

OPTIMIZATION TECHNIQUES APPLIED  
TO NEURAL NETWORKS

By

ZHIMIN LU

Bachelor of Engineering

Shanghai University of Engineering Science

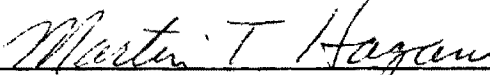
Shanghai, China


1986

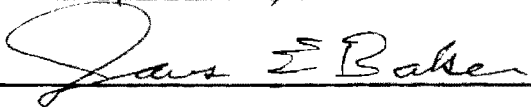
Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December, 1993


OPTIMIZATION TECHNIQUES APPLIED  
TO NEURAL NETWORKS

Thesis Approved:

  
\_\_\_\_\_  
Thesis Adviser

  
\_\_\_\_\_  
Thesis Adviser

  
\_\_\_\_\_  
Thesis Adviser

  
\_\_\_\_\_  
Dean of the Graduate College

## ACKNOWLEDGMENTS

I wish to express my gratitude to Dr. Martin T. Hagan, my advisor, for his guidance, dedication, patience, invaluable instructions and instruction. His insights in academics inspires my interests in this field. He made a great deal of efforts to improve the thesis, both in contents and in English. My appreciation is also extended to Dr. Ronald Rhoten and Dr. James Baker for being members of my committee.

I am grateful to my brother-in-law, Aimin Yan, and sister, Huimin Lu, for their consistent support and encouragement. My deep thanks are dedicated to my parents and another sister and brother-in-law, for their understanding and love.

I would also like to thank Mr. and Mrs. Duncans and Mr. Mike Dower for their friendship. Special thanks to my friends, Weiping Xiao and Zuansun Ren, for their help and encouragement.

## TABLE OF CONTENTS

Chapter	Page
I INTRODUCTION.....	1
II. NEURAL NETWORKS -- BASICS.....	3
Basic Ideas About Neural Networks.....	3
Artificial Neural Network Structure.....	4
Training Artificial Neural Networks.....	7
About Multilayer Neural Networks & Backpropagation.....	8
Properties of Multilayer Neural Networks.....	8
Backpropagation.....	12
Problems With Training An MPNN.....	16
Local Minima.....	16
Overfitting.....	17
Slow Training.....	18
III. OPTIMIZATION TECHNIQUES.....	20
Formulation of the Nonlinear Optimization Problem.....	20
The Newton Method and Its Variations.....	22
The Newton Method.....	22
Some Remarks on the Newton Method.....	24
The Hartley Method.....	25
Least Squares, Gauss-Newton and Marquardt Methods.....	26
Least Squares and the Gauss-Newton Method.....	26
The Marquardt Method.....	29
Iterated Least Squares.....	33
Summary.....	36
IV. MARQUARDT'S METHOD FOR MPNNS.....	37

Chapter	Page
V. RECURSIVE ALGORITHMS FOR TRAINING NEURAL NETWORKS.....	48
Quasi-recursive Gauss-Newton Method.....	49
Recursive Gauss-Newton Method.....	57
Recursive Marquardt Method.....	65
Summary.....	79
VI. APPLICATIONS OF RGN AND RMBP.....	72
Introduction.....	72
Test Problems.....	73
Test Procedures/Results for Prob. #1.....	76
Test Results for RGN.....	76
Test Results for RMBP with Inverse.....	80
Test Results for RMBP without Inverse.....	82
Test Procedures/Results for Prob. #2.....	84
Test Results for RGN.....	84
Test Results for RMBP with Inverse.....	86
Test Results for RMBP without Inverse.....	87
Summary of Test Results.....	89
VII. CONCLUSIONS.....	91
REFERENCES.....	93

## LIST OF FIGURES

Figure		Page
1	Structure of a typical neuron.....	4
2	Perceptron.....	5
3	Topology of feedforward neural networks.....	7
4	Input-output curve of a 1-5-1 MPNN.....	4
5	Function approximation (underfitting).....	11
6	Function approximation (overfitting).....	18
7	Example of trust region operation.....	31
8	Training an MPNN.....	38
9	Parameter updating in QRGN.....	64
10	Parameter updating in RGN.....	64
11	The first test problem.....	74
12	The second test problem.....	75

## LIST OF TABLES

Tables		Page
6.1(a)	Test Results of RGN (6.3) for Prob. #1.....	78
6.1(b)	Test Results of Modified RGN (6.4) for Prob. #1.....	80
6.2	Test Results of RMBP With Inverse for Prob. #1.....	82
6.3	Test Results of RMBP Without Inverse for Prob. #1.....	84
6.4	Test Results of Modified RGN (6.4) for Prob. #2.....	85
6.5	Test Results of RMBP With Inverse for Prob. #2.....	86
6.6	Test Results of RMBP Without Inverse for Prob. #2.....	88

## CHAPTER I

### INTRODUCTION

Artificial neural networks (ANNs) are man-made systems which imitate biological neural networks (BNNs) existing in biological organisms. Even though people do not completely understand BNNs, researchers around the world have found that the up-to-date knowledge about BNNs can be used to design ANNs which exhibit some intelligence and have significant computational capabilities. The research on ANNs also helps the research on BNNs.

Mathematical models of BNNs are used to describe operations and functions of BNNs, and form the basis for the design of ANNs. Among the mathematical models, multilayer feedforward perceptron neural networks (MPNNs) have been intensively investigated, and there exist a large number of articles on their performances and capabilities (see references [2] and [3] for more details). As Hagan and Menhaj pointed out, most existing learning algorithms used to train MPNNs are far from being effective and efficient [1]. There have been efforts to explore more powerful algorithms to circumvent such problems as the slow learning speed in training MPNNs (see References [1] - [10]). One of the basic methods these authors used to accelerate the learning process of MPNNs is to apply some nonlinear optimization techniques to learning algorithms. This research will use the same method to propose three learning



algorithms: the Quasi-recursive Gauss-Newton method (QRGN), the Recursive Gauss-Newton method (RGN) and the Recursive Marquardt Backpropagation method (RMBP). The purpose of developing RGN and RMBP is to train MPNNs with data which is sequentially presented to MPNNs, as in real-time control and signal processing problems.

Chapter 2 provides an outline of ANNs, especially MPNNs. Because many good articles are available in the literature, this introduction is brief. Complete descriptions of MPNNs can be found in references [1] - [3] and [16]. A basic learning algorithm for training MPNNs, Backpropagation, will be described in Chapter 2.

Because the nonlinear optimization techniques are the main source for us to derive learning algorithms for training MPNNs, Chapter 3 summarizes some powerful nonlinear optimization techniques. Based on these techniques, we will derive the batching Gauss-Newton method and the Marquardt method. Based on the batching algorithms developed in Chapter 4, we will further derive three algorithms, namely the QRGN, RGN and RMBP in Chapter 5.

In Chapter 6, we will test the algorithms developed in Chapter 5 on two nonlinear function approximation problems. Various design variables will be investigated. Chapter 7 will present conclusions and further remarks.

## CHAPTER II

### NEURAL NETWORKS -- BASICS

#### 2.1 Basic Ideas About Neural Networks

Researchers would like to develop artificial neural networks (ANNs) which imitate the functions of biological neural networks (BNNs), because BNNs can:

- perform complex actions precisely, often in noisy and uncertain environments,
- adapt themselves to the changes in their environment,
- learn to recognize and respond to stimuli properly, and
- generalize from experiences obtained previously.

The mechanisms of BNNs' organization and operation are partially known. The knowledge of them available up to now is the important source for creating new approaches for designing intelligent devices, such as intelligent controllers.

The research on neurophysiology and psychology reveals that neurons are the basic building blocks of the nervous system. A BNN consists of large numbers of neurons connected together in various ways. Even though the neurons are simple in function and slow in processing speed, the BNN is very versatile in function and fast in processing speed.

Based on the mathematical models developed to describe the

organization and operation mechanisms of BNNs, various artificial neural networks (ANNs) have been proposed in the previous decades. At present, researchers not only develop ANNs to imitate the functions of BNNs better, but also make efforts to apply ANNs to solve difficult engineering problems, such as controlling nonlinear dynamic systems.

### 2.1.1 Artificial Neural Network Structure

An ANN normally consists of a large number of processing elements which are interconnected with some topology. It is the topology and the strength (weight) of the connections which determine the performance of the network. A learning algorithm is used to update the weight connections of an ANN during the training period.

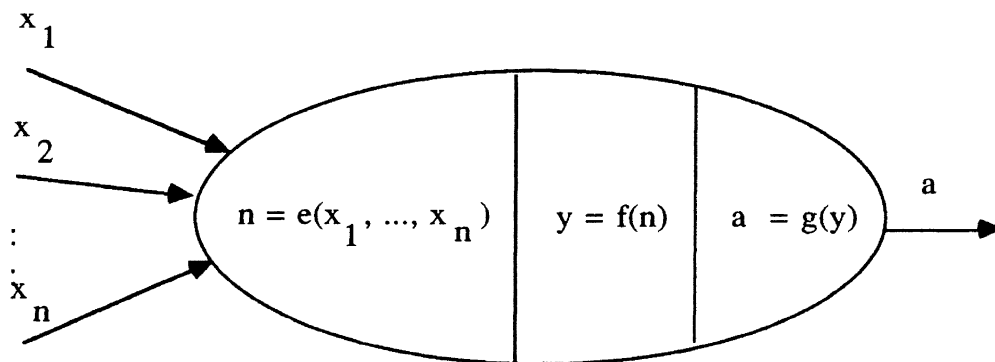


Figure 1. Structure of a typical neuron

A neuron typically has the structure depicted in Figure 1 (also see [7]), where  $e$  is a mapping from input  $\underline{x}$  to  $n$ . The purpose of  $e$  is

to encode the input signals, since the input signals are normally corrupted by noise. The mapping  $f$  from  $n$  to  $y$  determines the activation of the neuron,  $g$  is a mapping from  $y$  to  $a$ , which forms the output signal.

The perceptron is the most popular model of the neuron in the current literature, where the mapping  $e$  is an affine function, and the output mapping  $a$  is a linear function. The activation of a perceptron results from a nonlinear function of a weighted sum of its inputs, which is normally either a sigmoid or a hard limit function. For the perceptron, Figure 1 can be simplified as Figure 2.

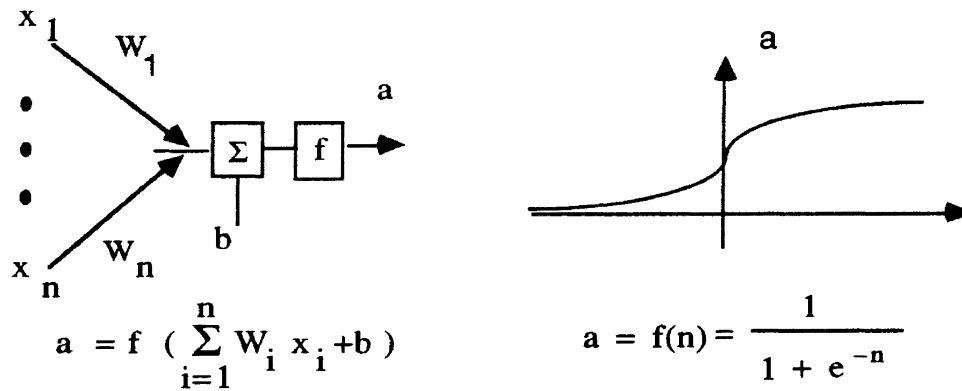


Figure 2. Perceptron

The structure of the perceptron depicted in Figure 2 is a static model, which can be described as

$$a = f \left( \sum_{i=1}^n w_i x_i + b \right)$$

where

$a$  is the output of the neuron,

$x_i$  is the  $i$ th input,

$w_i$  is the weight for the  $i$ th input, and

$b$  is the bias.

The interconnection topologies of ANNs can be divided into following categories:

- **Feedforward:** an ANN in this category consists of an input layer, several hidden layers and an output layer. Figure 3 indicates an ANN with this kind of topology which has one input layer, two hidden layers and one output layer.
- **Feedback:** a neuron of an ANN in this category normally connects to other neurons as well as itself (e.g. the Hopfield network [8]).
- **Mixed topology.** An ANN in this category consists of both of the topologies mentioned above. (The counterbackpropagation network of Hecht-Nielson belongs to this category [8]).

The topological features of the feedforward ANN have the following features:

- Each output of every neuron in a layer is connected to each input of any neuron in the next layer (Refer to Figure 3)
- Because there are no feedback connections between layers, the effect of the feedforward neural network topology is to produce a nonlinear mapping between the input nodes and output nodes. As long as the weights are fixed, this mapping is completely determined.

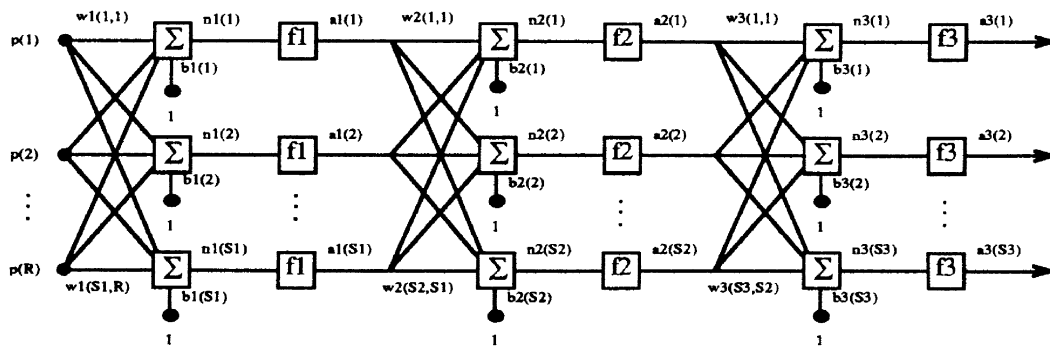


Figure 3. Topology of feedforward neural network

### 2.1.2 Training Artificial Neural Networks

An artificial neural network can be used in one of the two modes:

- operational mode, where weights are fixed so that a given input leads to a determined output.
- learning mode, where we adjust weights so that the output approaches some desired results. In the learning mode, the performance of the ANN is determined by a learning algorithm. Some kinds of learning algorithms which may not be related to the behavior of BNNs are derived from optimization techniques.

Learning algorithms are generally divided into:

- Supervised learning. The desired network output is known for each network input. During learning each member of the training set is presented to the network individually, and upon each presentation the weights are readjusted. After the entire training set is presented, the set is presented again many times. At first, the performance of the network

is improved, but eventually the performance stops improving and the network is said to have converged. Two possibilities after convergence exist: either the network learns the examples from the training set successfully, or it fails.

- **Unsupervised learning.** The desired output is unknown but learning is based on the statistical characteristics of the training data. Such learning algorithms usually learn to extract features from a set of training data.
- **Reinforcement learning.** Weights associated with a neuron are not changed corresponding to the output error of that particular neuron, but instead are changed in proportion to a global reinforcement signal. Such a signal may give a qualitative measure of performance, good or bad (+1 or -1)

In this thesis, we focus on the ANNs with perceptron type neurons, feedforward topologies and supervised learning laws.

## 2.2 About Multilayer Perceptron Neural Networks and Backpropagation

### 2.2.1 Properties of Multilayer Neural Networks

It has been shown that an MPNN is a universal mapping [17], if there are enough hidden neurons in the MPNN. The term "universal mapping" means that any relationship between state variables and control variables, or patterns and features, or stimuli and response can be expressed by an appropriately defined and well-trained MPNN. That an MPNN is defined appropriately means the weights

and biases of the MPNN are chosen correctly. Figure 4 illustrates how an MPNN can be used to approximate a function when the weights and biases are appropriately selected. This MPNN has one input terminal, one linear output terminal, and five hyperbolic tangent sigmoid hidden neurons (We will denote this as 1-5-1).

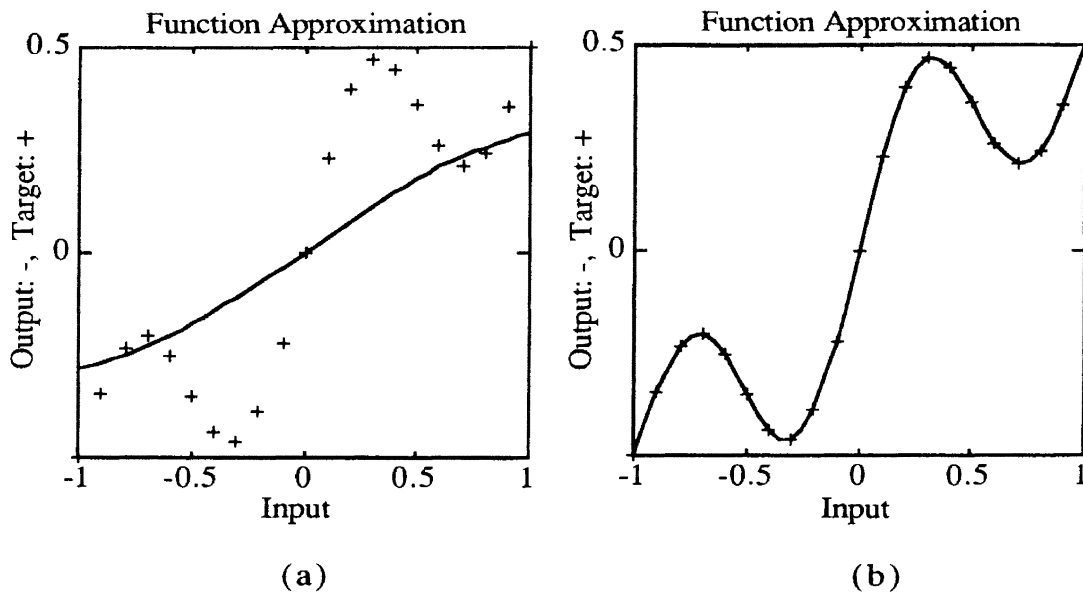


Figure 4. Input-output curve of a 1-5-1 MPNN

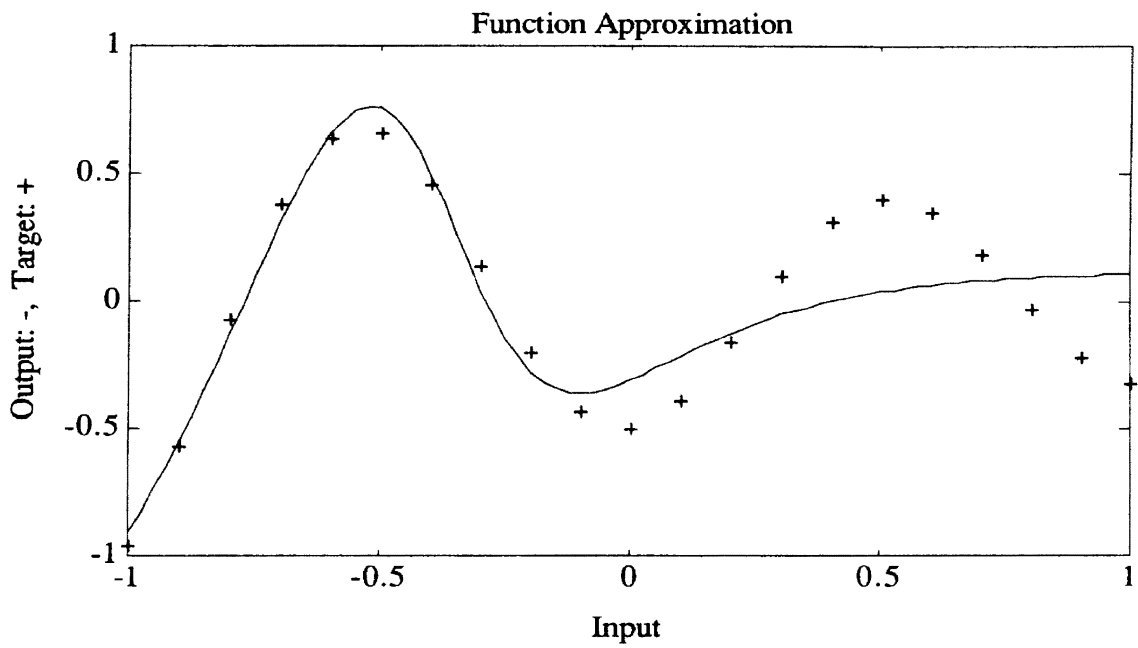
- (a) During training, weights & biases are not appropriately determined.
- (b) After training, weights & biases are correctly determined

