TRAINING MODULAR NEURAL NETWORKS WITH MARQUARDT-LEVENBERG ALGORITHM

By

MENG HOCK FUN

Bachelor of Engineering

Oklahoma State University

Stillwater, Oklahoma

1993

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
May, 1996

TRAINING MODULAR NEURAL NETWORKS WITH MARQUARDT-LEVENBERG ALGORITHM

Thesis Approved:

Martin T Hagan

Thesis Adviser

R. Ramakumar.

Thomas C. Collins

Dean of Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my major adviser, Dr. Martin Hagan for his brilliant ideas, intelligent supervision, guidance, support, encouragement and friendship. My achievement today would not be possible without his generous help. I would like to thank Texas Instrument and Department of Electrical Engineering for providing me with this research opportunity and their generous financial support. In addition, I would like to thank other committee members, Dr. Ramakumar and Dr. Acton, for giving invaluable helps in correcting my thesis.

Moreover, I would like to express my special appreciation to my girlfriend, Amelia Lor, for her strong encouragement at times of difficulty, love, and understanding. Thanks also go to my mother, my beloved father, my sisters and brother for their encouragement and support.

TABLE OF CONTENTS

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

Figure **Page**

Figure **Page**

LIST OF SYMBOLS

(s), (v), and (m) represent scalar, vector and matrix respectively.

a	the output of a single neuron. (s)
a_j	the j^{th} element of the network output of a single layer network. (s)
a_j^m	the j^{th} element at m^{th} layer of the network output of a multilayer network. (s)
$a_{j_i}^m$	the j^{th} element at m^{th} layer of the i^{th} expert network output. (s)
$\mathbf{a}$	the output vector of a single layer network. (v)
$\mathbf{a}^m$	the m^{th} layer network output vector. (v)
$\mathbf{a}_i^m$	the m^{th} layer output vector of the i^{th} expert network. (v)
$\mathbf{A}_k$	the Hessian matrix of $J(\mathbf{x})$. (m)
b	the scalar bias. (s)
b_j	the j^{th} element of the bias vector. (s)
b_j^m	the j^{th} element of the m^{th} layer bias vector. (s)
$b_{j_i}^m$	the j^{th} element of the m^{th} layer bias vector of the i^{th} expert network. (s)
$\mathbf{b}$	the bias vector of a single layer network. (v)
$\mathbf{b}^m$	the m^{th} layer bias vector. (v)
$\mathbf{b}_i^m$	the m^{th} layer bias vector of the i^{th} expert network. (v)
$\mathbf{B}(\mathbf{x})$	the second derivative of the gating network's weights and biases. (m)
$\mathbf{c}_k$	the gradient vector of $J(\mathbf{x})$. (v)

$\mathbf{d}_k$	the eigenvectors. (v)
e	the scalar net input of the gating network. (s)
e_l	the l^{th} element of the net input vector of the gating network. (s)
e_l^m	the l^{th} element of the m^{th} layer net input vector of the gating network. (s)
$\mathbf{e}^m$	the m^{th} layer net input vector of the gating network. (v)
$\mathbf{e}_j$	the error at j^{th} expert network.
f	the activation function. (s)
$\mathbf{f}$	the activation function vector. (v)
$\mathbf{f}^m$	the activation function vector at m^{th} layer. (v)
$\mathbf{f}_i^m$	the activation function vector at m^{th} layer of the i^{th} expert network. (v)
g_i	the activation of the i^{th} output of the gating network. (s)
$g_{i,q}$	the activation of the i^{th} output of the gating network at q^{th} data. (s)
G	the maximum number of expert plus one; $N+1$. (s)
$\mathbf{G}_k$	the Marquardt-Levenberg modification to the Hessian matrix $\mathbf{A}_k$. (m)
h_i	the i^{th} element of the posterior probability. (s)
$\mathbf{H}(\mathbf{x})$	the second derivative of all the expert networks' weights and biases. (m)
$\mathbf{I}$	the identity matrix. (m)
$\mathbf{j}$	the performance index vector of all the input data. (v)
J, $J(\mathbf{x})$	the performance index. (s)
J_q	the performance index at q^{th} data (a scalar term). (s)
$\mathbf{K}_i$	Jacobian matrix for the i^{th} expert network. (m)
$\mathbf{K}_{(t,d)_i}$	Jacobian matrix for the i^{th} expert network with t row and d column. (m)

$\mathbf{K}_{(t,d)_G}$	Jacobian matrix for the gating network with t row and d column. (m)
$\mathbf{m}$	the total output of the networks. (v)
$\mathbf{m}_q$	the total output of the network for the q^{th} input data. (v)
n	the net input to the activation function. (s)
n_j	the j^{th} element of the net input. (s)
n_j^m	the j^{th} element at m^{th} layer of the net input. (s)
$n_{j_i}^m$	the j^{th} element at m^{th} layer of the i^{th} expert network net input. (s)
$n_{j_{i,q}}^m$	the j^{th} element at m^{th} layer of the i^{th} expert network net input at q^{th} data. (s)
$\mathbf{n}$	the net input vector of a single layer network. (v)
$\mathbf{n}^m$	the m^{th} layer net input vector. (v)
$\mathbf{n}_i^m$	the m^{th} layer net input vector of the i^{th} expert network. (v)
N	the maximum number of expert networks. (s)
p	scalar input to a neural network. (s)
p_j	the j^{th} element of the input vector. (s)
$\mathbf{p}$	input vector to a neural network or actual tasks. (v)
$\underline{\mathbf{p}}$	classes of each task. (v)
$\mathbf{p}_k$	the direction vector. (v)
q	q^{th} element of data.
q_l	the l^{th} element of the gating network bias vector. (s)
q_l^m	the l^{th} element at the m^{th} layer of the gating network bias vector. (s)
$\mathbf{q}$	the gating network bias vector. (v)
$\mathbf{q}^m$	the m^{th} layer of the gating network bias vector. (v)
Q	the maximum number of data.

