are being taken in the correct direction. This behavior is appreciated, since it accelerates the convergence. When approaching the minimum, step size may exceed the desired distance to the stationary point and the update leads to increase in the cost. In that case, iteratively reducing the step size is the way to keep the algorithm descending towards the minimum. If the cost function at current estimate is ill-approximated, then by increasing the damping term it is ensured that steps are taken in the direction of the gradient descent direction. LM thus alternates between the Gauss-Newton method (small damping terms) and gradient descent (large damping terms), taking the advantages of both. Numerous conditions exist for terminating the LM algorithm, ensuring that iterations stop when convergence occurs.

The performance index \( F(w) \) of the BPN network is defined as the sum of squared errors between the target outputs and the network’s simulated outputs

\[
F(w) = e^T e \quad (3.2)
\]

Where \( w = [w_1, w_2, \ldots, w_N] \) consists of all weights of the network, \( e \) is the error vector comprising the error for all the training examples.

When training with the LM method, the increment of weights \( \Delta w \) can be obtained as follows:

\[
\Delta w = [J^T J + \mu I]^{-1} J^T e \quad (3.3)
\]

Where \( J \) is the Jacobian matrix, \( I \) is the identity matrix, \( \mu \) is the learning rate which is to be updated using \( \beta \) depending on the outcome. In particular, \( \mu \) is multiplied by
decay rate $\beta$ ($0<\beta<1$) whenever $F(w)$ decreases, whereas $\mu$ is divided by $\beta$ whenever $F(w)$ increases in a new step.

The LM training process can be illustrated in the following steps

Step 1: Initialize the weights and parameter $\mu$ ($\mu=0.1$ is appropriate).
Step 2: Compute the sum of the squared errors of overall inputs $F(w)$
Step 3: Solve (2) to obtain the increment of weights $\Delta w$
Step 4: Recompute the sum of squared errors $F(w)$

Using $w + \Delta w$ as the trial $w$, and judge
If trial $F(w) < F(w)$ in step 2 then
$$w = w + \Delta w; \ \mu = \mu \cdot \beta \ (\beta = .1)$$
Go back to step 2
else
$$\mu = \mu / \beta$$
Go back to step 4
endif

3.5 Modified Levenberg Marquardt Algorithm (MLM)

A modification is made on Learning parameter resulted in decrement of both learning iteration and oscillation together. A modification method by varying the learning parameter has been made to speed up LM algorithm. In addition, the error oscillation has been decreased.

Considering performance index is $F(w) = e^T e$, using the Newton method we have,

$$W_{k+1} = W_k - A^{-1}_k \cdot g_k$$  \hspace{1cm} (3.4)

$$A_k = \nabla^2 F(w) \bigg|_{W=W_k}$$

$$g_k = \nabla F(w) \bigg|_{W=W_k}$$

$$[\nabla F(w)]_{jj} = \frac{\partial F(w)}{\partial w_j}$$
\[ = 2 \sum_{i=1}^{N} c_i(w) \frac{\partial G_i(w)}{\partial w_j} \]  \hspace{1cm} (3.5)

The gradient can be written as:
\[ \nabla F(x) = 2j^T c(w) \]  \hspace{1cm} (3.6)

where
\[
\frac{\partial c_{i1}}{\partial w_1} \quad \frac{\partial c_{i1}}{\partial w_2} \quad \cdots \quad \frac{\partial c_{i1}}{\partial w_N} \\
\frac{\partial c_{i2}}{\partial w_1} \quad \frac{\partial c_{i2}}{\partial w_2} \quad \cdots \quad \frac{\partial c_{i2}}{\partial w_N} \\
\vdots \\
\frac{\partial c_{ik}}{\partial w_1} \quad \frac{\partial c_{ik}}{\partial w_2} \quad \cdots \quad \frac{\partial c_{ik}}{\partial w_N} \\
\frac{\partial c_{k1}}{\partial w_1} \quad \frac{\partial c_{k1}}{\partial w_2} \quad \cdots \quad \frac{\partial c_{k1}}{\partial w_N}
\]

\[ j(w) = \begin{pmatrix}
\frac{\partial c_{i1}}{\partial w_1} & \frac{\partial c_{i1}}{\partial w_2} & \cdots & \frac{\partial c_{i1}}{\partial w_N} \\
\frac{\partial c_{i2}}{\partial w_1} & \frac{\partial c_{i2}}{\partial w_2} & \cdots & \frac{\partial c_{i2}}{\partial w_N} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial c_{ik}}{\partial w_1} & \frac{\partial c_{ik}}{\partial w_2} & \cdots & \frac{\partial c_{ik}}{\partial w_N} \\
\frac{\partial c_{k1}}{\partial w_1} & \frac{\partial c_{k1}}{\partial w_2} & \cdots & \frac{\partial c_{k1}}{\partial w_N}
\end{pmatrix}
\]