Another characteristic of the MPNN is the generalization capacity. An well-trained MPNN can satisfactorily interpolate or extrapolate the output values for input values that are not shown during training. This is very useful in control engineering where only a small subset of data sampled from a dynamic system can be

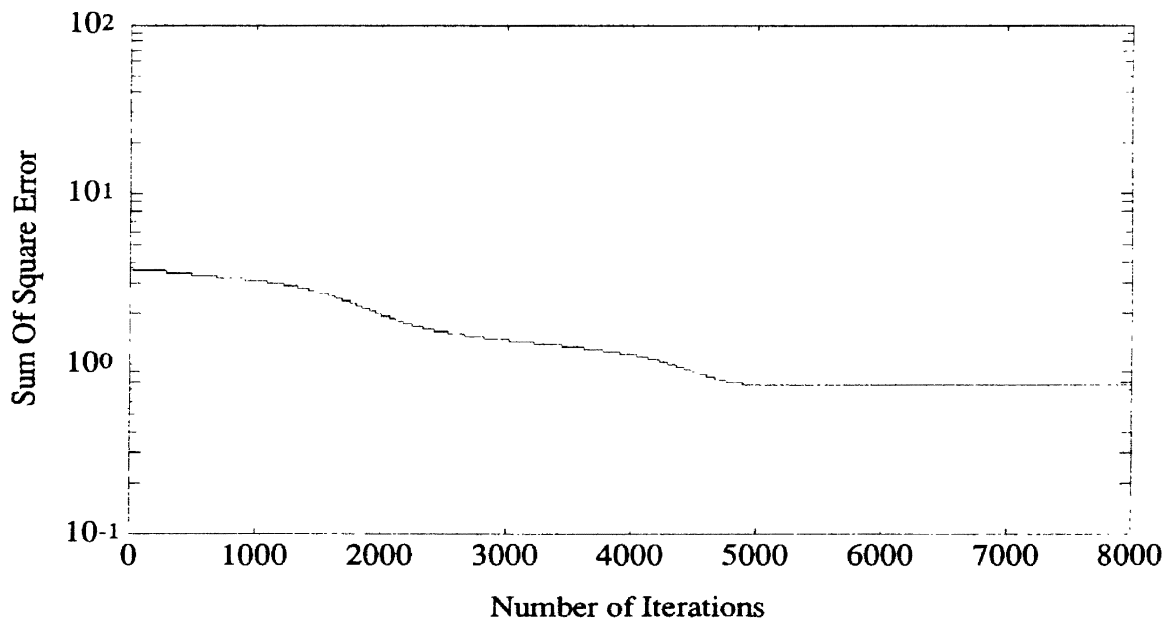


presented to the MPNN during training. An MPNN often makes a good interpolation based on a subset of data [15].

The universal mapping and generalization capability of an MPNN largely depends on the number of hidden neurons. Sometimes we may encounter an underfitting problem. In other words, no matter how long we train the MPNN, the MPNN cannot give a satisfactory approximation to the data. Figure 5 illustrates a case where a 1-2-1 network is used to approximate a function [23]. The two neurons in the hidden layer are not sufficient to allow the network to properly approximate the function. Figure 5 (a) illustrates the best approximation obtained by the backpropagation learning algorithm (which will be described in the next section). Figure 5 (b) shows the learning curve. Note that the learning process stopped after 5000 iterations. This situation can be avoided by adding more hidden neurons to the original MPNN.



(a)



(b)

**Figure 5. Function Approximation (Underfitting)**

### 2.2.2 Backpropagation

The capability of universal mapping and generalization of an MPNN is generated from a successful training process. Training an MPNN requires an learning algorithm and a data set as examples. Backpropagation is the most widely used learning algorithm. It is also the basis on which we understand and derive other learning algorithms.

An MPNN consists of several layers of perceptrons (see Figure 3). We define  $\underline{p}_q$  as the  $q$ th input pattern vector,  $\underline{a}_q$  as the output vector corresponding to that  $\underline{p}_q$ ,  $\underline{a}^k$  as the output of the  $k$ th layer,  $\underline{W}^k$  and  $\underline{b}^k$  as weights and biases related to the  $k$ th layer,  $\underline{t}_q$  as the desired output for  $\underline{p}_q$ , and  $\underline{f}^k(\cdot)$  as the activation function vector for the  $k$ th layer. We can derive the output of the MPNN upon the  $q$ th input pattern  $\underline{p}_q$  as

$$\underline{a}^0 = \underline{p}_q; \quad (2-1)$$

$$\underline{a}^{k+1} = \underline{f}^{k+1}(\underline{W}^{k+1} \underline{a}^k + \underline{b}^{k+1}) \quad k = 0, \dots, M-1; \quad (2-2)$$

$$\underline{a} = \underline{a}^M; \quad (2-3)$$

The training task is to minimize a least squares performance index (PI)

$$J = \sum_{q=1}^Q [(\underline{t}_q - \underline{a}_q)^T (\underline{t}_q - \underline{a}_q)] \quad (2-4)$$

Following the same procedure as in [1], an approximation to the above PI is

$$\hat{J} = (\underline{t}_q - \underline{a}_q)^T (\underline{t}_q - \underline{a}_q) = \underline{t}_q^T \underline{t}_q - 2 \underline{t}_q^T \underline{a}_q + \underline{a}_q^T \underline{a}_q \quad (2-5)$$

We define the derivative of the approximate PI with respect to  $n^k(i)$  as

$$\delta^k(i) = \frac{\partial \hat{J}}{\partial n^k(i)}, \quad (2-6)$$

where  $n^k(i)$  is the net input to neuron  $i$  in layer  $k$

$$n^k(i) = \sum_{j=1}^{S_{k-1}} W^k(i, j) a^{k-1}(j) + b^k(i). \quad (2-7)$$

$S_{k-1}$  is the number of neurons in the  $(k-1)$ th layer. Therefore,

$$\frac{\partial \hat{J}}{\partial W^k(i, j)} = \frac{\partial \hat{J}}{\partial n^k(i)} \frac{\partial n^k(i)}{\partial W^k(i, j)} = \delta^k(i) a^{k-1}(j) \quad (2-8)$$

$$\frac{\partial \hat{J}}{\partial b^k(i)} = \frac{\partial \hat{J}}{\partial n^k(i)} \frac{\partial n^k(i)}{\partial b^k(i)} = \delta^k(i) \quad (2-9)$$

According to the steepest descent rule [1], the weights and biases are updated according to

$$W^{k(i,j)\text{new}} = W^{k(i,j)\text{old}} - \alpha \frac{\partial \hat{J}}{\partial W^{k(i,j)}} \quad (2-10)$$

$$b^{k(i)\text{new}} = b^{k(i)\text{old}} - \alpha \frac{\partial \hat{J}}{\partial b^{k(i)}} \quad (2-11)$$

where  $\alpha$  is a learning rate. The  $\delta^k(i)$  (for all  $k$ ) can be calculated by the backpropagation (BP) rule as

$$\underline{\delta}^M = -\dot{F}^M(\underline{n}^M)(\underline{t}_q - \underline{a}_q) \quad (2-12)$$

$$\underline{\delta}^k = \dot{F}^k(\underline{n}^k) W^{k+1T} \underline{\delta}^{k+1} \quad k = M-1, M-2, \dots, 1 \quad (2-13)$$

where

$$\dot{F}^k(\underline{n}^k) = \begin{bmatrix} \dot{f}^k(n^k(1)) & 0 & \dots & 0 \\ 0 & \dot{f}^k(n^k(2)) & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \dot{f}^k(n^k(S_k)) \end{bmatrix}$$

and

$$\dot{f}^k(n) = \frac{df^k(n)}{dn}$$

Backpropagation is derived based on the following facts

$$\begin{aligned} \frac{\partial n^{k+1}(i)}{\partial n^k(j)} &= \frac{\partial \sum_{m=1}^{S_k} W^{k+1}(i, m) a^k(m) + b^{k+1}(i)}{\partial n^k(j)} \\ &= W^{k+1}(i, j) \dot{f}^k(n^k(j)) \end{aligned} \quad (2-14)$$

$$\begin{aligned} \delta^k(j) &= \frac{\partial \hat{J}}{\partial n^k(j)} = \frac{\partial \hat{J}}{\partial n^{k+1}(i)} \frac{\partial n^{k+1}(i)}{\partial n^k(j)} \\ &= \delta^{k+1}(i) W^{k+1}(i, j) \dot{f}^k(n^k(j)) \end{aligned} \quad (2-15)$$

Therefore, when we train an MPNN by backpropagation, we should go through the following steps

(1) -> (2) -> (3) -> (12) -> (13) -> (8) -> (9) -> (10) -> (11).

With the standard BP algorithm, the multilayer feedforward perceptron neural network is presented with a set of patterns, the input patterns paired with output target patterns. Upon each presentation, weights and biases of the neural network are adjusted to decrease the difference between the network output and the target output. A training set is used for training and is presented to the network many times until the error is less than an acceptable value, the error goal.

Both the forward and backward propagation steps are performed for each pattern presentation during training. The error correction step takes place after a pattern is presented at the input

layer and the forward propagation step is complete. Each processing neuron in the output layer produces a single real value, which is compared to the target output specified in the training example set. Based on the difference, an error value is calculated for each neuron in the output layer. Then, the weights of the interconnection going to the output layer and the biases of the output neurons are adjusted. Next an error sensitivity ( $\delta^k(i)$ ) is calculated for all of the neurons in the hidden layer that just proceeds the output layer. Then, the weights and biases of the proceeding layer of the hidden neurons are adjusted. The process is continued until the last layer of weights has been adjusted.

## 2.3 Problems With Training An MPNN

### 2.3.1 Local Minima

Training neural networks can be considered as a nonlinear regression problem. We can treat a multilayered perceptron neural network (MPNN) as a specific kind of nonlinear regression model. What we have to do is to determine the parameters of the model, the weights and the biases, that provides the best fit to the data. From experience with nonlinear regression, we know that if the structure of the model is not appropriate, the fitting process may not converge. It is same in the training of MPNNs. If the number of hidden layers and the number of hidden neurons are not sufficient, then the MPNN will not be able to fit a given set of data, and the training process will not converge.

Theoretically, the MPNN can approximate arbitrary non-linear

functions if the MPNN contains a sufficient number of hidden layers and a sufficient number of neurons in each hidden layer. In practice, however, even when we have a sufficient number of hidden layers and a sufficient number of neurons in each hidden layer, sometimes the learning algorithm will not converge to a satisfactory solution. The reason is that in the error-weight space, there exist some local minima which may trap the weights so that the error never decreases. But it has been shown that the local minima can be avoided if we restart the training process with a new set of initial weights and maintain sufficient hidden neurons.

### 2.3.2 Overfitting

It is worth pointing out that if the data presented to a MPNN are contaminated with noise, it may happen that the noisy data set may be fitted "precisely." This is called overfitting (Refer to Figure 6). Overfitting also takes a long time for training. We can avoid overfitting by processing the contaminated data before presenting it to an MPNN, by using a small number of hidden neurons, or by using a separate "validation" data set to determine when to stop training.



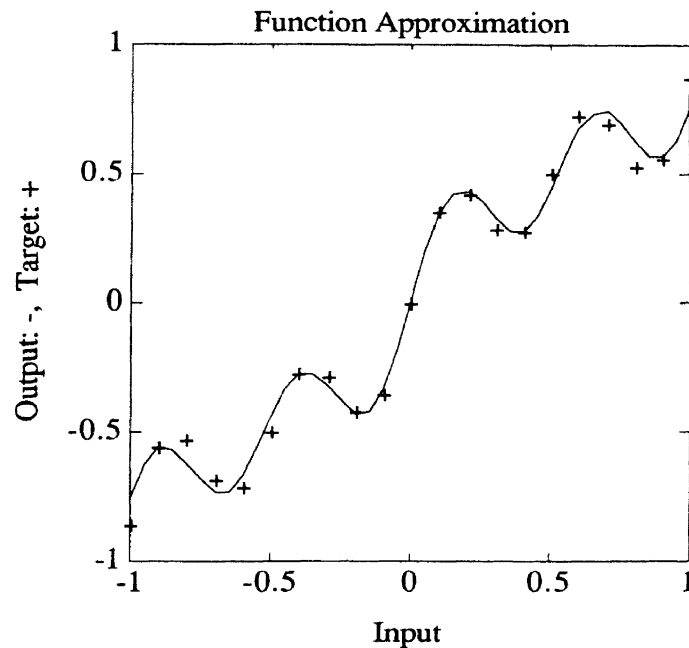


Figure 6 A 1-7-1 MPNN to approximate a function  $y = \sin(\pi/3 * x)$  which is contaminated with noise  $0.18 * \sin(3\pi x) + 0.1 * \sin(2\pi x)$ .

### 2.3.3 Slow Training

In addition to the local minima and overfitting problems, the slow learning speed of standard backpropagation can also be a problem. This has prohibited the application of backpropagation to large problems. Some modifications of backpropagation have been devised to improve the convergence rate. These modifications are generally cast into two categories: those which are based on standard optimization techniques (e. g., conjugate gradient, quasi-Newton, extended Kalman filter), and those which use ad hoc techniques (e. g., momentum, appropriate initialization of weights and biases, variable

learning rate).

Besides the efforts to devise more powerful learning algorithms, it is reported that changing the performance index from a sum of squares of errors to some other kind of performance index can improve the convergence rate. For example, [14] proposed a performance index based on a log measure.

Most of the algorithms for training MPNNs use first derivatives to determine search directions. The information provided by the first derivatives may not be sufficient. It was shown [1] that if we use more sophisticated methods, such as the Gauss-Newton method and the Marquardt method, we might greatly speed up the training process. In the next chapter, this problem will be discussed in more detail.

## CHAPTER III

### OPTIMIZATION TECHNIQUES

#### 3.1 Formulation of the Nonlinear Optimization Problem

We noted in Chapter 2 that optimization techniques, especially nonlinear optimization techniques, form the backbone of many learning algorithms for training MPNNs. Nonlinear optimization techniques are iterative methods for finding the extremum (minimum or maximum) of a nonlinear function. Consider an unconstrained nonlinear optimization problem as follows:

$$\min_{\underline{\theta}} S(\underline{\theta}), \quad (3-1)$$

where the vector  $\underline{\theta}$  contains the free parameters and  $S$  is a nonlinear function. There are no universal analytical approaches to obtain the solution of (3-1). Numerical methods for the solution of (3-1) are typically iterative. They start with some initial guess  $\underline{\theta}_0$  and then proceed according to an equation of the form:

$$\underline{\theta}_{k+1} = \underline{\theta}_k + \alpha_k \underline{p}_k, \quad (3-2)$$

where  $\alpha_k$  is a scalar step size and  $\underline{p}_k$  is a search direction. This research will focus on techniques where the search direction is

determined by using derivatives of  $S$  with respect to the free parameters  $\underline{\theta}$ .

Solving the optimization problem generally deals with the following questions:

(1) Does there exist  $\underline{\theta}^* \in \mathbb{R}^n$  such that  $S(\underline{\theta}^*) \leq S(\underline{\theta}) \quad \forall \underline{\theta} \in \mathbb{R}^n$ ?

If so,  $\underline{\theta}^*$  is called a global minimum. This question addresses the existence of the solution of (3-1).

(2) If the answer to (1) is yes, is there only one such point?

This question addresses the uniqueness of the solution of (3-1).

(3) If the answers to both (1) and (2) are yes, how do we find  $\underline{\theta}^*$ ? If it is impossible to find  $\underline{\theta}^*$ , or the procedure is very complicated, how do we find vectors arbitrarily close to  $\underline{\theta}^*$ ?

We can restate questions (3) mathematically:

(3') How can we find a series of vectors  $\{\underline{\theta}_k\}$  such that

$$\lim_{k \rightarrow \infty} \|\underline{\theta}^* - \underline{\theta}_k\| = 0$$

Formula (3-2) proposes a technique for obtaining  $\{\underline{\theta}_k\}$ .

(4) Do there exist points  $\underline{\theta}^* \in \mathbb{R}^n$  such that  $S(\underline{\theta}^*) \leq S(\underline{\theta})$  whenever  $\underline{\theta}$  is a point in  $\mathbb{R}^n$  lying near  $\underline{\theta}^*$ ? If so,  $\underline{\theta}^*$  is called a local minimum.

We can restate question (4) mathematically:

(4') Do there exist points  $\underline{\theta}^* \in \mathbb{R}^n$ , for which there exists some  $\delta > 0$  such that  $S(\underline{\theta}^*) \leq S(\underline{\theta})$  for all  $\underline{\theta} \in \mathbb{R}^n$  satisfying  $\|\underline{\theta}^* - \underline{\theta}\| < \delta$ ? If  $\delta$  can be made arbitrarily large, then the

local minimum becomes a global minimum.

We will discuss how to realize (3-2) in the following paragraphs.

One way to realize (3-2) locating a minimum is to set

$$\underline{p}_k = -I \cdot \nabla S(\underline{\theta}_k), \quad (3-3)$$

where  $\nabla S(\underline{\theta}_k)$  is the gradient of  $S$  evaluated at  $\underline{\theta}_k$ , and then to determine an appropriate  $\alpha_k$  by means of some one-dimensional search algorithm. This method is called the steepest descent method since  $-\nabla S(\underline{\theta}_k)$  is the direction in which the function decreases fastest. The term  $\alpha_k$  is called the learning rate in the neural network literature.

Another common minimization procedure is known as Newton's method, which involves the calculation of second derivatives. The remainder of this chapter will emphasize Newton's method, and will discuss some methods for improving Newton's method.