$\mathbf{r}_i$ the weights and biases of the i^{th} expert network that arrange in a column that form a vector (v)

$\mathbf{r}_{i_k}$ the i^{th} expert network's weights and biases at k^{th} iteration. (v)

R the number of inputs to a neural network. (s)

s^m_{j,t_i} the sensitivity of the t^{th} element of the last layer output to a change in the net input of unit j in layer m of the i^{th} expert network. (s)

$s^m_{j,t_{i,q}}$ the sensitivity of the t^{th} element of the last layer output to a change in the net input of unit j in layer m of the i^{th} expert network at q^{th} data. (s)

s^m_{j,t_G} the sensitivity of the t^{th} element of the last layer output to a change in the net input of unit j in layer m of the gating network. (s)

$s^m_{j,t_{G,q}}$ the sensitivity of the t^{th} element of the last layer output to a change in the net input of unit j in layer m of the gating network at q^{th} data. (s)

S the maximum number of neuron. (s)

S^m the maximum number of neuron in m^{th} layer. (s)

S^m_i the maximum number of neurons in m^{th} layer of i^{th} expert network(s)

$\mathbf{S}^m_i$ the m^{th} layer sensitivity matrix of the i^{th} expert network. (v)

u_l the l^{th} element of the output vector of the gating network. (s)

u^m_l the l^{th} element of the m^{th} layer gating network output vector. (s)

$u_{l,q}$ the l^{th} element at the q^{th} data of the gating network output vector. (s)

$u^k_{l,q}$ the l^{th} element at the q^{th} data and k^{th} layer of the gating network output vector. (s)

$\mathbf{u}$ the output vector of the gating network. (v)

$\mathbf{u}^m$ the output vector at the m^{th} layer gating network. (v)

$\mathbf{u}_q$ the output vector of the gating network at q^{th} data. (v)

$v_{l,j}$ the l, j element of the gating network's weights(s)

$v^{m}_{l,j}$	the *l*, *j* element of the m^{th} layer gating network's weights (s)
$\mathbf{V}$	the gating weight matrix. (m)
$\mathbf{V}^{m}$	the m^{th} layer gating network's weight. (m)
w	the scalar weight. (s)
$w_{j,k}$	the *j*, *k* element of the weight. (s)
$w^{m}_{j,k}$	the *j*, *k* element at the m^{th} layer of the weight. (s)
w^{m}_{j,k_i}	the *j*, *k* element at the m^{th} layer of the weight of the i^{th} expert network. (s)
$\mathbf{w}_i$	the i^{th} expert network weight matrix that arrange in a column vector. (v)
$\mathbf{W}$	the weight matrix. (m)
$\mathbf{W}^{m}$	the m^{th} layer weight matrix. (m)
$\mathbf{W}^{m}_{i}$	the m^{th} layer weight matrix of the i^{th} expert network. (m)
x_j	the j^{th} element of vector $\mathbf{x}$. (s)
$\mathbf{x}$	a parameter vector represents all the weights and biases in a modular network. (v)
$\mathbf{x}^{*}$	the nominal point of $\mathbf{x}$. (v)
$\mathbf{x}_{0}$	the initial value of $\mathbf{x}$. (v)
$\mathbf{x}_{k}$	the value of $\mathbf{x}$ at time step *k*. (v)
$\mathbf{x}_{k+1}$	the value of $\mathbf{x}$ at time step $k+1$. (v)
$\Delta\mathbf{x}_{k}$	$\mathbf{x}_{k+1} - \mathbf{x}_{k}$ (v)
y	the scalar output of the modular network. (s)
y_i	the scalar output of the i^{th} expert network. (s)
y_{m_i}	the m^{th} element of the output of the i^{th} expert network. (s)
$y_{m_{i,q}}$	the m^{th} element of the output of the i^{th} expert network that correspond to q^{th} data. (s)

$y^{k}_{m_{i,q}}$ the m^{th} element at k^{th} layer of the output of the i^{th} expert network that correspond to q^{th} data. (s)

$\mathbf{y}$ the output vector of the modular network. (v)

$\mathbf{y}_i$ the output vector of the i^{th} expert network. (v)

$\mathbf{y}_{i,q}$ the output vector of the i^{th} expert network at q^{th} data. (v)

$\mathbf{y}^*$ the target vector. (v)

$\mathbf{y}^*_q$ the target vector at q^{th} data. (v)

$\mathbf{z}$ the weights and biases of the gating network that arrange in a column that form a vector (v)

$\mathbf{z}_k$ the gating network's weights and biases at k^{th} iteration. (v)

α_k the learning rate. (s)

β the increasing and decreasing factor.

ε_l the Gaussian random vector with zero mean and covariance matrix σ^2I, where I is the identity matrix and σ^2 is the variance of the input vector, **p**. (v)

$\Delta_{i,j}$ the i, j element of the update value. Use in Rprop algorithm.

Δ_o the initial value of Δ. Use in Rprop algorithm.

$\Delta_{\max}$ the maximum value of Δ. Use on Rprop algorithm.

∇ the gradient vector. (v)

∇^2 the Hessian matrix. (m)

λ_k the eigenvalues. (s)

η^+ the increasing factor. Use in Rprop algorithm.

η^- the decreasing factor. Use in Rprop algorithm.

μ_k the increasing and decreasing parameter for Marquardt-Levenberg method. (s)

CHAPTER I

INTRODUCTION

This research studies a particular architecture of neural network called the modular neural network. This neural network architecture is capable of performing piecewise control strategies and implementing discontinuous functions; in other words, it can partition a plant's parameter space into several regions and assign different neural networks to learn separate control laws in each region.

The main focus of this research is to develop a fast training algorithm, which is called the Marquardt-Levenberg (**ML**) algorithm, for the modular network. The need for this algorithm is inspired from the slow convergence rate of the conventional **S**teepest **A**scent training method (**SA**) [9] [11] for the modular network. To compare the performances of this newly developed algorithm, two other gradient methods, Resilient Backpropagation (**Rprop**) and Steepest Ascent (**SA**) algorithms, are developed for the modular network. These comparison results are discussed in Chapter VI.

In addition, we have also investigated two methods to improve the training time: the what and where modular network architecture and the gating weights' initialization. We have also included several possible applications using the modular networks to model friction.