\( J(w) \) is called the Jacobian matrix. Next Hessian matrix is to be determined. The \( k, j \) elements of the Hessian matrix yields as:
\[ [\nabla^2 F(w)]_{kj} = \frac{\partial F(w)}{\partial w_k \partial w_j} \]

\[ [\nabla^2 F(w)]_{kj} = 2 \sum_{i=1}^{N} \left( \frac{\partial c_i(w)}{\partial w_k} \frac{\partial c_i(w)}{\partial w_j} + c_i(w) \frac{\partial^2 G_i(w)}{\partial w_k \partial w_j} \right) \]  \hspace{1cm} (3.7)

The Hessian matrix can then be expressed as follows:
\[ \nabla^2 F(w) = 2j^T J(w) \cdot j(w) + S(W) \]  \hspace{1cm} (3.8)

Where
\[ S(W) = \sum_{i=1}^{N} c_i(w) \cdot \nabla^2 G_i(w) \]
If $S(w)$ is small, Hessian matrix is approximated as
\[
\nabla^2 F (w) = 2 \mathbf{J}^\top (w) \cdot J (w)
\]
By using Gauss-Newton method the following equation obtained as
\[
W_{k+1} = W_k - \left[ 2 \mathbf{J}^\top (w_k) \cdot J (w_k) \right]^{-1} 2 \mathbf{J}^\top (w_k) e(w_k)
\]
\[
\approx W_k - \left[ J^\top (w_k) \cdot J (w_k) \right]^{-1} J^\top (w_k) e(w_k)
\]  \hspace{1cm} (3.9)

The advantage of Gauss-Newton is that it does not require calculation of second derivatives.

The problem of Gauss-Newton method is that the matrix $H = J^\top J$ may or may not be invertible. This can be overcome by using the following modification. Hessian matrix can be written as
\[
G = H + \mu I
\]  \hspace{1cm} (3.10)

Suppose that the eigen values and eigenvectors of $H$ are $\{\lambda_1, \lambda_2, ..., \lambda_n\}$ and $\{z_1, z_2, ..., z_n\}$. Then
\[
Gz_i = \left[ (H + \mu I) \right]z_i
\]
\[
= H z_i + \mu z_i
\]
\[
= \lambda_i z_i + \mu z_i
\]
\[
= (\lambda_i + \mu) z_i
\]

Therefore, the eigenvectors of $G$ are the same as the eigenvectors of $H$, and the eigenvalues of $G$ are $(\lambda_i + \mu)$. The matrix $G$ is positive definite by increasing $\mu$ until $(\lambda_i + \mu) > 0$ for all $i$. Therefore, the matrix will be invertible. This leads to Levenberg-Marquardt algorithm
\[
W_{k+1} = W_k - \left[ J^\top (w_k) J (w_k) + \mu I \right]^{-1} J^\top (w_k) e(w_k)
\]
\[
\Delta w_k = \left[ J^\top (w_k) J (w_k) + \mu I \right]^{-1} J^\top (w_k) e(w_k)
\]  \hspace{1cm} (3.11)
As known, learning parameter, $\mu$ is illustrator of steps of actual output movement to desired output. In the standard LM method, $\mu$ is a constant number. The modified learning parameter is given below:

$$\mu = 0.01 e^T e$$  \hspace{1cm} (3.12)

Where $e$ is a $k \times 1$ matrix therefore $e^T e$ is a $1 \times 1$, therefore $[J^T J + \mu I]$ is invertible.

Since the measurement of error is small, the actual output approaches to the desired output with minimum number of iterations. Thus error oscillation reduces greatly.

3.6 Implementation of MLM

The MATLAB implementation of the Modified Levenberg-Marquardt algorithm is given in Appendix I.

3.7 Summary

This chapter introduces the basic Back propagation Network and the learning methods. A detailed discussion is given on Levenberg-Marquardt algorithm. It also discuss disadvantage of Levenberg-Marquardt algorithm and the steps involved in Modified Levenberg-Marquardt algorithm and its implementation in MATLAB.