## 3.2 Newton's Method and Variations

### 3.2.1 Newton's Method

Generally, when we are faced with an unconstrained nonlinear optimization problem (3-1), we have no universal methods to solve it directly. Instead we solve the following quadratic optimization problem iteratively:

$$\min_{\underline{\theta}} \left\{ S(\underline{\theta}_k) + \nabla S^T(\underline{\theta}_k)(\underline{\theta} - \underline{\theta}_k) + \frac{1}{2}(\underline{\theta} - \underline{\theta}_k)^T \nabla^2 S(\underline{\theta}_k)(\underline{\theta} - \underline{\theta}_k) \right\} \quad (3-4)$$

assuming the function is continuous and smooth. The quadratic function of (3-4) is the truncated Taylor series expansion of (3-1):

$$S(\underline{\theta}) = S(\underline{\theta}_k) + \nabla S^T(\underline{\theta}_k)(\underline{\theta} - \underline{\theta}_k) + \frac{1}{2}(\underline{\theta} - \underline{\theta}_k)^T \nabla^2 S(\underline{\theta}_k)(\underline{\theta} - \underline{\theta}_k) + \dots \quad (3-5)$$

at  $\underline{\theta}_k$  where  $\nabla S(\underline{\theta}_k)$  and  $\nabla^2 S(\underline{\theta}_k)$  in (3-4) denote gradient vector and Hessian matrix of the function  $S$  respectively. According to the optimality condition [4], the minimum of (3-4) exists only when

$$\left[ \nabla^2 S(\underline{\theta}_k) \right] (\underline{\theta} - \underline{\theta}_k) = -\nabla S(\underline{\theta}_k). \quad (3-6)$$

A possible solution of (3-6) is:

$$\underline{\theta} - \underline{\theta}_k = -\left[ \nabla^2 S(\underline{\theta}_k) \right]^{-1} \nabla S(\underline{\theta}_k), \quad (3-7)$$

which produces the iteration:

$$\underline{\theta}_{k+1} = \underline{\theta}_k - \left[ \nabla^2 S(\underline{\theta}_k) \right]^{-1} \nabla S(\underline{\theta}_k). \quad (3-8)$$

A series  $\{\underline{\theta}_k\}$  can be generated iteratively using (3-8), until the minimum point of (3-1) is found. This is called Newton's method.

### 3.2.2 Some Remarks on Newton's Method

1. If the nonlinear function in (3-1) is quadratic, then Newton's method will converge in one step. For general smooth nonlinear functions, if the search occurs in the vicinity of the minimum, it can be shown ([4] and [6]) that the nonlinear function can be approximated well enough by the quadratic function so as to exhibit rapid convergence in the region.
2. Comparing (3-8) with (3-3), we can see that the Hessian matrix in Newton's method replaces the diagonal matrix in the steepest descent method.
3. Since, from (3-5), the function

$$S(\underline{\theta}_k) + \nabla S^T(\underline{\theta}_k)(\underline{\theta} - \underline{\theta}_k) + \frac{1}{2}(\underline{\theta} - \underline{\theta}_k)^T \nabla^2 S(\underline{\theta}_k)(\underline{\theta} - \underline{\theta}_k) \quad (3-9)$$

is, in general, an approximation of  $S(\underline{\theta})$  at  $\underline{\theta}_k$ . The sequence  $\{\underline{\theta}_k\}$  generated from (3-8) may not approach  $\underline{\theta}^*$  if the approximation is not accurate or if the initial guess,  $\underline{\theta}_0$ , is far from the true minimum. In addition, the repetitive application of (3-8) is based on the assumption that a positive definite Hessian matrix always exists. When the parameters and the coefficients in the nonlinear function are not scaled properly, or the round-off errors are significant, or if in the vicinity of some point  $\underline{\theta}_k$  the function is not convex, then the Hessian matrix will not be positive definite.

In order to improve Newton's method in these circumstances, many techniques have been proposed (see [1], [4] - [6], and [13]).

Some of them adapt linear algebra computation techniques to avoid calculating the inverse of the Hessian matrix in (3-6). Some of them use more sophisticated techniques to maintain the numerical stability and tractability of the Hessian matrix. In the following sections, we describe some of these techniques.

4. Newton's method and its variants are the fastest and most robust algorithms for unconstrained optimization of a general smooth function ([1], [2], [4], and [6]). Even though it is important to scale the parameters properly in practical computation to avoid an ill-conditioned Hessian matrix, Newton's method is less sensitive to scaling than are the steepest descent method and its variants [4]. The weight adjustments based on Newton's method use curvature information in addition to gradient information, so that the training algorithms derived from Newton's method and its variants are more efficient than non-Newton methods. In addition, the variants of Newton's method mentioned above make the computation results more reliable.

### 3.2.3 Hartley's Method

From (3-8), we can see that a minimum can be reached by means of Newton's method if the Hessian matrix is always positive definite. However, the value found by (3-8) may not be a minimum of the original nonlinear function but of the truncated second order approximation of the original function. It is therefore necessary to make an initial estimate of  $\underline{\theta}^*$ ,  $\underline{\theta}_0$ , sufficiently close to  $\underline{\theta}^*$  to assure that the iterations approach the minimum.



In some situations, the step generated from (3-8) may not decrease the function even though the Hessian is positive definite. So a modification can be made by adding an adjustable parameter  $\alpha_k$  to (3-8), this is called the Hartley method [2] and corresponds to the variable step size steepest descent method (3-3). The Hartley method is:

$$\underline{\theta}_{k+1} = \underline{\theta}_k - \alpha_k [\nabla^2 S(\underline{\theta}_k)]^{-1} \nabla S(\underline{\theta}_k) \quad (3-10)$$

where  $\alpha_k$  can be determined by line search algorithms. The detailed procedures for determining  $\alpha_k$  can be found in [4].

### 3.3 Least Squares, Gauss-Newton and Marquardt Methods

#### 3.3.1 Least Squares and the Gauss-Newton Method

We have investigated Newton's method for the general nonlinear optimization problem. Least squares is a special type of nonlinear optimization problem. The Gauss-Newton method applies Newton's method to the least squares problem and simplifies the complex calculation of the Hessian matrix. It was shown in [16] that if the initial guess  $\underline{\theta}_0$  is sufficiently close to  $\underline{\theta}^*$ , the iteration using the Gauss-Newton method converges quadratically.

Assume that we have a nonlinear regression equation with single input/single output:

$$t(q) = a(\underline{\theta}, p(q)) + v(q) \quad (3-11)$$

where

$$\underline{\theta} \in \mathbb{R}^n; \quad t \in \mathbb{R}^1; \quad a \in \mathbb{R}^1; \quad q = 1, \dots, Q$$

$\underline{\theta}$  is the parameter vector,  $t_q$  is the dependent variable and  $p_q$  is the independent variable. If we are given  $Q$  equations with the form of (3-11) and  $Q$  pairs of examples  $[p_q, t_q]$  ( $q=1, \dots, Q$ ), then the least squares problem can be formulated as

$$\min_{\underline{\theta}} S(\underline{\theta}) = \min_{\underline{\theta}} \sum_{q=1}^Q |t_q - a(\underline{\theta}, p_q)|^2$$

If we define

$$\underline{f} = \begin{bmatrix} t_1 - a(\underline{\theta}, p_1) \\ \vdots \\ t_Q - a(\underline{\theta}, p_Q) \end{bmatrix}; \quad \underline{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_Q \end{bmatrix}; \quad \underline{a} = \begin{bmatrix} a(\underline{\theta}, p_1) \\ \vdots \\ a(\underline{\theta}, p_Q) \end{bmatrix}.$$

then

$$S(\underline{\theta}) = \sum_{q=1}^Q |t_q - a(\underline{\theta}, p_q)|^2 = \underline{f}^T \underline{f} \quad (\underline{f} \in \mathbb{R}^Q) \quad (3-12)$$

From (3-12), we have

$$\nabla S(\underline{\theta}) = 2J_f^T \cdot \underline{f}$$

and

$$\nabla^2 S(\underline{\theta}) = 2 \left[ J_f^T J_f + B(\underline{\theta}) \right]$$

where

$$J_f = \begin{bmatrix} \nabla f_1^T \\ \vdots \\ \nabla f_Q^T \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_Q}{\partial \theta_1} & \frac{\partial f_Q}{\partial \theta_2} & \cdots & \frac{\partial f_Q}{\partial \theta_n} \end{bmatrix} \quad (3-13)$$

is a Jacobian matrix and

$$B(\underline{\theta}) = \sum_{q=1}^Q f_q(\underline{\theta}) \cdot \nabla^2 f_q(\underline{\theta}) = \sum_{q=1}^Q (t_q - a(\underline{\theta}, p_q)) \nabla^2 a_q \quad (3-14)$$

where  $\nabla^2 f_q(\underline{\theta})$  is the Hessian matrix of  $f_q(\underline{\theta})$  evaluated at  $\underline{\theta}_i$ . Using (3-8), Newton's method becomes:

$$\begin{aligned} \underline{\theta}_{i+1} &= \underline{\theta}_i - \nabla^2 S(\underline{\theta})^{-1} \nabla S(\underline{\theta}) \\ &= \underline{\theta}_i - \left[ 2J_f^T J_f + B(\underline{\theta}) \right]^{-1} \left[ 2J_f^T \cdot \underline{f} \right] \\ &= \underline{\theta}_i - \left[ 2J_f^T J_f + 2 \sum_{q=1}^Q f_q(\underline{\theta}) \nabla^2 f_q(\underline{\theta}) \right]^{-1} \left[ 2J_f^T \cdot \underline{f} \right] \end{aligned} \quad (3-15)$$

Consider the second term in  $\nabla^2 S(\underline{\theta})$ ,  $B(\underline{\theta})$ . If the residuals are so small (that is,  $\underline{f}$  is small) that  $B(\underline{\theta})$  can be neglected, we get the Gauss-Newton method:

$$\underline{\theta}_{i+1} = \underline{\theta}_i - \left[ J_f^T J_f \right]^{-1} \left[ J_f^T \cdot \underline{f} \right] \quad (3-16)$$

### 3.3.2 The Marquardt Method

In practice, it is found that the Gauss-Newton method does one of the three things [6]:

- quickly converges to a minimum; or
- quickly diverges to infinity; or
- calculates values that wander about.

A part of the convergence problem with the Gauss-Newton method arises from the Jacobian matrix being ill-conditioned or indefinite. Omitting  $B(\underline{\theta})$  from the exact Hessian matrix is also an intrinsic problem for computation. Accumulated calculation error or round-off error may cause the  $[J^T J]$  matrix to become ill-conditioned or non-invertible.

Marquardt added a  $\mu I$  matrix to the  $[J^T J]$  term. This provides a tool to fix the ill-conditioned Hessian matrix during calculation of the inverse. It has been shown ([1], [2], [4], and [6]) from many practical experiences that the Marquardt  $\mu$  can be very effective. The Marquardt method is:

$$\underline{\theta}_{i+1} = \underline{\theta}_i - \left[ \mathbf{J}_f^T \mathbf{J}_f + \mu \mathbf{I} \right]^{-1} \left[ \mathbf{J}_f^T \cdot \underline{\mathbf{f}} \right] \quad (3-17)$$

The parameter  $\mu$  is multiplied by some factor ( $\beta > 1$ ) whenever a step would result in an increased  $S(\underline{\theta})$ . When a step reduces  $S(\underline{\theta})$ ,  $\mu$  is divided by  $\beta$ .

The function of the Marquardt  $\mu$  is related to the trust region method [6]. We know that the Newton method and Gauss-Newton method are derived from a truncated Taylor series expansion, which is an approximation of the original nonlinear function. Therefore the result obtained from (3-8) or (3-16) must be justified by  $(S(\underline{\theta}_{i+1}) - S(\underline{\theta}_i)) < 0$ . The  $\underline{\theta}_{i+1}$  derived from (3-8) or (3-16) may be far from  $\underline{\theta}_i$ , so that  $S(\underline{\theta}_{i+1})$  may not be less than  $S(\underline{\theta}_i)$ . In this case, the minimum derived from (3-8) or (3-16) is not a reasonable approximation to the minimum of  $S(\underline{\theta}_i)$ . The trust region method adds a constraint to (3-8) or (3-16):

$$\left| \underline{\theta}_{i+1} - \underline{\theta}_i \right| < \varepsilon \quad (3-18)$$

rather than checking  $(S(\underline{\theta}_{i+1}) - S(\underline{\theta}_i)) < 0$ . That is, we can treat our nonlinear least square problem as a constrained optimization problem:

$$S(\underline{\theta}) = \sum_{q=1}^Q \left| t_q - a(\underline{\theta}, p_q) \right|^2 = \underline{\mathbf{f}}^T \underline{\mathbf{f}} \quad (\underline{\mathbf{f}} \in \mathbb{R}^Q) \quad (3-19)$$

Subject to

$$|\underline{\theta}_{i+1} - \underline{\theta}_i| \leq \epsilon \quad (3-20)$$

where  $\epsilon$  is an adjustable parameter which guarantees that  $S(\underline{\theta})$  will go down. Fletcher [4] gave a formula to determine the  $\epsilon$  at each iteration. It was proved [4] that (3-19) and (3-20), a constrained optimization problem, can be reduced to the Marquardt algorithm using Lagrangian multipliers [4]. The effectiveness of the Marquardt  $\mu$  can also be seen in Figure 8. There the point  $\underline{\theta}_{k+1}$ , derived from a quadratic function (truncated Taylor series expansion), leads to an increase in the nonlinear function. A trust region restricts  $\underline{\theta}_{k+1}$  to  $\underline{\theta}_{k+1}'$ , which decreases the nonlinear function.

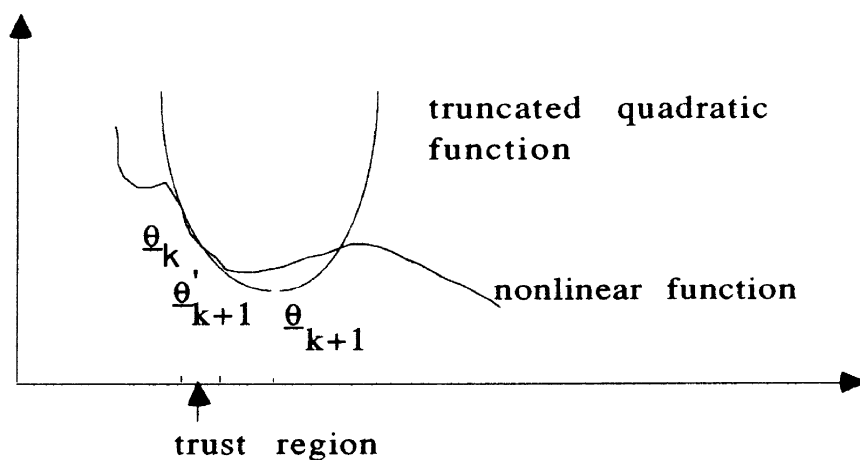


Figure 8 Example of Trust Region Operation

In most cases, the Marquardt method works best among the methods for solving least squares problems. The Gauss-Newton method using  $[J^T J]$  to approximate  $[J^T J + B]$  is effective when  $B$  can be ignored compared with  $[J^T J]$ . Even though the Marquardt method is

effective and efficient in solving least squares problems, it may fail to converge or may converge painfully slowly in some cases where residual errors are large. Gill and Murray defined the "large residual" case as the case when the term  $\det\{B[J^T J]^{-1}\}$  is large. If noisy data with large outliers are present (in realistic problems they invariably are) the second term may dominate the Hessian, so that excluding the term  $B(\underline{\theta})$  prevents (3-13) from approximating Hessian matrix well. Adjusting  $\mu$  may not compensate the omitted term, so that the algorithm may diverge. If there exists a large residual,  $\theta_{i+1} - \theta_i$  is much smaller when using the exact Hessian matrix of  $S(\underline{\theta})$  than when using the Gauss-Newton or the Marquardt approximation to the Hessian matrix. In [6], the problem is discussed in detail.

Now, there are some strategies to deal with the large residual problem. The failure of the Levenberg-Marquardt method and the Gauss-Newton method on large residual problems is caused by ignoring the second term of Hessian matrix:

$$B(\underline{\theta}) = \sum_{q=1}^Q f_q(\underline{\theta}) \cdot \nabla^2 f_q(\underline{\theta})$$

There are two efforts to solve the problem:

- estimate  $B(\underline{\theta})$  using the Quasi-Newton method (see Dennis et al. [9]), and
- dynamically partition the eigenvalues of  $[J^T J]$  into a set of dominant eigenvalues and a complementary set of undominant eigenvalues (see Gill and Murray [6]).

There are many details on the theory and implementation of

these two methods, which are out of the scope of the thesis, see [6] for reference. When we apply the Gauss-Newton and Marquardt methods to an MPNN, if the large residual problem occurs, we can use other techniques to process the training data beforehand. Therefore, we can use the Marquardt method safely and avoid large residual problems.

### 3.4 Iterated Least Squares

The Gauss-Newton method is derived by omitting the second term in the Hessian matrix in the exact Newton method for least squares. The Marquardt method is an improvement on the Gauss-Newton method. In this section we will prove that the formula (3-16) can also be obtained from a method, called iterated least squares (ILS) [13]. This section is devoted to proving the equivalence between the Gauss-Newton method and ILS. By means of ILS, we will develop a batch version (Chapter 4) and a recursive version (Chapter 5) of the learning algorithm to train MPNNs.