This document contains eight chapters. Starting from the basic neural network building block -- an artificial neuron model, the feedforward multilayer network is derived and described in the Chapter II.

Using the feedforward multilayer network as each of the modules in modular network, we will introduce the modular network architecture in Chapter III. One section of this Chapter III is devoted to compare the modular network and the multilayer network. This is followed by a section discussing how the modular network works.

In Chapter IV, we will present the modular network's learning rules. In the first part of this Chapter, we will discuss how the modular network is formulated from a statistical point of view. In the second part, we will present the **SA** method and use it in the training of the modular network.

Chapter V discusses a faster training algorithm -- the **ML** algorithm, and shows how it can be incorporated into the modular network. In this section, we will give a brief introduction of the Marquardt-Levenberg algorithm. Then, we formulate this algorithm for the modular networks. Details of each equation are presented in Appendices A to C.

In Chapter VI, we will apply the **ML** training for the modular network and will compare it to the **SA** method and the **Rprop** method. Several test problems are simulated and the results are compared and summarized at the end of this Chapter.

In Chapter VII, we will first compare the modular network with the multilayer network. Then, we will investigate two methods to speed up the training. These methods are the modular network with prior knowledge versus the modular network without prior knowledge and the gating weight's initialization. In addition, several tests are performed to model the friction.

Finally, Chapter VIII summaries the important results and some future research that can be done on this Marquardt-Levenberg training method for the modular network. In conclusion, we also summarize some possible applications in modeling and controlling a system that contains friction dynamics.

CHAPTER II

MULTILAYER NEURAL NETWORKS

The works on multilayer neural networks are motivated right from the studies of how the human brain processes information. To understand how the multilayer neural network works, we must study how the human brain processes information; and most important, the information processing nerve cell -- the neuron.

Biological Neuron

The struggle to understand how the brain operates owes much to the pioneering work of Ramon and Cajal [1], who first introduced the idea of neurons. There are approximately 10^{11} neurons in the brain, and each neuron has approximately 10^{4} connections with adjacent neurons.

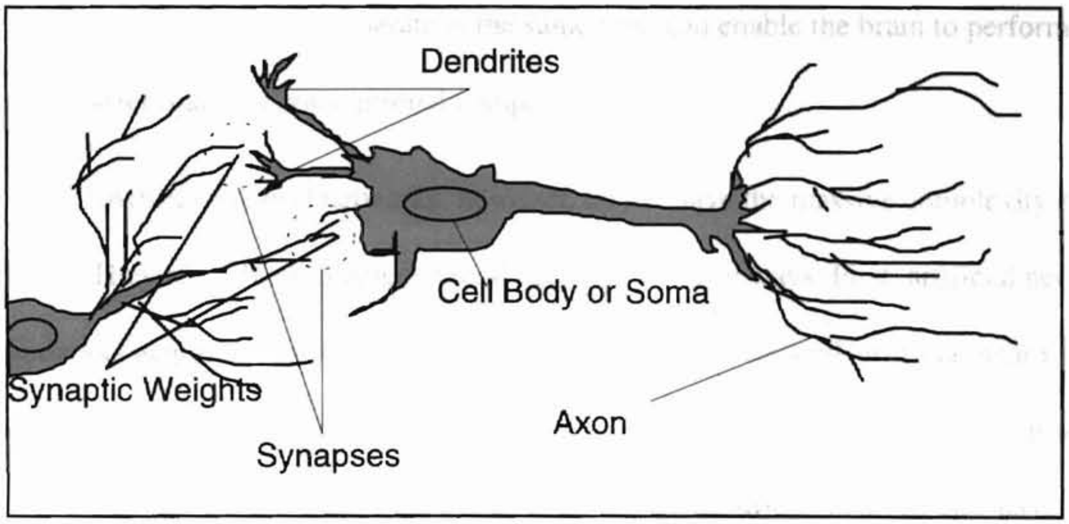


Figure 2 - 1. Two Neurons

There are 4 important parts in a neuron: synapses, dendrites, cell body (soma) and axons. The junction points between the axons and the dendrites are the *synapses*. The inputs, which are electrical signals, transmit through the synaptic junctions from the axons of adjacent neurons to the *dendrites*. These inputs are modulated or weighted by the complex chemical process in the synapses, which the biologists call synaptic weights, and are carried away by the dendrites into the *cell body (soma)*. The cell body sums and threshold these modulated incoming electrical signals and passes them to the *axon*. The axon, a single long fiber, then carries the outgoing electrical signal from the cell body to the other neurons. Figure 2 - 1 shows a simplified biological neuron that is connected with another biological neuron.

Typically, biological neurons are 5 to 6 orders of magnitude slower than silicon logic gates (10^{-3} seconds compare to 10^{-9} seconds). However, brain makes up for the relatively slow rate of operation by having massive interconnections and massive parallel structure between neurons. Because of these massive interconnections and parallel structure, all neurons can operate at the same time and enable the brain to perform many tasks faster than any conventional computer.

Artificial neural networks, however, do not have the massive complexity of the brain. They resemble biological neural networks in three ways. First, artificial neural networks acquire knowledge through a learning process. Second, artificial neural networks are made up of simple building blocks, where these building blocks imitate simple neurons and are highly interconnected. Lastly, artificial neurons also have artificial weights, which imitate the synaptic weights, to store knowledge.

Biological Neuron to Artificial Neuron Model

To imitate the biological neurons, engineers and mathematicians have developed an artificial neuron model of the biological neuron, called a *perceptron*. It has an input (electrical signal), artificial weight, bias (synaptic weights), connection between weight and summer (dendrite), a summer, an activation function (cell body) and an output (axon). Figure 2 - 2 shows a perceptron in symbolic representation.

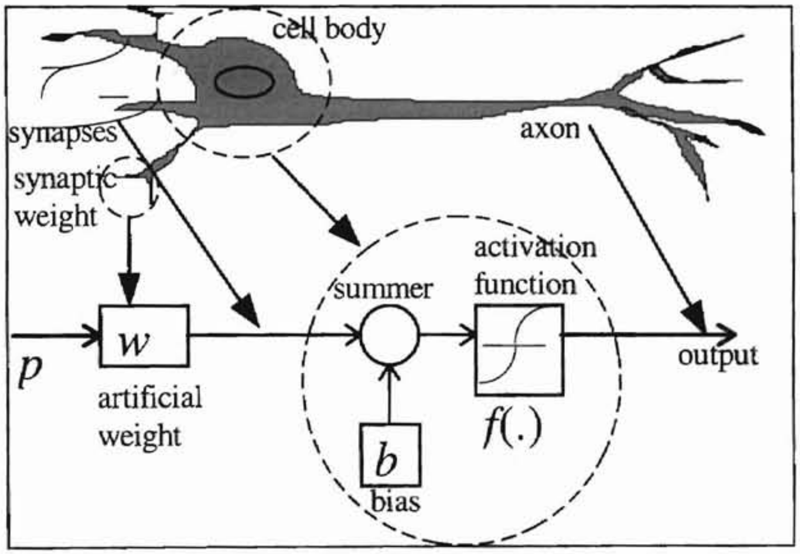


Figure 2 - 2. Artificial Neuron: Imitation of Biological Neuron

The input, *p*, is multiplied by a scalar artificial weight, *w*, to form $w \times p$, which imitates the modulated electrical signal of the synaptic weight in the biological neuron. Then, the weighted input, $w \times p$, is sent to a summer, which imitates the modulated electrical signal carried by the dendrite and sent to the cell body. The summer and activation function closely resemble the cell body, which has the effect of summing and thresholding the modulated electrical signal. After the weighted input, $w \times p$, is processed by the summer and activation function, it is sent to the output, which is represented by the electrical signal carried to the axon in the biological neuron. The artificial neuron model shown in Figure 2 - 2, also included an externally applied bias that has the effect of increasing or decreasing the net input of the activation function. Without the bias, the neuron could not perform an affine transformation. The following example shows the importance of the bias.

Example

Assume that we have an artificial neuron that has no bias and uses a linear activation function. Given p = 0 we want an output of 1. Can the artificial neuron achieve this task?

***Ans.**: No, any scalar weight, w, multiplied by p is 0.*

Typically, the bias, *b,* is considered to be part of the weight, except that it has a constant 1 as an input. There are some artificial neural networks which use no bias, such as the Kohonen and Hebbian Network Architectures, but these networks are beyond the discussion of this research. Also, from this point on, we will refer only to artificial neural networks and not to biological neural networks.

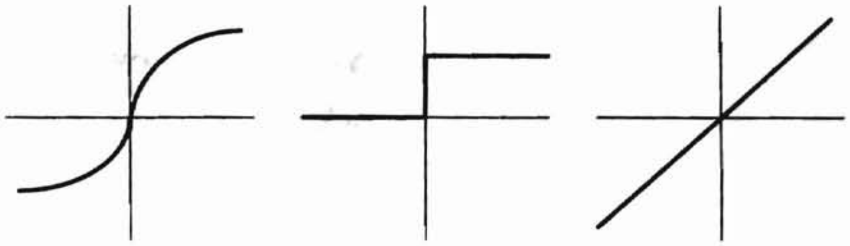
The equations that describe an artificial neuron are:

$$n = wp + b \qquad \text{and} \qquad a = f(n)$$

The objective of neural network training is to adjust the *w* and *b* in each artificial neuron so that the input and output relationship meets some specific goals.

Activation Function of a Neuron

Since the actual output depends on a particular activation function, it is important for a designer to select a suitable activation function to suit a particular task. There are several varieties of activation functions. Three of the most common activation functions are hyperbolic tangent sigmoid, hard limiter, and linear. Figure 2 - 3 shows these typical activation functions:



$$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}} \qquad a = \begin{cases} +1 & \text{if } n \geq 0 \\ 0 & \text{if } n < 0 \end{cases} \qquad a = n$$

$a = \mathrm{tansig}(n)$ $a = \mathrm{hardlim}(n)$ $a = \mathrm{purelin}(n)$

Figure 2 - 3 Three Typical Activation Functions

The left side of Figure 2 - 3 shows a hyperbolic tangent activation function. This activation function is used most commonly in backpropagation networks because it is a monotonically increasing function and it is differentiable.

In the center, we have the hard-limiting activation function. Neurons that use this activation function are commonly referred to as *McCulloch-Pitts* neurons. The output of such a neuron always has a value of 1 or 0. Typically, it is used for binary classification.

On the right, Figure 2 - 3 shows a linear activation function. Such activation functions are most commonly used in the last layer of a multilayer perceptron when used for function approximation.

There are many other activation functions. A list of other activation functions can be found in "Neural Network Design" page 2-6. [2].

Multilayer Networks Architecture

In order to solve complex problems, many neurons can be combined together to form multilayer perceptrons or multilayer neural networks. The main idea behind this multilayer perceptrons are the "layer". It allows each neuron within a layer to operate in parallel with other neurons.

Single Layer Perceptron

We will examine a one layer perceptrons in this section and then will expand to multilayer networks in the next section. Assume that we have *R* inputs and *S* neurons in a layer of perceptrons, then a layer of perceptrons is shown in Figure 2 - 4.