We consider a nonlinear regression model with one input and one output terminal as in (3-11). Recall that the performance index for that model is:

$$S(\underline{\theta}) = \sum_{q=1}^Q |t_q - a(\underline{\theta}; p_q)|^2 = \underline{f}^T \underline{f} \quad (3-21)$$

(The result for a multi-input/multi-output model is similar). From (3-16), the Gauss-Newton algorithm to minimize  $S(\underline{\theta})$  is



$$\begin{aligned}
\theta_{i+1} &= \theta_i - \nabla^2 S(\theta)^{-1} \nabla S(\theta) \\
&= \theta_i - \left[ 2J_f^T J_f + B(\theta) \right]^{-1} \left[ 2J_f^T \cdot \underline{f} \right] \\
&\approx \theta_i - \left[ J_f^T J_f \right]^{-1} \left[ J_f^T \cdot \underline{f} \right]
\end{aligned} \tag{3-22}$$

Now we apply ILS to (3-19), in which the nonlinear function is replaced by its first order approximation:

$$\begin{aligned}
S(\theta) &= \sum_{q=1}^Q \left| t_q - a(\theta; \underline{p}_q) \right|^2 \\
&\approx \sum_{q=1}^Q \left| t_q - a(\theta_i; \underline{p}_q) - \nabla^T a(\theta_i; \underline{p}_q) (\theta - \theta_i) \right|^2 \\
&= \underline{f}_1^T \underline{f}_1
\end{aligned} \tag{3-23}$$

where  $\underline{f}_1$  is the linear approximation of  $\underline{f}$ , defined as:

$$\underline{f}_1 = \begin{bmatrix} t_1 - a(\theta_i; \underline{p}_1) - \nabla^T a(\theta_i; \underline{p}_1) (\theta - \theta_i) \\ \vdots \\ t_Q - a(\theta_i; \underline{p}_Q) - \nabla^T a(\theta_i; \underline{p}_Q) (\theta - \theta_i) \end{bmatrix}$$

Furthermore, we define  $S_1(\theta) = \underline{f}_1^T \underline{f}_1$ ;

Note that

$$\underline{f}(\theta_i) = \underline{t} - \underline{a}(\theta_i)$$

Then we have

$$\underline{f}_1 = \underline{f}(\underline{\theta}_i) - J_a^T(\underline{\theta}_i)(\underline{\theta} - \underline{\theta}_i)$$

where

$$J_a^T(\underline{\theta}_i) = \begin{bmatrix} \nabla^T a(\underline{\theta}_i; \underline{p}_1) \\ \vdots \\ \nabla^T a(\underline{\theta}_i; \underline{p}_Q) \end{bmatrix}$$

It is easily found from (3-21) that:

$$\begin{aligned} \nabla S_1(\underline{\theta}) &= \nabla(\underline{f}_1^T \underline{f}_1) = \nabla \left\{ \underline{f}(\underline{\theta}_i)^T \underline{f}(\underline{\theta}_i) + (\underline{\theta} - \underline{\theta}_i)^T J_a^T J_a (\underline{\theta} - \underline{\theta}_i) \right. \\ &\quad \left. - (\underline{\theta} - \underline{\theta}_i)^T J_a^T \underline{f}(\underline{\theta}_i) - \underline{f}^T(\underline{\theta}_i) J_a (\underline{\theta} - \underline{\theta}_i) \right\} \\ &= -2J_a^T \underline{f}(\underline{\theta}_i) + 2J_a^T J_a (\underline{\theta} - \underline{\theta}_i) \end{aligned}$$

Furthermore

$$\begin{aligned} \nabla S_1(\underline{\theta}) &= -2J_a^T \underline{f}(\underline{\theta}_i) + 2J_a^T J_a (\underline{\theta} - \underline{\theta}_i) = 0 \\ \Rightarrow \quad \underline{\theta} - \underline{\theta}_i &= \left[ J_a^T J_a \right]^{-1} J_a^T \underline{f}(\underline{\theta}_i) \end{aligned}$$

Therefore

$$\underline{\theta}_{i+1} = \underline{\theta}_i + \left[ J_a^T J_a \right]^{-1} J_a^T \underline{f}(\underline{\theta}_i) \quad (3-24)$$

We can easily verify (3-23) below:

$$J_a = -J_f \quad (3-25)$$

Hence, substituting (3-23) to (3-22) leads to

$$\underline{\theta}_{i+1} = \underline{\theta}_i - \left[ J_f^T J_f \right]^{-1} \left[ J_f^T \cdot \underline{f}(\underline{\theta}_i) \right] \quad (3-26)$$

We can see that the Gauss-Newton method can be obtained from ILS by truncating the error function  $\underline{f}(\underline{\theta})$  to its first order Taylor series expansion. We will use ILS again in Chapter 4 and 5.

### 3.5 Summary

The results of this chapter are summarized as follows:

- (1) If the Hessian matrix of the objective function is available, Newton's method is most appropriate; a Marquardt  $\mu$  can maintain the search direction down hill.
- (2) In nonlinear least squares, the Marquardt compromise for the Gauss-Newton method is most appropriate, even though it may converge slowly, and sometimes will diverge when the problem involves large residual error.
- (3) There are the complements for the Marquardt algorithm to deal with large residual error effectively, but these algorithms are very complex.

## CHAPTER IV

### MARQUARDT'S METHOD FOR MPNNS

In Chapter 3, Gauss-Newton and Marquardt methods are discussed under the framework of the nonlinear regression problem. This chapter applies these methods to MPNNs and derives batch algorithms to train MPNNs. The conclusions obtained in this chapter are same as those in [1]. However the results are derived using ILS, which was not used in [1]. The two methods were proved equivalent in the last chapter. The ILS method, which is used in this chapter, can be easily extended to derive the recursive versions of training algorithms in the next chapter.

An MPNN can be considered as a nonlinear regression model, where the adjustable parameters,  $\underline{\theta}$ , are the weights and the offsets of the networks. Successful training leads to minimizing the sum of squares of the total residuals with respect to  $\underline{\theta}$ . That is

$$\min_{\underline{\theta}} \sum_{q=1}^Q v_q^2.$$

where  $v_q$  is the difference between the network output and the desired output.

When training an MPNN we first constitute a group of nonlinear equations describing the relations between all the

quantities. Then, a set of examples,  $[p_q; t_q]$  ( $q=1, 2, \dots, Q$ ), are presented to the MPNN sequentially (Figure 9). In order for the MPNN to match the pairs correctly, the set of examples must be presented repeatedly. We call one presentation of the whole set of examples as one epoch.

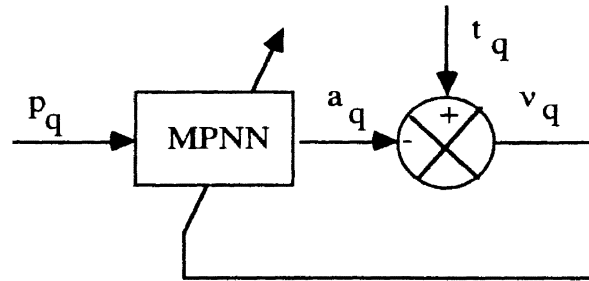


Figure 9 Training an MPNN

The purpose of training MPNNs is to look for an appropriate set of parameters  $\underline{\theta}$ , such that the output of the MPNN matches the desired value  $t$  with minimum error. The ILS method [12] can be used to find the solution. At first, we consider an MPNN with single input/single output. We formulate the expression as:

$$t_q = a(\underline{\theta}; p_q) + v_q \quad q = 1, \dots, Q \quad (4-1)$$

We use the truncated Taylor series expansion to linearize the relationship between  $\underline{\theta}$  and  $t_q$ :

$$\delta t_q \cong \nabla a^T(\underline{\theta}^*; p_q) \delta \underline{\theta} + v'_q \quad (4-2)$$

where

$$\delta t_q = t_q - a(\underline{\theta}^*; p_q) \quad (4-3)$$

$$\delta \underline{\theta} = \underline{\theta} - \underline{\theta}^* \quad (4-4)$$

We define  $\underline{\theta}^*$  as a nominal value of  $\underline{\theta}$ . We stack (4-2) through all examples ( $q=1, \dots, Q$ ) and obtain:

$$\delta \underline{t} = J_a * \delta \underline{\theta} + \underline{v}' \quad (4-5)$$

where

$$J_a = \begin{bmatrix} \frac{\partial a(\underline{\theta}; p_1)}{\partial \theta_1} & \frac{\partial a(\underline{\theta}; p_1)}{\partial \theta_2} & \dots & \frac{\partial a(\underline{\theta}; p_1)}{\partial \theta_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial a(\underline{\theta}; p_Q)}{\partial \theta_1} & \frac{\partial a(\underline{\theta}; p_Q)}{\partial \theta_2} & \dots & \frac{\partial a(\underline{\theta}; p_Q)}{\partial \theta_n} \end{bmatrix}_{\underline{\theta}^*}$$

and

$$\underline{\theta} \in \mathbb{R}^n, \quad \delta \underline{t} \in \mathbb{R}^Q, \quad J_a \in \mathbb{R}^{Q \times n}, \quad \underline{v}' \in \mathbb{R}^Q.$$

Solving (4-5) for  $\delta \underline{\theta}$  by linear least squares [13] leads to:

$$\delta \underline{\theta} = (J_a^T J_a)^{-1} J_a^T \delta \underline{t} \quad (4-6)$$

The formula (4-6) has the same form as the solution obtained from the Gauss-Newton method for unconstrained optimization techniques

[1]. Following Levenberg and Marquardt's idea to improve the Gauss-Newton method, we add a term  $\mu$  to compensate the omitted higher order term in the Taylor series expansion:

$$\delta \underline{\theta} = (J_a^T J_a + \mu I_{n \times n})^{-1} J_a^T \delta \underline{t} \quad (4-7)$$

After we get  $\delta \underline{\theta}$  from (4-7), we can adapt the nominal value of  $\underline{\theta}$ ,  $\underline{\theta}^*$ , as follows:

$$\underline{\theta} = \delta \underline{\theta} + \underline{\theta}^* \quad (4-8)$$

Repeating the procedure above, an optimal value of  $\underline{\theta}$  can be obtained, which may have least residual errors. The iteration to find the optimal  $\underline{\theta}^*$  stops when the condition

$$|\delta \underline{\theta}| \leq \varepsilon$$

is satisfied, where  $\varepsilon$  is a predetermined termination value.

Now we consider training a multi-input/multi-output MPNN. The set of training examples, therefore, has the form  $\{\underline{p}_q, \underline{t}_q\}$  ( $q=1, \dots, Q$ ). In this case, the performance index has the form:

$$\min_{\underline{\theta}} \sum_{q=1}^Q \|\underline{v}_q\|^2 = \min_{\underline{\theta}} \sum_{q=1}^Q \sum_{i=1}^{S_m} v_q^2(i) \quad (4-9)$$

The formula (4-9) indicates that the multi-input/multi-output case

has the same form of performance index as does the single input/  
single output case. Therefore, we follow the same procedure to  
derive the solution of optimal  $\underline{\theta}$ :

$$\underline{t}_q = \underline{a}(\underline{\theta}; \underline{p}_q) + \underline{v}_q \quad (4-10)$$

Applying a first order truncated Taylor series expansion to (4-10)  
about a nominal  $\underline{\theta}^*$ , we have:

$$\underline{t}_q = \underline{a}(\underline{\theta}^*; \underline{p}_q) + J_a(\underline{p}_q) \Big|_{\underline{\theta}^*} (\underline{\theta} - \underline{\theta}^*) + \underline{v}_q \quad (4-11)$$

where

$$J_a(\underline{p}_q) \Big|_{\underline{\theta}^*} = \begin{bmatrix} \frac{\partial a_1(\underline{\theta}; \underline{p}_q)}{\partial \theta_1} & \frac{\partial a_1(\underline{\theta}; \underline{p}_q)}{\partial \theta_2} & \dots & \frac{\partial a_1(\underline{\theta}; \underline{p}_q)}{\partial \theta_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial a_{Sm}(\underline{\theta}; \underline{p}_q)}{\partial \theta_1} & \frac{\partial a_{Sm}(\underline{\theta}; \underline{p}_q)}{\partial \theta_2} & \dots & \frac{\partial a_{Sm}(\underline{\theta}; \underline{p}_q)}{\partial \theta_n} \end{bmatrix} \Big|_{\underline{\theta}^*}$$

Defining

$$\delta \underline{t}_q = \underline{t}_q - \underline{a}(\underline{\theta}^*; \underline{p}_q)$$

$$\delta \underline{\theta} = \underline{\theta} - \underline{\theta}^*,$$



we reformulate (4-11) as

$$\delta \underline{t}_q = J_a(\underline{p}_q) \Big|_{\underline{\theta}^*} \delta \underline{\theta} + \underline{v}'_q \quad (4-12)$$

Following the same convention as that in [1], we stack (4-12) through  $q=1, \dots, Q$  and get to:

$$\delta \underline{T} = H \delta \underline{\theta} + \underline{V}' \quad (4-13)$$

where

$$\delta \underline{T} \in \mathbb{R}^{QSm}, \quad H \in \mathbb{R}^{QSm \times n}, \quad \delta \underline{\theta} \in \mathbb{R}^n, \quad \underline{V}' \in \mathbb{R}^{QSm}$$

$$\delta \underline{T} = \begin{bmatrix} \delta t_1 \\ \delta t_2 \\ \vdots \\ \delta t_Q \end{bmatrix}, \quad H = \begin{bmatrix} J_a(\underline{p}_1) \\ \vdots \\ J_a(\underline{p}_Q) \end{bmatrix}, \quad \underline{V}' = \begin{bmatrix} \underline{v}'_1 \\ \underline{v}'_2 \\ \vdots \\ \underline{v}'_Q \end{bmatrix}$$

The solution of (4-13) can be derived by batch least squares estimation techniques [13]:

$$\delta \underline{\theta} = [H^T H]^{-1} H^T \delta \underline{T} \quad (4-14)$$

We clarify the H in (4-14):

$$H = \begin{bmatrix} \frac{\partial a_1(\underline{\theta}; \underline{p}_1)}{\partial \theta_1} & \cdots & \frac{\partial a_1(\underline{\theta}; \underline{p}_1)}{\partial \theta_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial a_{S_m}(\underline{\theta}; \underline{p}_1)}{\partial \theta_1} & \cdots & \frac{\partial a_{S_m}(\underline{\theta}; \underline{p}_1)}{\partial \theta_n} \\ \frac{\partial a_1(\underline{\theta}; \underline{p}_2)}{\partial \theta_1} & \cdots & \frac{\partial a_1(\underline{\theta}; \underline{p}_2)}{\partial \theta_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial a_{S_m}(\underline{\theta}; \underline{p}_2)}{\partial \theta_1} & \cdots & \frac{\partial a_{S_m}(\underline{\theta}; \underline{p}_2)}{\partial \theta_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial a_1(\underline{\theta}; \underline{p}_Q)}{\partial \theta_1} & \cdots & \frac{\partial a_1(\underline{\theta}; \underline{p}_Q)}{\partial \theta_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial a_{S_m}(\underline{\theta}; \underline{p}_Q)}{\partial \theta_1} & \cdots & \frac{\partial a_{S_m}(\underline{\theta}; \underline{p}_Q)}{\partial \theta_n} \end{bmatrix}_{\underline{\theta}^*}$$

Modifying (4-14) by the Marquardt method leads to:

$$\delta \underline{\theta} = \left[ H^T H + \mu I \right]^{-1} H^T \delta \underline{T} \quad (4-15)$$

Then we update  $\underline{\theta}$  with the same procedure as (4-8).

Because the Jacobian matrix in (4-6), (4-7) and (4-14), (4-15) is the key to the constitution of the search direction, it will be derived for MPNNs explicitly next. For an MPNN with  $Q$  pairs of training examples,  $M$  layers and  $S_m$  neurons in each layer ( $m=0, 1, \dots, M$ ), we have the following feedforward equations for each pair of examples:

$$\underline{a}^0 = \underline{p}_q; \quad (4-16)$$

$$\underline{a}^{k+1} = \underline{f}^{k+1}(\underline{W}^{k+1} \underline{a}^k + \underline{b}^{k+1}) \quad k = 0, \dots, M-1; \quad (4-17)$$

$$\underline{a} = \underline{a}^M; \quad (4-18)$$

Where the  $\underline{W}^k$  ( $k=1, \dots, M$ ) are the connection weights and the  $\underline{b}^k$  are the bias vectors [1]. These parameters are arranged as  $\underline{\theta}$  with the following form:

$$\begin{aligned} \underline{\theta} = & [W^1(1,1); \dots; W^1(S_1,1); W^1(1,2); \dots; W^1(S_1,S_0); \\ & b^1(1); \dots; b^1(S_1); \dots; \\ & W^M(1,1); \dots; W^M(S_M,1); W^M(1,2); \dots; \\ & W^M(S_M, S_{M-1}); b^M(1); \dots; b^M(S_M)] \end{aligned} \quad (4-19)$$

Each element of Jacobian matrix can be written out by chain rule:

$$\frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial \theta_i} = \frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial n_j^k} \cdot \frac{\partial n_j^k}{\partial \theta_i} \quad (4-20)$$

where, as in [1], we define:

$$\delta_m^k(j) = \frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial n_j^k} \quad (4-21)$$

Then if the parameter  $\theta_i$  is a connection weight, we have:

$$\frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial \theta_i} = \frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial W^k(j, r)} = \delta_m^k(j) \cdot a^{k-1}(r) \quad (4-22)$$

which can be expressed in matrix form:

$$\frac{\partial \underline{a}(\underline{\theta}; \underline{p}_q)}{\partial W^k(:, r)} = \Delta^{kT} \cdot a^{k-1}(r) \quad (4-22')$$

where

$$\Delta^k = \begin{bmatrix} \delta_1^k(1) & \cdots & \delta_M^k(1) \\ \vdots & \vdots & \vdots \\ \delta_1^k(S_k) & \cdots & \delta_M^k(S_k) \end{bmatrix}$$

If the  $\theta_i$  is bias, then we have

$$\frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial \theta_i} = \frac{\partial a_m(\underline{\theta}; \underline{p}_q)}{\partial b^k(j)} = \delta_m^k(j) \quad (4-23)$$

which can be expressed in matrix form:

$$\frac{\partial \underline{a}(\underline{\theta}; \underline{p}_q)}{\partial \underline{b}^k} = \Delta^{kT} \quad (4-23')$$

The last layer has its sensitivity matrix as:

$$\underline{\Delta}^M = \dot{F}^M(\underline{n}^M) \quad (4-24)$$

which is different from the sensitivity vector in standard BP. The backpropagation formula is

$$\Delta^k = \dot{F}^k(\underline{n}^k) W^{k+1T} \Delta^{k+1} \quad k = M-1, M-2, \dots, 1 \quad (4-25)$$

where

$$\dot{F}^k(\underline{n}^k) = \begin{bmatrix} \dot{f}^k(n^k(1)) & 0 & \dots & 0 \\ 0 & \dot{f}^{kk}(n^k(2)) & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \dot{f}^k(n^k(S_k)) \end{bmatrix} \quad (4-26)$$

and

$$\dot{f}^k(n) = \frac{df^k(n)}{dn} \quad (4-27)$$

We summarize the steps for applying the Marquardt algorithm to an MPNN as follows:

- (1) forward calculate the actual output  $a$  by (4-16) - (4-18);
- (2) backward calculate all sensitivities by (4-24) and (4-25) and their associated terms;
- (3) calculate each term in Jacobian matrix  $H$  by (4-22) and (4-23) or (4-22') and (4-23') to find  $H$ .
- (4) calculate  $\underline{\theta}$  using (4-15), where, the  $\mu$  is adjusted by means of the Marquardt algorithm described in Chapter 3. Then follow the procedure (1) to (3) above until  $|\delta\underline{\theta}| \leq \varepsilon$  and/or

the performance index (4-9) decreases to an acceptable value.

## CHAPTER V

### RECURSIVE ALGORITHMS FOR TRAINING NEURAL NETWORKS

In the last chapter, we derived batching algorithms for training MPNNs using the Gauss-Newton (GN) method and the Marquardt method (MBP). This chapter will propose two recursive algorithms for training MPNNs, recursive Gauss-Newton backpropagation (RGN) and recursive Marquardt Backpropagation (RMBP).

Given a set of examples  $\{(p_q, t_q), q=1, \dots, Q\}$ , which are generated by a mapping  $f(p_q)=t_q$ , an MPNN can be trained to match the set of examples by some batching algorithm, e.g. GN or MBP. Furthermore, if the MPNN is trained well, it should capture the "general features" in the training data, so that it can predict values of  $f$  from new, previously unseen, pattern values,  $p$ .