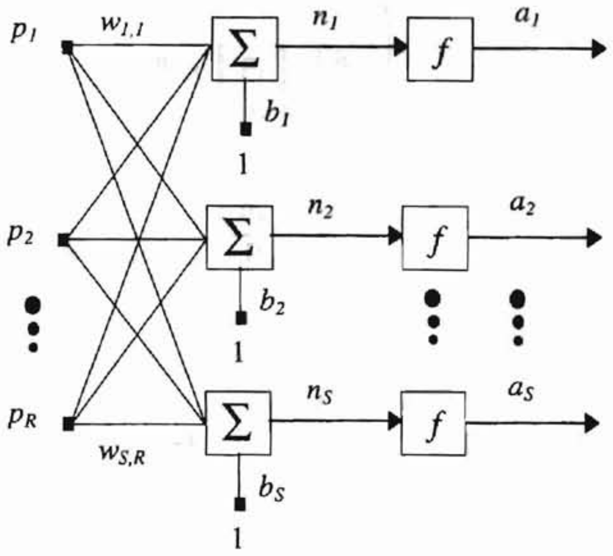


Figure 2 - 4. A single Layered Neural Networks with *S* Neurons

It seem rather complicated to calculate the output of the neural network as shown above. Fortunately, we can express the output of the one layer neural network using vectors and matrices. The output of the single layer neural network is given in vector form as

$$\mathbf{a} = \mathbf{f}(\underbrace{\mathbf{Wp} + \mathbf{b}}_{\mathbf{n}}) \qquad (2-1)$$

The weights are expressed in matrix form, **W**, and the inputs (**p**) and biases (**b**) in vector form. Taken together, the **Wp+b** forms the net input vector **n** .The activation function, **f**(.), then processes the net input vector **n** element by element and forms the output vector **a**.

$$\mathbf{a} = \mathbf{f}\left(\underbrace{\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix}}_{\mathbf{p}} + \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_R \end{bmatrix}}_{\mathbf{b}} \right) \qquad (2-2)$$

In the weight matrix, the row indices indicate the number of neurons, S, and the column indices indicate the number of inputs, R, to a layer of neurons. The total number of weights (included biases) in a layer is given by $S \times R + R$.

Multilayer Perceptron

Now that we have defined what we mean by a layer of neurons, we can easily cascade several layers of neurons to form a multilayer neural network. Each layer has its own weight matrix **W**, its own bias vector **b**, a net input vector **n**, and an output vector **a**. To distinguish between layers, we will use superscripts to identify the layer number. For example, $\mathbf{W}^1$ is the weight matrix for the first layer and $\mathbf{a}^3$ is the output vector for the third layer.

Figure 2 - 5 shows a two layer neural network and the notation is used thoroughly.

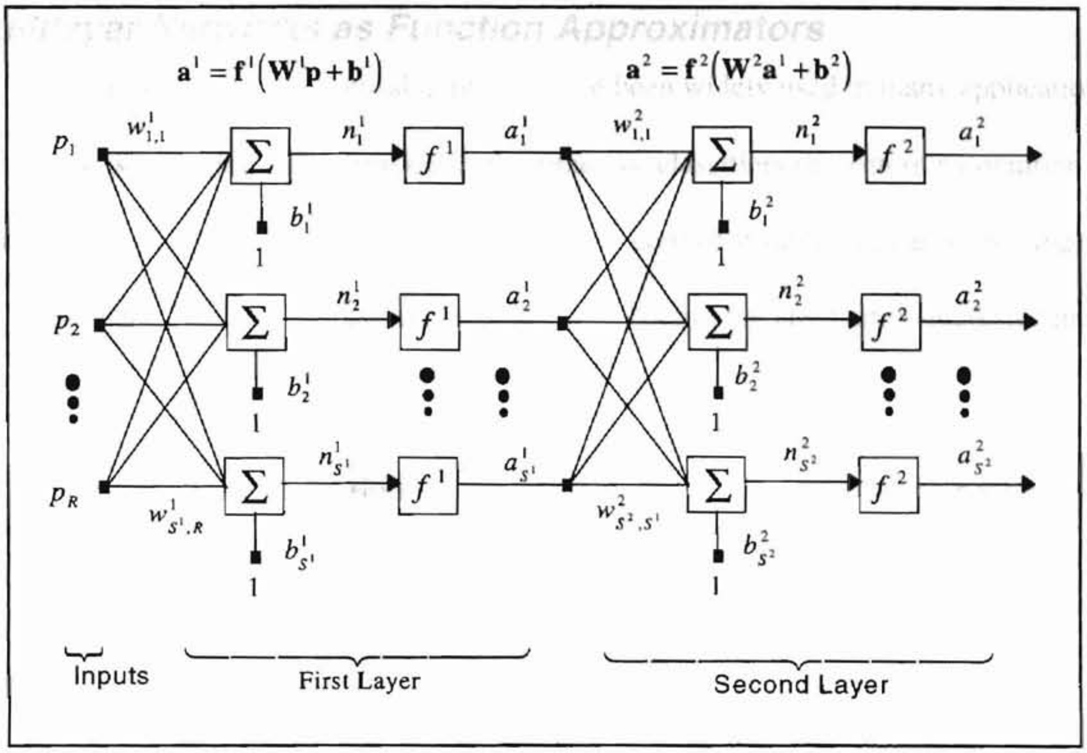


Figure 2 - 5. Two Layer Neural Network

The above network shows R inputs, S^1 neurons in the first layer and S^2 neurons in the second layer. S^1 and S^2 can be of different values. However, the last layer of the neurons must correspond to the number of outputs, which in this case is S^2. The last layer is also called the output layer. A notation used to describe the layered architecture is $R \times S^1 \times S^2 \times \cdots$ and so on. For example, 5-8-2 indicates a two layer network with 5 inputs, 8 neurons in the first layer and 2 outputs. Usually, if a network has more than 2 layers, we called the layer in between the input and output layer as the hidden layer.

A mathematical equation that describes the total output of the multilayer network is given by

$$\mathbf{a}^2 = \cdots\ \mathbf{f}^2\left(\mathbf{W}^2\left[\mathbf{f}^1\left(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1\right)\right]+\mathbf{b}^2\right) \cdots. \qquad (2\text{-}3)$$

Multilayer Networks as Function Approximators

During recent years, neural networks have been widely used in many applications. In general, we can think of neural networks either as classifiers (in pattern recognition, voice recognition) or as function approximators (in control systems, signal processing). Since the main focus of this research is in control systems, we are mainly interested in the function approximation areas.

It has been proven by Hornik [3] that a two layer neural network is a nonlinear parametric model and can approximate any continuous function. Consequently, an interesting notion is to use multilayer network as each of the modules in modular network. We will see in Chapter III that by using multilayer networks in each of the modules of a modular network, we can achieve modularity and improve several aspects of multilayer networks.

CHAPTER III

MODULAR NEURAL NETWORKS

The modular neural network was first presented by Robert A. Jacob and Michael I. Jordan [4] [9]. This network was designed from the statistical point of view; and it is capable of decomposing each task into several sub-tasks. Accordingly, this neural network architecture is well suited to perform piecewise control strategies; in other words, it can partition a plant's parameter space into different regions and select a different neural network to learn a separate control law in each region. For this reason, the modular network is capable of implementing discontinuous function. Also, due to its architecture, it is susceptible to the crosstalk problem. In the following, we will present the modular network architecture and compare the modular network with the multilayer neural network. Then, we will present the modular network's learning rules together with some preliminary simulation results.

Modular Network Architecture

The modular neural network consists of two kinds of sub-networks: the *expert networks* and a *gating network*. The expert networks are networks that compete to learn the input training patterns. There is more than one expert network in a modular network. The integrating unit is called the gating network; it is a network that mediates the competition of the expert networks.

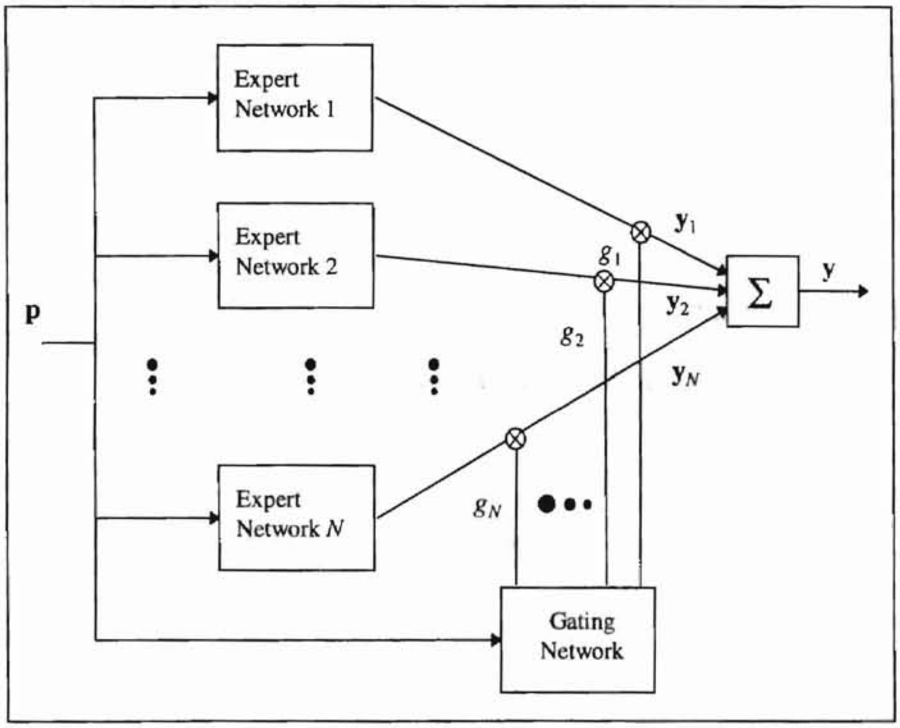


Figure 3 - 1. Block Diagram of a Modular Network

Figure 3 - 1 shows the modular network architecture. This architecture is composed of *N* expert networks and an integrating unit called the gating network. The total output of the modular network is calculated as follows. First, each expert network's

output is weighted by the gating network. Then, these weighted outputs are summed to give the total modular network output:

$$\mathbf{y} = \sum_{i=1}^{N} g_i \mathbf{y}_i \qquad (3-1)$$

where

$\mathbf{y}$ is the output vector of the modular network,

$\mathbf{y}_i$ is the output vector of the i^{th} expert network,

g_i is the activation of the i^{th} output of the gating network and

N is the number of expert networks.

The modular network works in the following way. When a task is fed into the modular network, the expert and gating networks will receive that task simultaneously. The gating network receives the task and learns how to divide that task into several sub-tasks and assign each sub-task to an expert network. Meanwhile each expert network will learn to complete a target in that sub-task that is assigned by the gating network. These sub-tasks are combined together using equation (3 - 1). In other words, we can think of the gating network as a supervisor and each expert network as a worker. The supervisor divides the task into sub-tasks and assigns each sub-task to a worker. Then, each worker gets to work on one sub-task and they are combined at the end. In this scheme we assume that we do not know how to divide the task into sub-tasks.