GN and MBP have superior computational efficiency. The main problem with them is that updating the weights of an MPNN requires a complete data set. In some kinds of applications, e.g. adaptive control and signal processing, the data are generated by a dynamic system and may be sequentially presented to an MPNN. In these situations, recursive algorithms are needed. One of the ways to develop recursive algorithms is to start from some batching algorithms and then make them "recursive" based on some reasonable assumptions. This is the approach we follow in this

chapter.

In this chapter, we will derive three algorithms. First in section 5.1, we will introduce a new form of the Gauss-Newton method, calling it the quasi-recursive GN algorithm, which is derived by means of ILS [12]. Secondly, RGN will be derived in section 5.2 based on some assumptions (these assumptions are same as those used in developing the on-line Maximum Likelihood estimation techniques [18]). The relationship between RGN and the quasi-recursive GN algorithm will be pointed out. Finally, we will derive RMBP in section 5.3, which improves on RGN.

### 5.1 The Quasi-recursive Gauss-Newton Method

We will derive the quasi-recursive Gauss-Newton algorithm by means of ILS from the batching GN algorithm (4-6) in this section. This algorithm may alleviate the storage burden associated with the batching GN. Let us consider a single-input/single-output MPNN, as in (4-1). Assume that we have obtained the first  $k$  pairs of examples from  $Q$  pairs of training data and have obtained  $\delta \underline{\theta}_k$  by means of (4-6):

$$\delta \underline{t}_k = J_k \delta \underline{\theta} + \underline{v}'_k \quad (5-1)$$

where



$$J_k = \begin{bmatrix} \frac{\partial a(\underline{\theta}; p_1)}{\partial \theta_1} & \frac{\partial a(\underline{\theta}; p_1)}{\partial \theta_2} & \cdots & \frac{\partial a(\underline{\theta}; p_1)}{\partial \theta_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial a(\underline{\theta}; p_k)}{\partial \theta_1} & \frac{\partial a(\underline{\theta}; p_k)}{\partial \theta_2} & \cdots & \frac{\partial a(\underline{\theta}; p_k)}{\partial \theta_n} \end{bmatrix}_{\underline{\theta}^*}$$

We want to modify  $\delta \underline{\theta}_k$  to  $\delta \underline{\theta}_{k+1}$ , based on the (k+1)th pair of examples  $(p_{k+1}, t_{k+1})$ , which corresponds to:

$$\delta t_{k+1} = \nabla a^T(\underline{\theta}^*; p_{k+1}) \delta \underline{\theta}_k + v'_{k+1} \quad (5-2)$$

where  $\underline{\theta}^*$  is the initial guess for the optimal parameter vector.

Following the covariance form of the recursive LS algorithm [13], we get:

$$\delta \underline{\theta}_{k+1} = \delta \underline{\theta}_k + K_{k+1} \left\{ \delta t_k - \nabla a^T(\underline{\theta}^*; p_{k+1}) \delta \underline{\theta}_k \right\} \quad (5-3)$$

where

$$K_{k+1} = P_k \nabla a^T(\underline{\theta}^*; p_{k+1}) \left\{ \nabla a^T(\underline{\theta}^*; p_{k+1}) P_k \nabla a(\underline{\theta}^*; p_{k+1}) + I \right\}^{-1} \quad (5-4)$$

and

$$P_{k+1} = \left\{ I - K_{k+1} \nabla a(\underline{\theta}^*; p_{k+1}) \right\} P_k \quad (5-5)$$

Note that when we modify  $\delta \underline{\theta}$  sequentially, the nominal value

$\underline{\theta}^*$  is not changed until all the examples are passed through (5-3) - (5-5). The procedure for finding the optimal  $\underline{\theta}$  is as follows:

(1) Initially choose a  $\underline{\theta}^*$  as an estimate of the optimal  $\underline{\theta}$ , and set:

$$P_0 = \alpha I \quad (5-6)$$

as well as

$$\delta \underline{\theta}_0 = \underline{0} \quad (5-7)$$

where  $\alpha$  is a very large number.

(2) Follow the sequence of (5-8) below and use (5-3), (5-4) and (5-5) to find a  $\delta \underline{\theta}$  for all examples.

$$P_k \rightarrow K_{k+1} \rightarrow \delta \underline{\theta}_{k+1} \rightarrow P_{k+1} \quad \text{for } k = 0, \dots, Q-1 \quad (5-8)$$

(3) Modify  $\underline{\theta}^*$  as:

$$\underline{\theta}^{*new} = \underline{\theta}^{*old} + \delta \underline{\theta}_Q \quad (5-9)$$

And repeat the procedure (1) - (3) until

$$|\delta \underline{\theta}| \leq \epsilon$$

is satisfied, where  $\epsilon$  is a predetermined termination value.

The quasi-recursive algorithm for training a multi-input/multi-

output MPNN has a similar form and can be derived in the same manner. Assuming that the examples have the form of  $\{\underline{p}_q, \underline{t}_q; q=1, \dots, Q\}$ , we start with:

$$\underline{t}_k = \underline{a}(\underline{\theta}; \underline{p}_k) + \underline{v}_k \quad (5-10)$$

Define

$$\begin{aligned} h_q &= J_a(\underline{p}_q) \Big|_{\underline{\theta}^*} \\ &= \left[ \begin{array}{cccc} \frac{\partial a_1(\underline{\theta}; \underline{p}_q)}{\partial \theta_1} & \frac{\partial a_1(\underline{\theta}; \underline{p}_q)}{\partial \theta_2} & \dots & \frac{\partial a_1(\underline{\theta}; \underline{p}_q)}{\partial \theta_n} \\ \frac{\partial a_{Sm}(\underline{\theta}; \underline{p}_q)}{\partial \theta_1} & \frac{\partial a_{Sm}(\underline{\theta}; \underline{p}_q)}{\partial \theta_2} & \dots & \frac{\partial a_{Sm}(\underline{\theta}; \underline{p}_q)}{\partial \theta_n} \end{array} \right]_{\underline{\theta}^*} \end{aligned} \quad (5-11)$$

where  $\underline{\theta}^*$  is the initial guess for the optimal parameter vector.

Considering the first  $k$  pairs of examples, we have

$$\begin{aligned} \delta \underline{t}_1 &= h_1 \delta \underline{\theta} + \underline{v}'_1 \\ \delta \underline{t}_2 &= h_2 \delta \underline{\theta} + \underline{v}'_2 \\ &\vdots \\ \delta \underline{t}_k &= h_k \delta \underline{\theta} + \underline{v}'_k \end{aligned} \quad (5-12)$$

Stacking the  $k$  equations into a super-vector equation as:

$$\delta \underline{T}_k = H_k \delta \underline{\theta} + \underline{V}'_k \quad (5-13)$$

The solution for (5-13) is (5-14), based on (4-6), which is derived in the last chapter:

$$\delta \underline{\theta}_k = [H_k^T H_k]^{-1} H_k^T \delta \underline{T}_k \quad (5-14)$$

The subscript  $k$  in the term  $\delta \underline{\theta}_k$  means that it is a result of the first  $k$  pairs of examples. We further consider the solution after the  $k+1^{\text{st}}$  data point is presented to the MPNN:

$$\delta \underline{t}_{k+1} = h_{k+1} \delta \underline{\theta} + \underline{v}'_{k+1} \quad (5-15)$$

If we stack (5-13) and (5-15) we get:

$$\delta \underline{T}_{k+1} = H_{k+1} \delta \underline{\theta} + \underline{V}'_{k+1} \quad (5-16)$$

where

$$\delta \underline{T}_{k+1} = \begin{bmatrix} \delta \underline{T}_k \\ \delta \underline{t}_{k+1} \end{bmatrix}, \quad H_{k+1} = \begin{bmatrix} H_k \\ h_{k+1} \end{bmatrix}, \quad \underline{V}'_{k+1} = \begin{bmatrix} \underline{V}'_k \\ \underline{v}'_{k+1} \end{bmatrix}$$

The solution for (5-16) is known as

$$\delta \underline{\theta}_{k+1} = [H_{k+1}^T H_{k+1}]^{-1} H_{k+1}^T \delta \underline{T}_{k+1} \quad (5-17)$$

Next we want to make (5-17) recursive. So we define

$$P_k = [H_k^T H_k]^{-1} \quad (5-18)$$

then

$$\begin{aligned} P_{k+1} &= [H_{k+1}^T H_{k+1}]^{-1} \\ &= \left\{ \begin{bmatrix} H_k \\ h_{k+1} \end{bmatrix}^T \begin{bmatrix} H_k \\ h_{k+1} \end{bmatrix} \right\}^{-1} \\ &= \left\{ \begin{bmatrix} H_k^T & h_{k+1}^T \end{bmatrix} \begin{bmatrix} H_k \\ h_{k+1} \end{bmatrix} \right\}^{-1} \\ &= \{H_k^T H_k + h_{k+1}^T h_{k+1}\}^{-1} \\ &= \{P_k^{-1} + h_{k+1}^T h_{k+1}\}^{-1} \end{aligned} \quad (5-19)$$

Therefore

$$P_k^{-1} = P_{k+1}^{-1} - h_{k+1}^T h_{k+1} \quad (5-20)$$

Put (5-18) into (5-14)

$$\delta \underline{\theta}_k = (H_k^T H_k)^{-1} H_k^T \delta \underline{T}_k = P_k H_k^T \delta \underline{T}_k \quad (5-21)$$

Then we have

$$H_k^T \delta \underline{T}_k = P_k^{-1} \delta \underline{\theta}_k \quad (5-22)$$

Further, we consider (5-17)

$$\begin{aligned}
\delta \underline{\theta}_{k+1} &= (\mathbf{H}_{k+1}^T \mathbf{H}_{k+1})^{-1} \mathbf{H}_{k+1}^T \delta \underline{\mathbf{T}}_{k+1} \\
&= \mathbf{P}_{k+1} \begin{bmatrix} \mathbf{H}_k \\ \mathbf{h}_{k+1} \end{bmatrix}^T \begin{bmatrix} \delta \underline{\mathbf{T}}_k \\ \delta \underline{t}_{k+1} \end{bmatrix} \\
&= \mathbf{P}_{k+1} [\mathbf{H}_k^T \delta \underline{\mathbf{T}}_k + \mathbf{h}_{k+1}^T \delta \underline{t}_{k+1}] \\
&= \mathbf{P}_{k+1} [\mathbf{P}_k^{-1} \delta \underline{\theta}_k + \mathbf{h}_{k+1}^T \delta \underline{t}_{k+1}] \\
&= \mathbf{P}_{k+1} \left[ \left\{ \mathbf{P}_{k+1}^{-1} - \mathbf{h}_{k+1}^T \mathbf{h}_{k+1} \right\} \delta \underline{\theta}_k \right. \\
&\quad \left. + \mathbf{h}_{k+1}^T \delta \underline{t}_{k+1} \right] \\
&= \delta \underline{\theta}_k + \mathbf{P}_{k+1} \mathbf{h}_{k+1}^T [\delta \underline{t}_{k+1} - \mathbf{h}_{k+1} \delta \underline{\theta}_k]
\end{aligned} \tag{5-23}$$

Consequently, we get the recursive solution:

$$\delta \underline{\theta}_{k+1} = \delta \underline{\theta}_k + \mathbf{K}_{k+1} [\delta \underline{t}_{k+1} - \mathbf{h}_{k+1} \delta \underline{\theta}_k] \tag{5-24}$$

where

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1} \mathbf{h}_{k+1}^T \tag{5-25}$$

$$\mathbf{P}_{k+1}^{-1} = \mathbf{P}_k^{-1} + \mathbf{h}_{k+1}^T \mathbf{h}_{k+1} \tag{5-26}$$

If the matrix inversion lemma is applied to (5-26), then we get:

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{h}_{k+1} [\mathbf{h}_{k+1} \mathbf{P}_k \mathbf{h}_{k+1}^T + \mathbf{I}_{S_m \times S_m}]^{-1} \mathbf{h}_{k+1}^T \mathbf{P}_k \tag{5-27}$$

Substituting (5-27) into (5-25), we get

$$\mathbf{K}_{k+1} = \mathbf{P}_k \mathbf{h}_{k+1}^T \left[ \mathbf{h}_{k+1} \mathbf{P}_k \mathbf{h}_{k+1}^T + \mathbf{I}_{S_m \times S_m} \right]^{-1} \quad (5-28)$$

Substituting (5-28) into (5-27), we get:

$$\mathbf{P}_{k+1} = [\mathbf{I} - \mathbf{K}_{k+1} \mathbf{h}_{k+1}] \mathbf{P}_k \quad (5-29)$$

The procedure is summarized as follows:

- (1) Initially choose a  $\underline{\theta}^*$  as an estimate of the optimal  $\underline{\theta}$ , and set

$$\mathbf{P}_0 = \alpha \mathbf{I}$$

as well as

$$\delta \underline{\theta}_0 = \underline{0}$$

where  $\alpha$  is a very large number.

- (2) Follow the sequence of (5-30) below and use (5-24), (5-28) and (5-29) to find a  $\delta \underline{\theta}$  for all examples.

$$\mathbf{h} \mathbf{P}_k \rightarrow \mathbf{K}_{k+1} \rightarrow \delta \underline{\theta}_{k+1} \rightarrow \mathbf{P}_{k+1} \quad \text{for } k = 0, \dots, Q-1 \quad (5-30)$$

- (3) Modify  $\underline{\theta}^*$  as:

$$\underline{\theta}^{* \text{new}} = \underline{\theta}^{* \text{old}} + \delta \underline{\theta}_Q \quad (5-31)$$

And repeat steps (1) - (3) until

$$|\delta\theta| \leq \varepsilon \quad (5-32)$$

is satisfied, where  $\varepsilon$  is a predetermined termination value.

In the quasi-recursive algorithm, all the examples are presented to an MPNN before  $\theta$  is updated. The quasi-recursive GN algorithm is different from the batching GN in the calculation of  $\delta\theta$ . The batching GN calculates  $\delta\theta$  in one step, while the quasi-recursive algorithm calculates  $\delta\theta$  in a series of  $Q$  steps ( $Q$  is the number of examples).

## 5.2 Recursive Gauss-Newton Method

In section 5.1, we derived the quasi-recursive GN. In this section, we will derive the recursive Gauss-Newton algorithm (RGN) for training MPNNs. For RGN, the parameter vector will be updated after each example is presented.

Before we begin to develop RGN, we have to point out that the performance index we used previously treats all data points in the example set equally. This kind of performance index is widely used in the neural network literature and may not be a good choice. A widely used performance index in recursive estimation [18] is of the form



$$S_q(\underline{\theta}) = \frac{1}{2} \sum_{s=1}^q \underline{v}_s^T W_s \underline{v}_s \quad (5-33)$$

where  $W_s$  is a weighting matrix which weights the each residual error at each data point differently. If  $W_s$  is selected as the identity matrix, the resulting performance index is the same as that we used previously.

We formulate our problem as follows. Consider again the regression equation:

$$\underline{t}_q = \underline{a}(\underline{\theta}, \underline{p}_q) + \underline{v}_q \quad (5-34)$$

where  $\underline{a}$  represents the MPNN output and

$$\underline{\theta} \in \mathbb{R}^{n \times 1}; \quad \underline{t}_q, \underline{v}_q \in \mathbb{R}^{S_m \times 1}; \quad \underline{p} \in \mathbb{R}^{S_0 \times 1}; \quad (q=1, \dots, Q)$$

we want to find the parameter vector  $\underline{\theta}$  which minimizes

$$S_q(\underline{\theta}) = \frac{1}{2} \sum_{s=1}^q \lambda^{q-s} \underline{v}_s^T W \underline{v}_s \quad (5-35)$$

where we have set  $W_s = \lambda^{q-s} W$ , where  $\lambda$  is called forgetting factor [18] which emphasizes recent data. Applying the Taylor series expansion with a nominal value of  $\underline{\theta}$ ,  $\underline{\theta}^*$ , to (5-35) leads to:

$$\begin{aligned}
S_q(\underline{\theta}) &\approx S_q(\underline{\theta}^*) + \nabla S_q^T(\underline{\theta}^*)(\underline{\theta} - \underline{\theta}^*) \\
&\quad + \frac{1}{2}(\underline{\theta} - \underline{\theta}^*)^T \nabla^2 S_q(\underline{\theta}^*)(\underline{\theta} - \underline{\theta}^*)
\end{aligned} \tag{5-36}$$

Hence, if we let  $\underline{\theta}^*$  be  $\underline{\theta}_{q-1}$ , then (5-36) is equal to (5-37):

$$\begin{aligned}
S_q(\underline{\theta}) &\approx S_q(\underline{\theta}_{q-1}) + \nabla S_q^T(\underline{\theta}_{q-1})(\underline{\theta} - \underline{\theta}_{q-1}) \\
&\quad + \frac{1}{2}(\underline{\theta} - \underline{\theta}_{q-1})^T \nabla^2 S_q(\underline{\theta}_{q-1})(\underline{\theta} - \underline{\theta}_{q-1})
\end{aligned} \tag{5-37}$$

We can minimize the right side of (5-37) and obtain:

$$\underline{\theta}_q - \underline{\theta}_{q-1} = -[\nabla^2 S_q(\underline{\theta}_{q-1})]^{-1} [\nabla S_q(\underline{\theta}_{q-1})] \tag{5-38}$$

which is the Gauss-Newton method. This is a batching algorithm, because  $\nabla^2 S_q(\underline{\theta}_{q-1})$  and  $\nabla S_q(\underline{\theta}_{q-1})$  are evaluated at  $\underline{\theta}_{q-1}$  for all the (q-1) examples.

Let us now derive a recursive algorithm. First, we can rewrite the performance index:

$$\begin{aligned}
S_q(\underline{\theta}) &= \frac{1}{2} \sum_{s=1}^q \lambda^{q-s} \underline{v}_s^T \mathbf{W} \underline{v}_s \\
&= \frac{1}{2} \lambda \sum_{s=1}^{q-1} \lambda^{q-1-s} \underline{v}_s^T \mathbf{W} \underline{v}_s + \frac{1}{2} \underline{v}_q^T \mathbf{W} \underline{v}_q \\
&= \lambda S_{q-1}(\underline{\theta}) + \frac{1}{2} \underline{v}_q^T \mathbf{W} \underline{v}_q
\end{aligned} \tag{5-39}$$

From (5-39), it is easy to obtain:

$$\begin{aligned}
\nabla S_q(\underline{\theta}) &= \nabla \left[ \lambda S_{q-1}(\underline{\theta}) + \frac{1}{2} \underline{v}_q^T \mathbf{W} \underline{v}_q \right] \\
&= \lambda \nabla [S_{q-1}(\underline{\theta})] + \frac{1}{2} \nabla [\underline{v}_q^T \mathbf{W} \underline{v}_q] \\
&= \lambda \nabla [S_{q-1}(\underline{\theta})] + J_v(\underline{p}_q) \mathbf{W} \underline{v}_q(\underline{\theta}) \\
&= \lambda \nabla S_{q-1}(\underline{\theta}) + J_v(\underline{p}_q) \mathbf{W} \underline{v}_q(\underline{\theta})
\end{aligned} \tag{5-40}$$

and

$$\begin{aligned}
\nabla^2 S_q(\underline{\theta}) &= \nabla [\nabla S_q(\underline{\theta})] \\
&= \nabla \left[ \lambda \nabla S_{q-1}(\underline{\theta}) + J_v(\underline{p}_q) \mathbf{W} \underline{v}_q(\underline{\theta}) \right] \\
&= \lambda \nabla^2 S_{q-1}(\underline{\theta}) + \nabla \left[ J_v(\underline{p}_q) \mathbf{W} \underline{v}_q(\underline{\theta}) \right] \\
&= \lambda \nabla^2 S_{q-1}(\underline{\theta}) + J_v(\underline{p}_q) \mathbf{W} J_v^T(\underline{p}_q) + \\
&\quad \sum_{i=1}^{Sm} \sum_{j=1}^{Sm} v_{i,q}(\underline{\theta}) W_{ij} \nabla^2 v_{j,q}(\underline{\theta}) \\
&= \lambda \nabla^2 S_{q-1}(\underline{\theta}) + J_v(\underline{p}_q) \mathbf{W} J_v^T(\underline{p}_q) + \mathbf{B}(q, \underline{\theta})
\end{aligned} \tag{5-41}$$

In (5-41), we define

$$\mathbf{B}(q, \underline{\theta}) = \sum_{i=1}^{Sm} \sum_{j=1}^{Sm} v_{j,q}(\underline{\theta}) W_{ij} \nabla^2 v_{j,q}(\underline{\theta})$$

Let us make three assumptions

$$(1) \quad \nabla S_{q-1}(\underline{\theta}_{q-1}) = 0 \quad (5-42)$$

(which means that  $\underline{\theta}_{q-1}$  is the minimum of  $S_{q-1}(\underline{\theta})$ ); and

$$(2) \quad \nabla^2 S_{q-1}(\underline{\theta}_{q-1}) = \nabla^2 S_{q-1}(\underline{\theta}_{q-2}) \quad (5-43)$$

(which means that the surface in the parameter space is smooth enough); and

$$(3) \quad B(q, \underline{\theta}) = 0 \quad (5-44)$$

(which means that the model produces small residuals). We substitute the assumptions (5-42) to (5-44) into (5-38):

$$\begin{aligned} \underline{\theta}_q - \underline{\theta}_{q-1} &= -[\nabla^2 S_q(\underline{\theta}_{q-1})]^{-1} [\nabla S_q(\underline{\theta}_{q-1})] \\ &= -\left[ \lambda \nabla^2 S_{q-1}(\underline{\theta}_{q-1}) + J_v(\underline{p}_q) W J_v(\underline{p}_q)^T \right]^{-1} \\ &\quad \cdot \left[ \lambda \nabla S_{q-1}(\underline{\theta}_{q-1}) + J_v^T(\underline{p}_q) W \underline{v}_q(\underline{\theta}_{q-1}) \right] \\ &= -\left[ \lambda \nabla^2 S_{q-1}(\underline{\theta}_{q-2}) + J_v(\underline{p}_q) W J_v(\underline{p}_q)^T \right]^{-1} \\ &\quad \cdot J_v^T(\underline{p}_q) W \underline{v}_q(\underline{\theta}_{q-1}) \end{aligned} \quad (5-45)$$

and define:

$$P_q = \nabla^2 S_q(\underline{\theta}_{q-1})^{-1}, \quad (5-46)$$

and

$$\phi_q = J_a(\underline{p}_q) \Big|_{\underline{\theta}_{q-1}} = -J_v(\underline{p}_q) \Big|_{\underline{\theta}_{q-1}} \quad (5-47)$$

then

$$P_{q-1} = \nabla^2 S_{q-1}(\underline{\theta}_{q-2})^{-1} \quad (5-48)$$

and consequently

$$\begin{aligned} P_q &= \nabla^2 S_q(\underline{\theta}_{q-1})^{-1} \\ &= \left[ \lambda \nabla^2 S_{q-1}(\underline{\theta}_{q-2}) + J_v(\underline{p}_q) \Big|_{\underline{\theta}_{q-1}} W J_v(\underline{p}_q) \Big|_{\underline{\theta}_{q-1}}^T \right]^{-1} \\ &= \left[ \lambda P_{q-1}^{-1} + \phi_q W \phi_q^T \right]^{-1} \end{aligned} \quad (5-49)$$

Applying the matrix inverse lemma to (5-49), we get

$$P_q = \frac{1}{\lambda} \left[ P_{q-1} - P_{q-1} \phi_q \left\{ \lambda W^{-1} + \phi_q^T P_{q-1} \phi_q \right\}^{-1} \phi_q^T P_{q-1} \right] \quad (5-50)$$

Therefore, we summarize (5-45) to (5-50) as follows:

$$\begin{aligned}
\underline{v}_q(\underline{\theta}_{q-1}) &= \underline{t}_q - \underline{a}(\underline{\theta}_{q-1}; \underline{p}_q) \\
\underline{P}_q &= \frac{1}{\lambda} \left[ \underline{P}_{q-1} - \underline{P}_{q-1} \underline{\phi}_q \left\{ \lambda \underline{W}^{-1} + \underline{\phi}_q^T \underline{P}_{q-1} \underline{\phi}_q \right\}^{-1} \underline{\phi}_q^T \underline{P}_{q-1} \right] \\
\underline{\theta}_q &= \underline{\theta}_{q-1} + \underline{P}_q \underline{\phi}_q \underline{W} \underline{v}_q(\underline{\theta}_{q-1})
\end{aligned} \tag{5-51}$$

This is the recursive Gauss-Newton method.

An alternative form of (5-51), which is used in our implementation of RGN is given in (5-52) below:

$$\begin{aligned}
\underline{v}_q(\underline{\theta}_{q-1}) &= \underline{t}_q - \underline{a}(\underline{\theta}_{q-1}; \underline{p}_q) \\
\underline{K}_q &= \underline{P}_{q-1} \underline{\phi}_q \left\{ \lambda \underline{W}^{-1} + \underline{\phi}_q^T \underline{P}_{q-1} \underline{\phi}_q \right\}^{-1} \\
\underline{\theta}_q &= \underline{\theta}_{q-1} + \underline{K}_q \underline{v}_q(\underline{\theta}_{q-1}) \\
\underline{P}_q &= \frac{1}{\lambda} \left[ \underline{P}_{q-1} - \underline{P}_{q-1} \underline{\phi}_q \left\{ \lambda \underline{W}^{-1} + \underline{\phi}_q^T \underline{P}_{q-1} \underline{\phi}_q \right\}^{-1} \underline{\phi}_q^T \underline{P}_{q-1} \right]
\end{aligned} \tag{5-52}$$

In (5-51) and (5-52),  $\lambda$  can be time-variant. This facilitates flexibility of the algorithm. The term  $\underline{W}$  may be used to weight each MPNN output when a multi-output MPNN is used.

The difference between RGN and the quasi-recursive GN algorithm is whether the weights are updated or not after each data pair is presented. RGN updates the weights after each data pair is presented, while the quasi-recursive GN algorithm updates the weights after all pairs of data are presented. Figures 10 and 11 are depicted this difference.

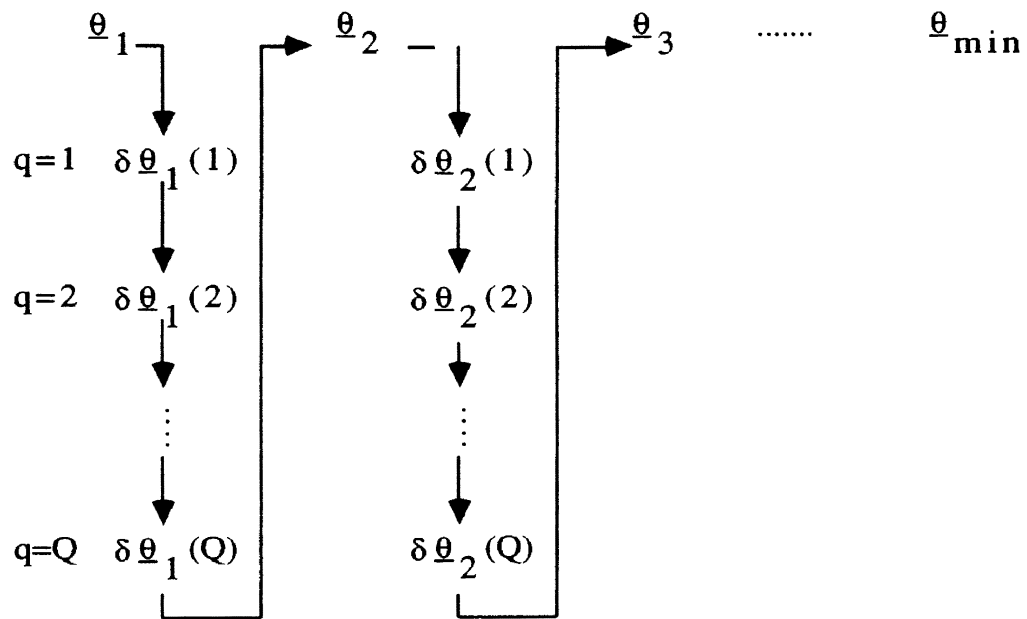


Figure. 10 Parameter Updating in Quasi-recursive GN

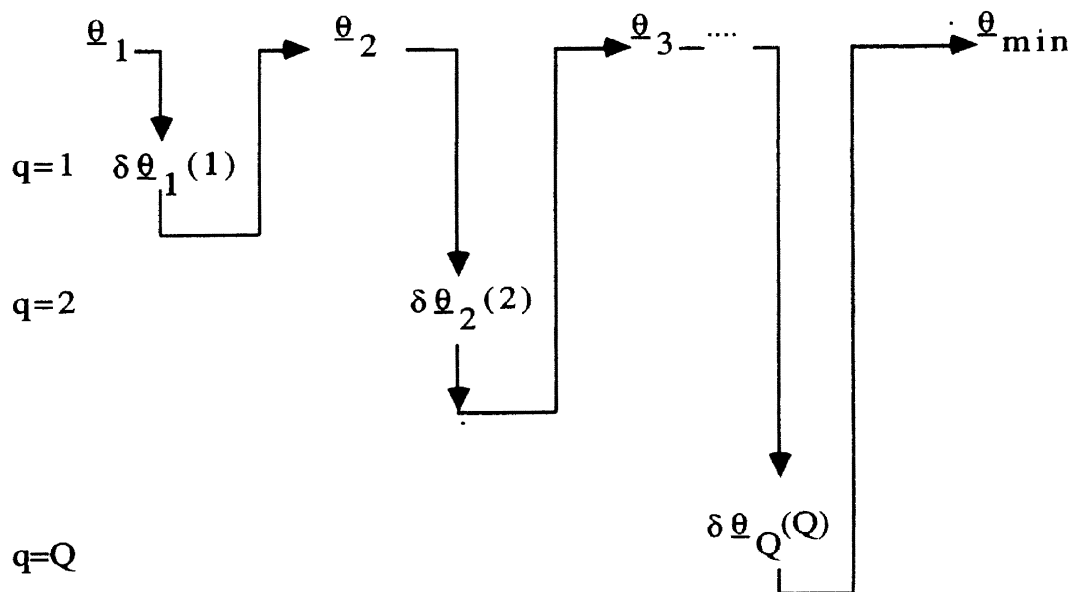


Figure. 11 Parameter Updating in RGN.

### 5.3 Recursive Marquardt Method

This section will derive the recursive Marquardt backpropagation algorithm (RMBP) for training MPNNs. In last chapter, we derived batching MBP to fix the ill-conditioning problem which may occur when we invert the Hessian matrix in batching GN. In the last section, RGN was derived and the possibility of an ill-conditioned Hessian matrix may also exist in RGN. We treat it as we did for batching GN in the last chapter.

Observing (5-49), we find that if the term on the right hand side of (5-49) is ill-conditioned at some step  $q$ , then the matrix may not be invertible. Therefore, we can apply the Marquardt  $\mu$  to (5-49)

$$P_q^{-1} = \lambda P_{q-1}^{-1} + \phi_q W \phi_q^T + \mu_q I \quad (5-54)$$

to ensure that  $P_q^{-1}$  is always positive definite. In (5-54),  $\mu_q$  is a small positive number and may change with  $q$ . After adding this  $\mu_q$ , the RMBP can be derived and written as:

$$\begin{aligned} \underline{v}_q(\underline{\theta}_{q-1}) &= \underline{t}_q - \underline{a}(\underline{\theta}_{q-1}; \underline{p}_q) \\ P_q^{-1} &= \lambda P_{q-1}^{-1} + \phi_q W \phi_q^T + \mu_q I \\ \underline{\theta}_q &= \underline{\theta}_{q-1} + P_q \phi_q W \underline{v}_q(\underline{\theta}_{q-1}) \end{aligned} \quad (5-55)$$

In practice, especially in real-time calculations, we may not know how to adjust  $\mu_q$  in an optimal way. If we set  $\mu_q$  in (5-55) as a constant  $\mu$  (e.g.  $\mu=10^{-3}$ ),  $P_q^{-1}$  has the form



$$P_q^{-1} = \lambda P_{q-1}^{-1} + \phi_q W \phi_q^T + \frac{1-\lambda^q}{1-\lambda} \mu I, \quad (5-56)$$

after  $q$  pieces of data have been presented. We explain (5-56) next.

For  $q = 1, 2$  and  $\lambda$  is a constant,  $P_1^{-1}$  and  $P_2^{-1}$  have the forms

$$\begin{aligned} P_1^{-1} &= \lambda P_0^{-1} + \phi_1 W \phi_1^T + \mu_1 I; \\ P_2^{-1} &= \lambda P_1^{-1} + \phi_2 W \phi_2^T + \mu_2 I \\ &= \lambda (\lambda P_0^{-1} + \phi_1 W \phi_1^T + \mu_1 I) + \phi_2 W \phi_2^T + \mu_2 I \\ &= \lambda (\lambda P_0^{-1} + \phi_1 W \phi_1^T) + \phi_2 W \phi_2^T + \lambda \mu_1 I + \mu_2 I. \end{aligned}$$

For  $q$  pieces of data, therefore,  $P_q^{-1}$  has the form

$$\begin{aligned} P_q^{-1} &= \lambda^q P_0^{-1} + \dots + \lambda \phi_{q-1} W \phi_{q-1}^T + \phi_q W \phi_q^T \\ &\quad + (\lambda^{q-1} \mu_1 + \dots + \lambda \mu_{q-1} + \mu_q) I. \end{aligned}$$

The term  $\mu_q^{\text{tot}} = \lambda^{q-1} \mu_1 + \dots + \lambda \mu_{q-1} + \mu_q$  is the accumulated effect of  $\mu_q$  on  $P_q^{-1}$  for the  $q$  pieces of data. If  $\mu_q = \mu$ , a constant number,

$$\mu_q^{\text{tot}} = \frac{1-\lambda^q}{1-\lambda} \mu,$$

which appeared in (5-56). In this case the accumulated effect of  $\mu_q$  on  $P_q^{-1}$  is adding an identity matrix with a coefficient:

$$\frac{1-\lambda^q}{1-\lambda}\mu \quad (5-57)$$

which changes with  $q$ . Note that (5-57) increases with  $q$  but is bounded by  $\mu/(1-\lambda)$  if  $\lambda < 1$ .

If we need  $\mu_q^{\text{tot}}$  to be constant, we have to set

$$\mu_q^{\text{tot}} = \mu_{q-1}^{\text{tot}}$$

$$\text{or } \lambda^{q-1}\mu_1 + \cdots + \lambda\mu_{q-1} + \mu_q = \lambda^{q-2}\mu_1 + \cdots + \lambda\mu_{q-2} + \mu_{q-1}. \quad (5-57')$$

Therefore, if  $q = 2$ ,  $\lambda\mu_1 + \mu_2 = \mu_1$ . That is

$$\mu_2 = (1-\lambda)\mu_1.$$

If  $q > 2$ , (5-57') can be written as

$$\mu_q = (1-\lambda)(\lambda^{q-2}\mu_1 + \cdots + \lambda\mu_{q-2} + \mu_{q-1}). \quad (5-57'')$$

Furthermore, for  $\mu_{q+1}$ , we have

$$\mu_{q+1} = (1-\lambda)(\lambda^{q-1}\mu_1 + \cdots + \lambda^2\mu_{q-2} + \lambda\mu_{q-1} + \mu_q). \quad (5-57''')$$

From (5-57'') and (5-57'''), we obtain

$$\mu_{q+1} - \lambda\mu_q = (1-\lambda)\mu_q \Leftrightarrow \mu_{q+1} = \mu_q \quad (q \geq 2)$$

Therefore, we have following formula to determine  $\mu_q$  when we need  $\mu_q^{\text{tot}}$  to be constant:

$$\mu_q = (1 - \lambda)\mu_1; \quad q = 2, 3, \dots \quad (5-58)$$

where  $\mu_1$  is assigned to whatever value is desired for  $\mu_q^{\text{tot}}$ .

We must point out that RMBP (5-55) requires a matrix inverse each time a piece of new data is presented. This is a disadvantage. The advantage is that we do not need to worry about the problem of having a singular matrix.

The operation of inverting the Hessian in (5-55) requires intensive computation. A version of RMBP without such inversion is proposed based on the idea of Levenberg and Marquardt. In Chapter 2, we showed that the weight update by means of the steepest descent method have the form of

$$\Delta \underline{\theta}_{\text{SD}} = \phi_q \mathbf{W} \underline{v}_q(\underline{\theta}_{q-1}). \quad (5-59)$$

We call the search direction generated from (5-59) as the steepest descent direction. In (5-51), the weight update takes the form of

$$\Delta \underline{\theta}_{\text{GN}} = P_q \phi_q \mathbf{W} \underline{v}_q(\underline{\theta}_{q-1}). \quad (5-60)$$

We call the search direction generated from (5-60) as the Gauss-Newton direction. If the search along the Gauss-Newton direction fails to approach the minimum because of such reasons as an ill-

conditioned Hessian matrix, we may try some directions other than the Gauss-Newton direction. The search direction from the Marquardt method is in between Gauss-Newton and steepest descent. From (5-55), we can see that the weight update for the Marquardt method is

$$\Delta \underline{\theta}_{\text{MD}} = \left[ \lambda P_{q-1}^{-1} + \phi_q W \phi_q^T + \mu I \right]^{-1} \phi_q W \underline{v}_q(\underline{\theta}_{q-1}). \quad (5-61)$$

The Marquardt direction becomes the steepest descent direction as  $\mu \rightarrow \infty$

$$\Delta \underline{\theta}_{\text{MD}} = \frac{1}{\mu} \phi_q W \underline{v}_q(\underline{\theta}_{q-1}) = \frac{1}{\mu} \Delta \underline{\theta}_{\text{SD}}. \quad (5-62)$$

or the Gauss-Newton direction as  $\mu \rightarrow 0$

$$\begin{aligned} \Delta \underline{\theta}_{\text{MD}} &= \left[ \lambda P_{q-1}^{-1} + \phi_q W \phi_q^T \right]^{-1} \phi_q W \underline{v}_q(\underline{\theta}_{q-1}) \\ &= P_q \phi_q W \underline{v}_q(\underline{\theta}_{q-1}) = \Delta \underline{\theta}_{\text{GN}}. \end{aligned} \quad (5-63)$$

When we search for a minimum by means of the Marquardt method, we adjust  $\mu$  to determine a search direction which changes between the Gauss-Newton and the steepest descent direction. We adjust  $\mu$  in order to maintain the current norm of the error (between network output and desired output) to be less than the previous norm of error.