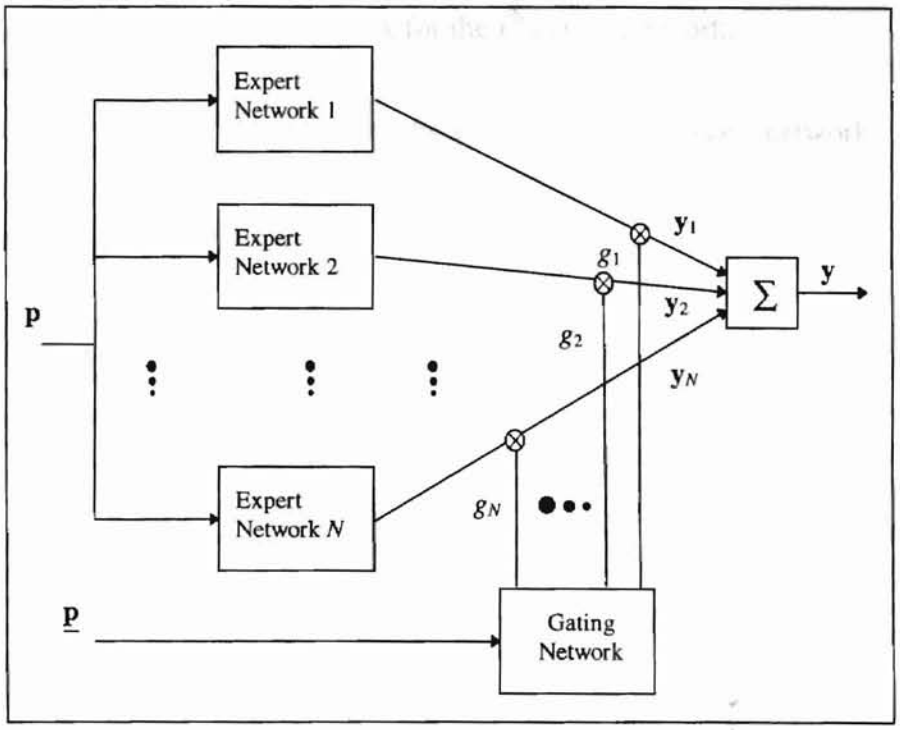


Figure 3 - 2. The What and Where Modular Network

If we have the information, $\underline{\mathbf{p}}$, of how the task should be divided, then we can use the modular network architecture as shown in Figure 3 - 2. This modular network architecture is called a what and where modular network, because each expert receives the actual tasks, **p**, (the what tasks) while the gating receives the classes of each task, $\underline{\mathbf{p}}$, (the where tasks). Typically, the inputs to the gating network are binary patterns. This architecture is also much easier to train as we will see in Chapter VII. Now, we would discuss the notation. Since we use a subscript to denote an expert network, there is a little different in denoting all the symbols of the expert network. When a symbol contains subscript, then the subscript is denotes the i^{th} expert network. When a symbol contains sub-subscript, then the sub-subscript denotes the i^{th} expert network. For example,

$\mathbf{W}_i$ denotes the weight matrix for the i^{th} expert network.

w_{j,k_i} denotes the *j*, *k* element of the weight for the i^{th} expert network.

A complete notation table is given in List of Symbols.

Expert Networks

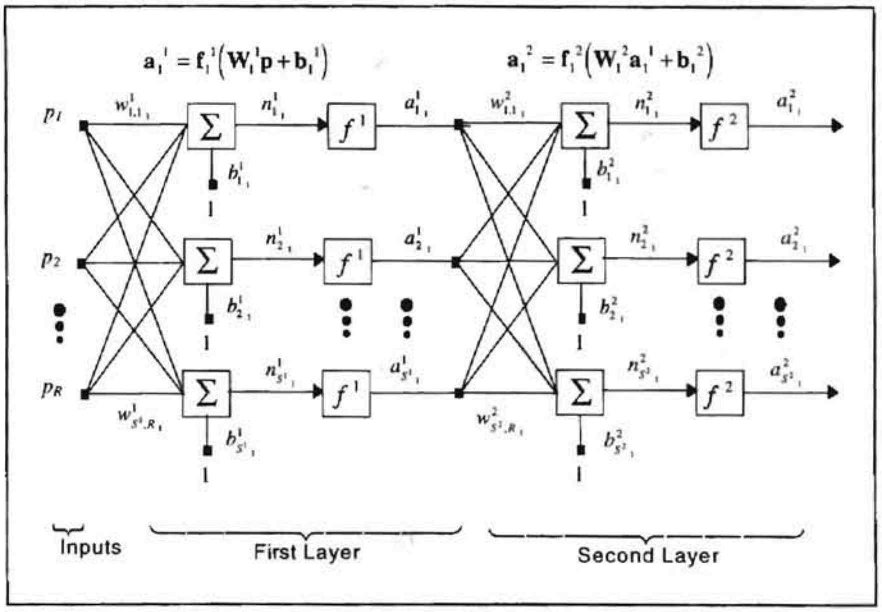


Figure 3 - 3. The Expert Network Architecture

The expert networks are networks that compete to learn the input patterns. Each expert network can operate in a different region to avoid crosstalk between other expert network. Typically, expert networks can be any kind of neural network, such as recurrent, multilayer perceptron or Kohonen. However, we will use multilayer neural networks as shown in Figure 3 - 3. There are two reasons why we use multilayer perceptrons instead of other networks. First, multilayer networks are conveniently trained by the backpropagation algorithm. By using multilayer network as expert networks, we have a systematic way of updating the weights and biases. Second, the multilayer networks are universal approximators [3]. They are very good in solving regression problems, and therefore they are suitable for control applications.

Gating Network

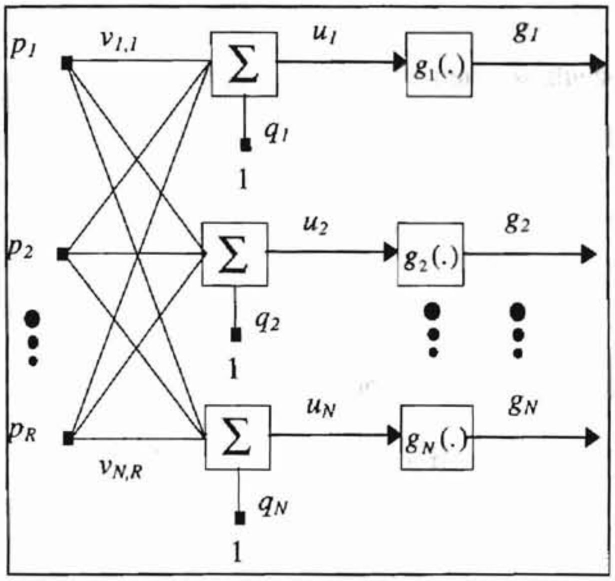


Figure 3 - 4. The Gating Network Architecture

The general architecture of the gating network is the same as the expert network, as shown in Figure 3 - 4 (any multilayer network would work). However, the gating network differs from the expert networks in two respects.

- The gating network has *N* output neurons, which is equal to the number of expert networks, whereas each expert network has *R* output neurons, which is equal to the dimension of the target output vector $\mathbf{y}^*$.
- In the final layer, the gating network uses a softmax activation function.

The softmax function is required to be the activation function for the gating network, because the g_i are interpreted as a priori probabilities [5] [6]. The interpretation of g_i generally requires the output of the gating network to satisfy two requirements:

$$0 \le g_i \le 1 \quad \text{for all } i. \quad \text{and} \quad \sum_{i=1}^{N} g_i = 1. \qquad (3-2)$$

To fulfill both constraints on the equation (3 - 2), we may use the softmax function:

$$g_i = \frac{\exp^{(u_i)}}{\sum_{j=1}^{N} \exp^{(u_j)}} \qquad (3-3)$$

To distinguish the gating network from the expert networks, we use a different notation in the gating network. The **V** and **q** represent the gating's weights and biases, respectively. The **u** and **g** represents the gating's net input vector and total output vector, respectively.