According to the idea of Levenberg and Marquardt, we increase

the  $\mu$  when the norm of the error increases, and decrease  $\mu$  when the norm of the error decreases (see [1] and Chapter 4 for detail). The Marquardt direction is close to the Gauss-Newton direction when  $\mu$  decreases, as seen in (5-63), and is close to the steepest descent direction when  $\mu$  increases, as seen in (5-62). The Marquardt direction is the combination of the two search directions.

A recursive Marquardt algorithm without matrix inversion is an approximation of the exact Marquardt algorithm (5-55) and should possess the same features. We propose the weight update scheme (i.e. the search direction) for a recursive Marquardt algorithm without matrix inversion as follows:

$$\Delta \underline{\theta}_{\text{MD}} = \frac{1}{(1 + \mu)^2} \Delta \underline{\theta}_{\text{GN}} + \frac{\mu}{(1 + \mu)^2} \Delta \underline{\theta}_{\text{SD}}. \quad (6-64)$$

As  $\mu \rightarrow \infty$ ,

$$\Delta \underline{\theta}_{\text{MD}} = \frac{1}{\mu} \Delta \underline{\theta}_{\text{SD}};$$

and as  $\mu \rightarrow 0$ ,

$$\Delta \underline{\theta}_{\text{MD}} = \Delta \underline{\theta}_{\text{GN}},$$

just as with the standard Marquardt method. Substituting (6-59) and (6-60) into (6-64) leads to a version of RMBP without matrix inversion as follows:

$$\begin{aligned}
\underline{v}_q(\underline{\theta}_{q-1}) &= \underline{t}_q - \underline{a}(\underline{\theta}_{q-1}; \underline{p}_q) \\
\mu_q &= \begin{cases} \mu_{q-1} \cdot \beta & \|\underline{v}_q(\underline{\theta}_{q-1})\| < \|\underline{v}_{q-1}(\underline{\theta}_{q-1})\| \\ \mu_{q-1} / \beta & \|\underline{v}_q(\underline{\theta}_{q-1})\| \geq \|\underline{v}_{q-1}(\underline{\theta}_{q-1})\| \end{cases} \\
K_q &= \frac{1}{(\mu+1)^2} P_{q-1} \phi_q \left\{ \lambda W^{-1} + \phi_q^T P_{q-1} \phi_q \right\}^{-1} + \frac{\mu}{(\mu+1)^2} \phi_q W \\
\underline{\theta}_q &= \underline{\theta}_{q-1} + K_q \underline{v}_q(\underline{\theta}_{q-1}) \\
P_q &= \frac{1}{\lambda} \left[ P_{q-1} - P_{q-1} \phi_q \left\{ \lambda W^{-1} + \phi_q^T P_{q-1} \phi_q \right\}^{-1} \phi_q^T P_{q-1} \right]
\end{aligned} \tag{5-65}$$

where,  $\beta$  is a constant. The initial  $\mu_0$  takes on a value in the range of [0.001 0.01],  $\beta$  in the range of [0.9 0.99].

#### 5.4 Summary

In this chapter, we derived three algorithms: quasi-recursive GN, RGN and RMBP. The quasi-recursive algorithm is really a batching algorithm. RGN is a truly recursive procedure which is an approximation of quasi-recursive GN algorithm. RMBP avoids the possibility of ill-conditioning which is found in RGN. In the next chapter, we will use RGN and RMBP to approximate two functions. We will investigate their properties and compare them with standard BP introduced in chapter 2.

## CHAPTER VI

### APPLICATIONS OF RGN AND RMBP

#### 6.1 Introduction

In the last chapter, we derived two recursive learning algorithms, the Recursive Gauss-Newton method (RGN) and the Recursive Marquardt method (MBP). In this chapter, we will use the two recursive algorithms to train MPNNs to approximate nonlinear functions. In Chapter 2, we pointed out that an MPNN with sufficient hidden neurons could approximate any nonlinear function if it was trained by some learning algorithms, such as backpropagation (BP), adaptive learning rate BP and conjugate gradient BP. At that time we trained MPNNs using batching algorithms.

In this chapter, we will train MPNNs using recursive algorithms, RGN and RMBP. We will select two functions to form test problems. (The two functions were also used in [1]. The purpose of selecting the two functions is to provide results which can be compared with those obtained in [1]). In addition, we will investigate the effects of the design variables involved in RGN and RMBP (such as  $P_0$ ,  $\lambda_0$ ,  $\underline{\theta}_0$  and  $\mu$ ) on the convergence rate of the learning process. We will give some details of the implementation of the two algorithms.

In section 6-2, we will define two functions to form test

problems, and we will describe the stopping rule which will be used to determine the convergence. In section 6-3, we will use RGN and RMBP to train MPNNs to approximate the first function defined in section 6-2, and will investigate the effects of the design variables involved in RGN and RMBP (such as  $P_0$ ,  $\lambda_0$  and  $\theta_0$ ) on the convergence rate of the learning process. What we do for the first nonlinear function in section 6-3 will be repeated for the second nonlinear function in section 6-4. Finally, in section 6-5, we will summarize the test results obtained in sections 6-3 and 6-4, and give some conclusions about RGN and RMBP based on our tests.

## 6.2 Test Problems

In this section, we will define two functions to form test problems and will define the stopping rule which will be used to determine the convergence. The two functions were used in [1] to test BP, CGBP and MBP. The purpose of selecting these functions is to provide test results which can be compared with those obtained in [1]. In our testing process, test problems are formed by computing a set of discrete points from the functions of interest. MPNNs are trained to approximate the functions at these points.

The first nonlinear function is a sine wave:

$$y = \frac{1}{2} + \frac{1}{4} \sin 3\pi x; \quad x \in [-1, 1]. \quad (6-1)$$

The first test problem is defined as follows:

- (1)  $x$  takes on values from -1 to 1 with steps of 0.05;



- (2)  $y$  takes on values calculated from (6-1) at those  $x$  values defined in (1).

The first test problem and the first function (6-1) are depicted in Figure 12, where each \* indicates a test example.

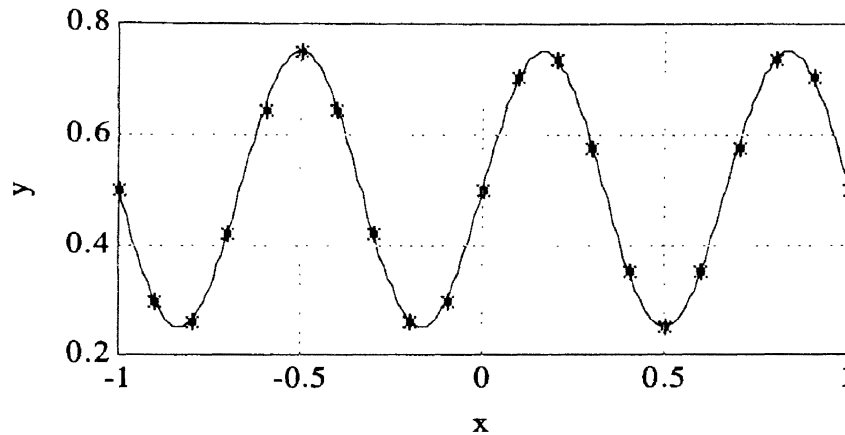


Figure 12. The First Test Problem

The second function is a square wave defined as follows:

$$y = \begin{cases} -1 & \{x \in (-1, 0] \cup (1, 2]\} \\ 0 & \{x \in [-3, -2] \cup (1, 3]\} \\ 1 & \{x \in (-2, -1] \cup (0, 1]\} \end{cases} \quad (6-2)$$

The second test problem is defined as follows:

- (1)  $x$  takes on values from -3 to 2.9 with steps of 0.1;  
 (2)  $y$  takes on values calculated from (6-2) at those  $x$  values defined above.

The second test problem and the second function (6-2) are depicted in Figure 13, where each \* indicates a test example.

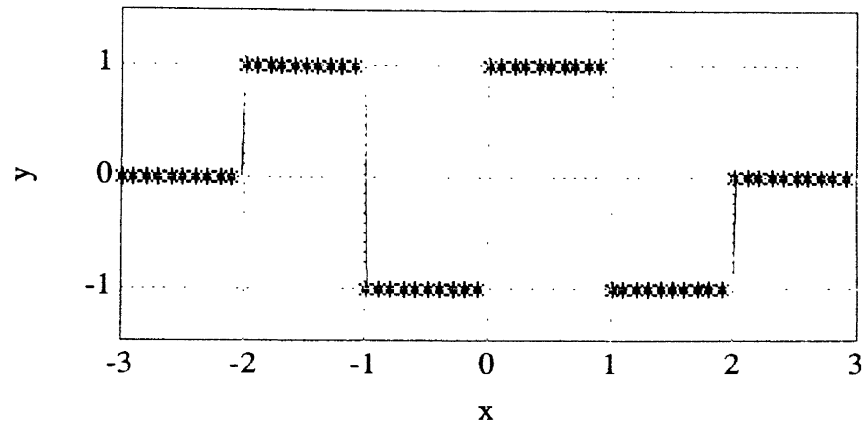


Figure 13. The Second Test Problem

A stopping rule is used to determine when the training process terminates. Training stops when the SSE (SSE is the sum of squares of errors between the desired outputs and MPNN outputs) is less than a prespecified value, or the number of iterations has exceeded some maximum number. We set the target SSE to 0.02, which is same as that used in [1], and the maximum iteration number to 800. If the SSE of a learning process does not reach 0.02 within 800 iterations, we will state that the process is not convergent.

In order to make the test results more reliable, we repeated the same test with 10 different seeds for the random number generator, which is used to create the initial weights and offsets, and averaged the results obtained. We observed that some particular seeds produced very good test results, while others did not. Averaging the test results for different seeds will avoid misleading results. We set the seeds here to the same values as those used in [1]

We will train MPNNs to approximate the two nonlinear

functions by means of RGN and RMBP in the next two sections.

### 6.3 Test Procedures/Results For Test Problem #1

#### 6.3.1 Test Results With RGN

In section 6.2, we defined the first test problem. In this section, we will apply RGN (5-52), RMBP with inversion (5-55) and RMBP without inversion (5-59) to the first test problem, and will investigate the effects of the design variables on the convergence rate.

First, we will apply RGN (5-52) to the first test problem and will investigate the effects of the design variables  $P_0$ ,  $\lambda_0$  and  $\theta_0$  on the convergence rate. We use an MPNN with one hidden layer and one output terminal. There are 15 hyperbolic tangent type neurons in the hidden layer. We use this kind of MPNN in all the tests. Because the MPNN has one output terminal, the design variable  $W$  in (5-52) is set to 1. The RGN used to train the MPNN is summarized in (6-3) below.

$$\begin{aligned}
a(\underline{\theta}_{q-1}; P_q) &= W_{q-1}^2 \underline{f}^1(W_{q-1}^1 \underline{P}_q + \underline{b}_{q-1}^1) + \underline{b}_{q-1}^2; \\
\phi_q &= [\Delta^1 P_q; \Delta^1; a; 1]; \\
\underline{\theta}_q &= \underline{\theta}_{q-1} + \frac{P_{q-1} \phi_q}{\lambda_q + \phi_q^T P_{q-1} \phi_q} [t_q - a(\underline{\theta}_{q-1}; P_q)]; \\
P_q &= \frac{1}{\lambda_q} \left[ P_{q-1} - \frac{P_{q-1} \phi_q \phi_q^T P_{q-1}}{\lambda_q + \phi_q^T P_{q-1} \phi_q} \right]; \\
\lambda_q &= 0.99 * \lambda_{q-1} + 0.01.
\end{aligned} \tag{6-3}$$

where

$$\Delta^1 = \begin{bmatrix} 1 - f_1^1 \cdot f_1^1 & 0 & \cdots & 0 \\ 0 & 1 - f_2^1 \cdot f_2^1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 - f_{S1}^1 \cdot f_{S1}^1 \end{bmatrix} \cdot W_{q-1}^2{}^T$$

In (6-3),  $W_{q-1}^1$ ,  $\underline{b}_{q-1}^1$ ,  $W_{q-1}^2$  and  $\underline{b}_{q-1}^2$  are the elements of  $\underline{\theta}_{q-1}$  (Refer to (4-19) in Chapter 4 for detail).

Table 6-1 (a) shows the test results obtained by means of (6-3). Each item in the table is an average value for ten different sets of initial random weights. We set the initial values of  $\underline{\theta}_0$  by means of two methods. One is suggested by Nguyen and Widrow (NW for short) [3]. The other method sets the initial values to small random numbers, for example, to random values in a range from -0.001 through 0.001. When we test the effects of  $P_0$ , we change  $P_0$  but set  $\lambda_0$  to 0.88 and initialize  $\underline{\theta}_0$  by the NW method. When we test the

effects of  $\lambda_0$ , we change  $\lambda_0$  but set  $P_0$  to  $10^3$  and initialize  $\underline{\theta}_0$  by the NW method. When we test the effects of  $\underline{\theta}_0$ , we initialize  $\underline{\theta}_0$  by either the NW method or a set of random values, but set  $\lambda_0$  to 0.88 and  $P_0$  to  $10^3$ . We mark \* in the row where we obtained failure records (some particular seeds do not lead to convergence). For other tables listing other test results, we will use the same conventions.

TABLE 6-1 (A)  
TEST RESULTS FOR RGN (6-3) FOR PROB. #1

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$	$10^5$	234.3*	1.2645e+8	0.01923
$(\lambda_0 = 0.88,$	$10^3$	224.2	1.2110e+8	0.01942
$\underline{\theta}_0$ -NW)	10	467.5*	2.5251e+8	0.01969
	0.95	226.1*	1.2213e+8	0.01917
$\lambda_0$	0.92	217.95*	1.1772e+8	0.01942
$(P_0 = 10^3,$	0.90	219.2	1.1842e+8	0.01930
$\underline{\theta}_0$ -NW)	0.88	224.2	1.2110e+8	0.01942
	0.86	225.4	1.2175e+8	0.01917
$\underline{\theta}_0$ ( $\lambda_0=0.88,$	NW	224.2	1.2110e+8	0.01942
$P_0 = 10^3)$	Small rand.	209.7	1.2645e+8	0.01890

The test results shown in Table 6-1 (a) reveal that RGN is comparable with variable learning rate BP in terms of the total flops (the total flops for variable learning rate BP is  $8.42e+7$ , refer to [1]).

When testing variable learning rate BP in [1], a batching algorithm was used, but RGN is a recursive algorithm.

For problem #1, the standard RGN algorithm (6-3) works well. For the next test problem (approximating a square wave (6-2)), the standard RGN algorithm cannot deal with ill-conditioned  $P_q$  matrices after tens of iterations. To circumvent this ill-conditioning problem, we can use RMBP. Another method is to add a small identity matrix to  $P_q$  in (6-3) so that an improved RGN is proposed as (6-4):

$$\begin{aligned}
 a(\underline{\theta}_{q-1}; P_q) &= W_{q-1}^2 f^1(W_{q-1}^1 \underline{P}_q + \underline{b}_{q-1}^1) + \underline{b}_{q-1}^2; \\
 \phi_q &= [\Delta^1 P_q; \Delta^1; a; 1]; \\
 \underline{\theta}_q &= \underline{\theta}_{q-1} + \frac{P_{q-1} \phi_q}{\lambda_q + \phi_q^T P_{q-1} \phi_q} [t_q - a(\underline{\theta}_{q-1}; P_q)]; \\
 P_q &= \frac{1}{\lambda_q} \left[ P_{q-1} - \frac{P_{q-1} \phi_q \phi_q^T P_{q-1}}{\lambda_q + \phi_q^T P_{q-1} \phi_q} \right] + R_q; \quad R_q = 0.1 \cdot I \\
 \lambda_q &= 0.99 * \lambda_{q-1} + 0.01
 \end{aligned} \tag{6-4}$$

where  $\Delta^1$  has the same form as in (6-3). We give the test results for algorithm (6-4) in Table 6-1 (b).

From Table 6-1 (b), we note that the improved version of RGN is comparable with CGBP in terms of the total flops (the total flops for CGBP is  $1.75e+7$ , refer to [1]). If the design variables  $\lambda_0 = 0.88$  or  $0.90$ ,  $P_0 = 10^3$  and  $\underline{\theta}_0$  is initialized by small random values, the total flops and the number of iterations have smaller values, i.e. the convergence is faster. It is also noted that the  $R_q$  in (6-4) makes the

TABLE 6-1 (B)  
TEST RESULTS FOR MODIFIED RGN (6-4) FOR PROB. #1

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$	$10^5$	33.4*	1.8040e+7	0.01923
$(\lambda_0=0.88,$ $\theta_0\text{-NW})$	$10^3$	39.1	2.1119e+7	0.01872
	10	207.833*	1.1226e+8	0.01765
	0.95	102.3	5.5256e+7	0.01903
$\lambda_0$	0.92	99.875*	5.3946e+7	0.01874
$(P_0 = 10^3,$ $\theta_0\text{-NW})$	0.90	23.4	1.2639e+7	0.01767
	0.88	39.1	2.1119e+7	0.01872
	0.86	46.6	2.5170e+7	0.01740
$\theta_0 (\lambda_0=0.88,$ $P_0 = 10^3)$	NW	39.1	2.1119e+7	0.01872
	Small rand.	37.5	2.0255e+7	0.01867

convergence faster.

### 6.3.2 Test Results for RMBP With Matrix Inversion

Next, we apply RMBP with matrix inversion to the first test problem and investigate the effects of the design variables  $P_0$ ,  $\lambda_0$ ,  $\theta_0$  and  $\mu$  on the convergence rate. We summarize the RMBP algorithm with matrix inversion as follows:

$$\begin{aligned}
a(\underline{\theta}_{q-1}; p_q) &= W_{q-1}^2 \underline{f}^1(W_{q-1}^1 \underline{p}_q + \underline{b}_{q-1}^1) + \underline{b}_{q-1}^2; \\
\phi_q &= [\Delta^1 p_q; \Delta^1; a; 1]; \\
P_q^{-1} &= \lambda_q P_{q-1}^{-1} + \phi_q \phi_q^T + \mu_q I; \\
\underline{\theta}_q &= \underline{\theta}_{q-1} + P_q \phi_q [t_q - a(\underline{\theta}_{q-1}; p_q)]; \\
\lambda_q &= 0.99 * \lambda_{q-1} + 0.01.
\end{aligned} \tag{6-5}$$

where where  $\Delta^1$  has the same form as in (6-3),  $\mu_q$  may take a constant number, for example 0.001, or a number calculated by

$$\mu_q = (1 - \lambda) \mu_1; \quad q = 2, 3, \dots \tag{6-6}$$

The test results are listed in Table 6-2. When we test the RMBP algorithm with matrix inversion, we investigate the effect of  $\mu$  on the convergence rate. From Table 6-2, we find that RMBP with matrix inversion needs more total flops to converge (SSE is less than 0.02 within 800 iterations) than RGN, even though it needs fewer iterations. The matrix inversion involved in (6-5) requires intensive flops. If the design variables  $\lambda_0 = 0.88$  or  $0.90$ , and  $P_0 = 10^3$ ,  $\mu = 0.001$  and  $\underline{\theta}_0$  is initialized by small random values, the total flops and iterations have smaller values, i.e. the convergence is faster.



TABLE 6-2  
TEST RESULTS FOR RMBP WITH INVERSE FOR PROB. #1

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$ ( $\lambda_0=0.88,$	$10^5$	63.1*	2.6968e+9	0.01923
$\theta_0$ -NW)	$10^3$	29.3	1.2522e+9	0.01901
$\mu=1e-3$ )	10	77.4	3.3080e+9	0.01989
	0.95	48.2	2.0600e+9	0.01991
$\lambda_0$ ( $P_0=10^3,$	0.92	51.7	2.2096e+9	0.01894
$\theta_0$ -NW	0.90	17.6	7.5220e+8	0.01813
$\mu=1e-3$ )	0.88	29.3	1.2522e+9	0.01901
	0.86	33.8	1.4446e+9	0.01993
$\theta_0$ ( $P_0=10^3$	NW	29.3	1.2522e+9	0.01901
$\lambda_0=.88,\mu=1e-3$ )	Small rand.	28.5	1.2180e+9	0.01952
$\mu$ ( $P_0=10^3$	fixed	29.3	1.2522e+9	0.01901
$\lambda_0=.88,\theta_0$ NW)	set by (6-6)	29.8	1.2530e+9	0.01910

### 6.3.3 Test Results for RMBP Without Matrix Inversion

In this section, we apply the RMBP algorithm which does not require matrix inversion to the first test problem and investigate the effects of the design variables  $P_0$ ,  $\lambda_0$  and  $\theta_0$  on the convergence rate. We summarize this RMBP algorithm as follows:

$$\begin{aligned}
\underline{v}_q(\underline{\theta}_{q-1}) &= \underline{t}_q - \underline{a}(\underline{\theta}_{q-1}; \underline{p}_q) \\
\mu_q &= \begin{cases} \mu_{q-1} \cdot \beta & \|\underline{v}_q(\underline{\theta}_{q-1})\| < \|\underline{v}_{q-1}(\underline{\theta}_{q-1})\| \\ \mu_{q-1} / \beta & \|\underline{v}_q(\underline{\theta}_{q-1})\| \geq \|\underline{v}_{q-1}(\underline{\theta}_{q-1})\| \end{cases} \\
\mathbf{K}_q &= \frac{1}{(\mu + 1)^2} \mathbf{P}_{q-1} \phi_q \left\{ \lambda \mathbf{W}^{-1} + \phi_q^T \mathbf{P}_{q-1} \phi_q \right\}^{-1} + \frac{\mu}{(\mu + 1)^2} \phi_q \mathbf{W} \\
\underline{\theta}_q &= \underline{\theta}_{q-1} + \mathbf{K}_q \underline{v}_q(\underline{\theta}_{q-1}) \\
\mathbf{P}_q &= \frac{1}{\lambda} \left[ \mathbf{P}_{q-1} - \mathbf{P}_{q-1} \phi_q \left\{ \lambda \mathbf{W}^{-1} + \phi_q^T \mathbf{P}_{q-1} \phi_q \right\}^{-1} \phi_q^T \mathbf{P}_{q-1} \right]
\end{aligned} \tag{6-7}$$

In the test,  $\mu_0 = 0.001$  and  $\beta = 0.95$ . The test results are listed in Table 6-3. From Table 6-3, we find that the RMBP without matrix inversion requires a little more total flops to converge than RGN, but much less than RMBP with matrix inversion. If the design variables  $\lambda_0 = 0.88$  or  $0.9$ ,  $P_0 = 10^3$  and  $\underline{\theta}_0$  is initialized to small random values, the total flops and iterations have smaller values, i.e. the convergence is faster. It was observed that if  $\beta$  took on the values close to and less than 1, the same conclusion was maintained. If  $\beta$  is too small, e.g. 0.8, it requires much more total flops and iterations than those indicated in Table (6-3) to converge.

TABLE 6-3  
TEST RESULTS FOR RMBP WITHOUT INVERSE FOR PROB. #1

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$	$10^5$	159.2	8.6958e+7	0.01921
$(\lambda_0=0.88,$	$10^3$	157.6	8.5734e+7	0.01919
$\theta_0$ -NW)	10	195.333*	1.3123e+8	0.01984
	0.95	168.5	9.2014e+7	0.01963
$\lambda_0$	0.92	167.7	9.1582e+7	0.01955
$(P_0=10^3,$	0.90	159.333*	8.7008e+7	0.01943
$\theta_0$ -NW)	0.88	157.6	8.5734e+7	0.01919
	0.86	170.667*	9.3197e+7	0.01983
$\theta_0$ ( $P_0=10^3,$	NW	157.6	8.5734e+7	0.01919
$\lambda_0=0.88)$	Small rand.	152.9	8.2322e+7	0.01902

## 6.4 Test Procedures/Results For Test Problem #2

### 6.4.1 Test Results for RGN

In section 6.2, we defined the second test problem as a set of examples from a square wave indicated in (6-2). In this section, we will apply RGN and RMBP to this test problem. Because the SSE of the standard RGN (6-3) cannot reach 0.02 within the maximum number of iterations, which we set to 800, we will incorporate a term into the standard RGN (6-3) to form the improved version of RGN (6-

4). We will apply this improved version of RGN, as well as RMBP with matrix inversion (6-5) and RMBP without matrix inversion (6-7) to the second test problem, investigating the effects of the design variables on the convergence rate.

First, we will use the improved RGN (6-4). The test results shown in Table (6-4) indicate that if the design variable  $\lambda_0$  takes on values near 0.86 or 0.88, and  $P_0=10^3$  and  $\underline{\theta}_0$  is initialized by the NW method, the total flops and iterations have smaller values. Note that there are failures when  $\lambda_0 = 0.86$ . we note that the modified RGN is comparable with CGBP in terms of the total flops (the total flops for CGBP for this test problem is  $1.49e+8$ , refer to [1])

TABLE 6-4  
TEST RESULTS FOR MODIFIED RGN (6-4) FOR PROB. #2

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$	$10^5$	525.4	4.1530e+8	0.01930
$(\lambda_0=0.88,$ $\underline{\theta}_0$ NW)	$10^3$	503.3	3.9783e+8	0.01994
	10	626.7	4.9537e+8	0.01877
	0.95	607.1	4.7988e+8	0.01946
$\lambda_0$	0.92	587.5	4.6438e+8	0.01875
$(P_0=10^3,$ $\underline{\theta}_0$ NW)	0.90	566.2	4.4755e+8	0.01996
	0.88	503.3	3.9783e+8	0.01994
	0.86	531.42*	4.2006e+8	0.01895
$\underline{\theta}_0$ ( $P_0=10^3,$ $\lambda_0=0.88$ )	NW	503.3	3.9783e+8	0.01994
	Small rand.	510.2	4.0328e+8	0.01830

#### 6.4.2 Test Results for RMBP With Matrix Inversion

Next, we apply RMBP with matrix inversion to the problem #2 and investigate the effects of the design variables  $P_0$ ,  $\lambda_0$ ,  $\theta_0$  and  $\mu$  on the convergence rate. The test results are listed in Table 6-5. From Table 6-5, we find that the RMBP with inversion needs more total flops to converge than RGN, even though it needs fewer iterations. If the design variable  $\lambda_0 = 0.86, 0.88$  or  $0.90$ , and  $P_0 = 10^3$ ,  $\mu$  takes on variable values by means of (6-6), and  $\theta_0$  is initialized to small random values, the total flops and iterations have smaller values, i.e. the convergence is faster.

#### 6.4.3 Test Results for RMBP Without Matrix Inversion

In this section, we apply RMBP without matrix inversion to the second test problem and investigate the effects of the design variables  $P_0$ ,  $\lambda_0$  and  $\theta_0$  on the convergence rate. The test results are listed in Table 6-6. From Table 6-6, we find that the RMBP without matrix inversion and RGN require similar total flops to converge. However, the RMBP without matrix inversion needs fewer iterations than RGN. If the design variables  $\lambda_0 = 0.88$  or  $0.95$ , and  $P_0 = 10^3$  and  $\theta_0$  is initialized by the NW method, the total flops and iterations have smaller values, i.e. the convergence is faster.

TABLE 6-5  
TEST RESULTS FOR RMBP WITH INVERSE FOR PROB. #2

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$ ( $\lambda_0=0.88$ ,	$10^5$	490.3	9.0191e+9	0.01930
$\theta_0$ -NW	$10^3$	429.1	7.8933e+9	0.01907
$\mu=1e-3$ )	10	508.2*	9.3483e+9	0.01873
	0.95	430.5	7.9190e+9	0.01961
$\lambda_0$ ( $P_0=10^3$ ,	0.92	457.1	8.4083e+9	0.01972
$\theta_0$ -NW,	0.90	411.6	7.5714e+9	0.01814
$\mu=1e-3$ )	0.88	429.1	7.8933e+9	0.01907
	0.86	423,7	7.7940e+9	0.01964
$\theta_0$ ( $\lambda_0=.88$ ,	NW	429.1	7.8933e+9	0.01907
$P_0=10^3, \mu=1e-3$ )	Small rand.	422.3	7.7682e+9	0.01953
$\mu$ ( $\lambda_0=.88$ ,	fixed	429.1	7.8933e+9	0.01907
$P_0=10^3, \theta_0$ NW)	set by (6-6)	428.2	7.8937e+9	0.01944

TABLE 6-6  
 TEST RESULTS FOR RMBP WITHOUT INVERSE FOR PROB. #2

VARIABLES INVESTI.	TESTING CONDITIONS	NUMBER OF ITERATIONS	TOTAL FLOPS	ULTIMATE SSE
$P_0$	$10^5$	403.6	4.2557e+8	0.01823
$(\lambda_0=0.88,$ $\theta_0\text{-NW})$	$10^3$	384.5	4.0550e+8	0.01804
	10	597.2	6.3128e+8	0.01843
	0.95	387.1	4.0878e+8	0.01876
$\lambda_0$	0.92	421.7	4.4627e+8	0.01922
$(P_0=10^3,$ $\theta_0\text{-NW})$	0.90	405.7	4.2842e+8	0.01790
	0.88	384.5	4.0550e+8	0.01804
	0.86	411.3	4.3433e+8	0.01972
$\theta_0 (P_0=10^3,$ $\lambda_0=0.88)$	NW	384.5	4.0550e+8	0.01804
	Small rand.	396.2	4.1839e+8	0.01945

## 6.5 Summary of Test Results

In this chapter, we defined two test problems: one is a set of examples from a sine wave (see (6-1)) and the other is a set of examples from a square wave (see (6-2)). The RGN and RMBP algorithms were implemented and tested on the two test problems. The properties of RGN and RMBP under the tests were revealed. Some details related to the implementations were pointed out. The results derived in sections 6-3 and 6-4 are summarized as follows.

1. The standard RGN (6-3) derived in Chapter 5 may not converge in terms of the stopping rule we defined in section 6-2 (see test problem #2). Some improved versions of the RGN could solve the problem (for example (6-4)) because in these algorithms such uncertainties as calculation errors, truncation errors, etc. are taken into account. In (6-4), the term  $R_q$  deals with such uncertainties. This term also makes faster convergence.
2. In RGN and its improved versions, the design parameter  $\lambda_0$  is a important factor to accelerate the convergence rate. We found that  $\lambda_0 = 0.88$  to  $0.90$  works better in all tests. It is appropriate to set the design parameter  $P_0$  to  $10^3$ . A  $P_0$  too large, like  $10^5$ , may cause the learning process to oscillate; while a  $P_0$  too small, like  $10$ , causes the learning process to become very slow or stops the learning after a few iterations.
3. Two versions of RMBP were investigated. The iterations required for convergence in the exact form of RMBP (6-5) are less than those needed in RGN. However, the total flops needed



in the exact RMBP are much more than those needed in RGN, because of the inversion of the Hessian matrix. A version of RMBP without inversion (6-7) was also investigated. The total flops and iterations required for convergence for this algorithm are similar to those for RGN. The parameter  $\mu$  makes the algorithm (6-7) not sensitive to the value of  $\lambda$ .

4. The advantage of RMBP and its variant is that they can work in all cases, without incorporating some ad hoc design parameters such as those used in the improved versions of RGN.

## CHAPTER VII

### CONCLUSIONS

In this thesis, we developed two recursive learning algorithms, the Recursive Gauss-Newton method (RGN) and the Recursive Marquardt method (RMBP) to train Multilayer feedforward Perceptron type Neural Networks (MPNNs), based on the batching Gauss-Newton method and the batching Marquardt method (MBP). We also derived a Marquardt Backpropagation (MBP) and a quasi-recursive Gauss-Newton method (QRGN) to train MPNNs using Iterated Least Squares. The MBP and QRGN are the starting points to develop RGN and RMBP.

The difference between QRGN and RGN was depicted in Figures 10 and 11. We showed in Chapter 5 that RGN, like other recursive algorithms, was an approximation to the batching Gauss-Newton method under certain assumptions. RMBP improves RGN just as MBP improves GN.

We applied RGN and RMBP to train MPNNs to approximate two nonlinear functions. The details of the implementation were pointed out, and the test results were summarized in Chapter 6. Standard RGN may not converge. Modified RGN has superior convergence rate in our test. RMBP with matrix inversion requires intensive computational burdens. RMBP without matrix inversion has a convergence rate which is similar to that of RGN. The advantage of

RMBP without matrix inversion is that it is not too sensitive to the values of  $\lambda_0$ .

RGN and RMBP can be used in the situation where an on-line learning algorithm is needed and calculation speed is critical. The current versions of RGN and RMBP were not tested on more complex problems, for example, multiple-output MPNNs.

The RGN and RMBP algorithms are derived from optimization techniques. On the one hand, the development of optimization techniques will inspire us to improve RGN and RMBP to be more powerful learning algorithms to train MPNNs. On the other hand, there exist many problems with practical applications of RGN and RMBP. We need to make more efforts to deal with these problems in the future.

## REFERENCES

- [1] M. T. Hagan, and M. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," accepted for publication, IEEE Transactions on Neural Networks.
- [2] R. Hecht-Nielsen, Neurocomputing. Addison-Wesley (1990).
- [3] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations. MIT Press (1986).
- [4] G. E. F. Seber, Nonlinear Regression. John Wiley (1989).
- [5] D.M. Himmelblau, Applied Nonlinear Programming. McGraw-Hill (1972).
- [6] P. E. Gill, and K. Murray, Practical Optimization. Academic (1981).
- [7] T.P Vogl, et. al., "Accelerating the Convergence of the Backpropagation Method," Biological Cybernetics, vol. 59, p 257-263, 1988.
- [8] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Network by Choosing Initial Values of the Adaptive Weights," Proceedings of the IEEE International Joint Conference on Neural Networks, vol. III, p.21-26, July 1990.
- [9] Dennis, J. E., Jr. and More, J. J., "Quasi-Newton methods, motivations and theory," SIAM Review, vol. 19, no. 1, p 46-89, 1977.

- [11] A. Guez and J. Selinsky, "A Trainable Neuromorphic Controller," *Journal of Robotic Systems*, vol. 5, no. 4, p 363-388, 1988.
- [12] J. Mendel Lessons on Digital Estimation Theory. Prentice-Hall (1987).
- [13] R.A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, Vol. 1, No. 4, pp 295-308, 1988.
- [14] A. von Ooyen and B. Nienhuis, "Improving the Convergence of the Backpropagation Algorithm." *Neural Networks*, Vol. 5, No. 3, 1992, pp 465-471.
- [15] A.K. Rigler, J.M. Irvine and T.P. Vogl, "Rescaling of Variables in Back Propagation Learning," *Neural Networks*, Vol. 3, No. 5, 1990, pp 561-573.
- [16] E. Barnard, "Optimization for Training Neural Nets," *IEEE Trans. on Neural Networks*, Vol. 3, No. 2, March 1992, pp 232-240.
- [17] D.C. Liu and J. Nocedal, "On the Limited Memory BFGS Method for Large Scale Optimization," *Mathematical Programming*, Vol. 45, 1989, pp 503-528.
- [18] M. T. Hagan and R. Klein, "On-line Maximum Likelihood Estimation for Load Forecasting" *IEEE Trans. on SMC*, Vol. 8, No. 9, Sept 1978, pp 711-715.
- [19] S. Kollias and D. Anastassiou, "An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks," *IEEE Trans. on Circuits and Systems*, Vol. 36, No. 8, August 1989, pp 1092-1101.
- [20] S. Singhal and L. Wu, "Training Multilayer Perceptrons with the Extended Kalman Algorithm," in *Advances in Neural*

- Information Processing Systems I, D.S. Touretzky (Ed.), San Mateo CA:Morgan Kaufman, 1989, pp 133-140.
- [21] G.V. Puskorius and L.A. Feldkamp, "Decoupled Extended Kalman Filter Training of Feedforward Layered Networks," IJCNN Proceedings, Vol. I, July 1991, pp 771-777.
- [22] D. Marquardt, "An Algorithm for Least Squares Estimation of Non-Linear Parameters," J. Soc. Ind. Appl. Math., 1963, pp 431-441.
- [23] The Manual of The Neural Network Toolbox, The Mathworks Inc., 1992.

Vita

Zhimin Lu

Candidate for the Degree of  
Master of Science

Thesis: OPTIMIZATION TECHNIQUES APPLIED TO NEURAL NETWORKS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Shanghai, P.R. China, January 1962, the son of Jiayi Lu and Hongying Zhang.

Education: Graduated From Kongjiang Senior High School, Shanghai, in July 1980; received Bachelor of Engineering Degree in Electrical Engineering from Shanghai University of Engineering Science in July 1986; completed requirements for the Master of Science degree at Oklahoma State University in December 1993.

Professional Experience: Research Assistant, Department of Electrical and Computer Engineering, 1992 through 1993. Supervisor of FA division in R & D department of Shanghai Applied Technology Co., Shanghai, P. R. C. from Jan. 90 to Jul. 91. Electrical Engineer in Shanghai Bell Telephone Equipment Mfg. Co. Ltd. from Jul. 86 to Dec. 89.

Membership in Professional Societies: China Institute of Automation; Association of Shanghai Electronic and Electrical Technology; Association of Science and Technology of China; SIAM.