Modular Neural Network versus Fully Connected Network

Problems with Finding the Gradient in a Fully Connected Network

A multilayer neural network often has difficulty in learning any function from a finite amount of data. A particular case of this problem, as identified by Sutton[7], is called *temporal crosstalk*.[8] This phenomenon happens when a fully connected network is trained to learn one task and then switched to learn another task that is incompatible with the first. As a result, the network takes a longer time to learn the first task after it has learned the second task. Although it may eventually learn both tasks, it's learning speed and generalization ability are affected by the incompatible training data.

Another problem with a fully connected neural network is *spatial crosstalk*[9]. Many weights in a multilayer network affect the network response, even if the inputs range over only a small region of the input space. The network response is spread out over all of the elements of the network.

With the modular neural network architectures, the temporal crosstalk and spatial crosstalk problems are easily handled. If a block of incompatible data is presented, a modular network tends to allocate different networks to different blocks. Consequently, each network is immune to temporal crosstalk and spatial crosstalk since it only receives the data from a single task.

Furthermore, the modular networks can be structured more easily than a multilayer network, because it can contain a variety of types of network modules (networks with different topologies) that are appropriate for particular tasks.

Illustration of the Crosstalk Problem

In the following, a multilayer network and a modular network are used to train an absolute value function as shown in Figure 3 - 5. Both networks are trained until they achieve a sum of square error less than 10^{-3}. The multilayer network has a 1-10-1 architecture, and the modular network has the architecture that is shown in Figure 3 - 6. For faster convergence, both networks utilize the Marquardt-Levenberg algorithm during the training.

After both networks are trained, the input data in Figure 3 - 6 is split into two data sets, one set taken from the interval $[-1,0]$ and the other set taken from the interval $[0,1]$. Using the steepest descent algorithm on the multilayer network and the steepest ascent algorithm on the modular network, feed both networks alternately with the two data sets, where each data set trains for ten epochs. The purpose of training the network for ten epochs on each data set is to see whether the network will forget the second data while training on the first data. Hence, we measure the sum of square error of the whole data range, from $[-1,1]$. The learning rates in both networks are varied with 0.01, 0.001, 0.0001 and 0.00001.

Figure 3 - 7 shows the learning curve of the multilayer network showing the sum of squared error of the whole data set. As seen in Figure 3 - 7, the multilayer network suffers the crosstalk phenomena. It learns one set of data and forgetts the other. This raises the sum of squared error for a learning rate of 0.01. When the learning rate decreases, the crosstalk phenomena is less, but it does not learn. Meanwhile, Figure 3 - 8 shows the learning curve of the modular network. As seen in the plot, the sum of squared errors are decreasing and show no sign of crosstalk. Hence, the modular network is not as susceptible to the crosstalk problem as the multilayer network.

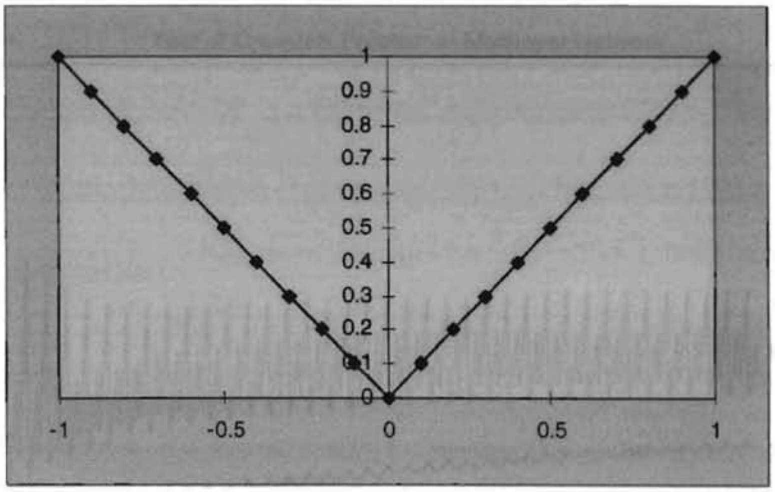


Figure 3 - 5. The Absolute Value

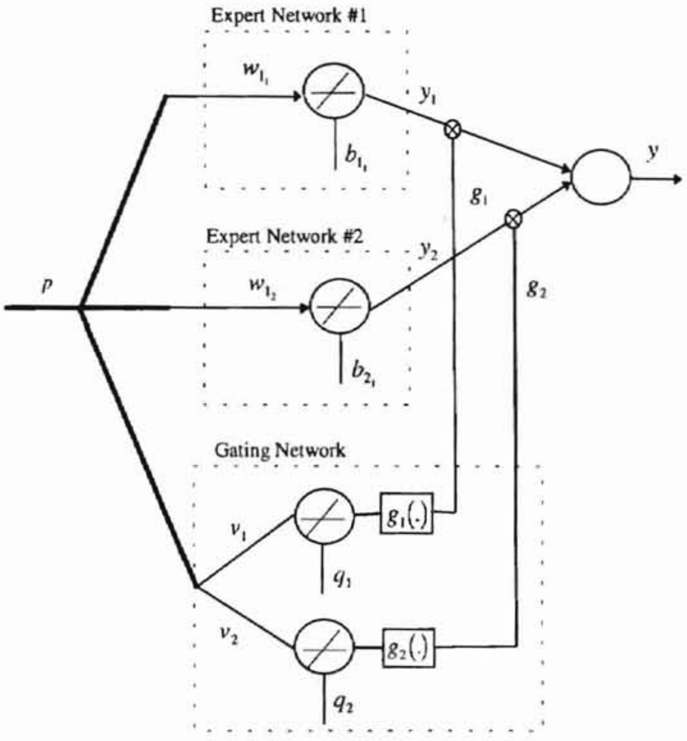


Figure 3 - 6. The Modular Network with 2 Single Layer Experts, and 1 Single Layer Gating

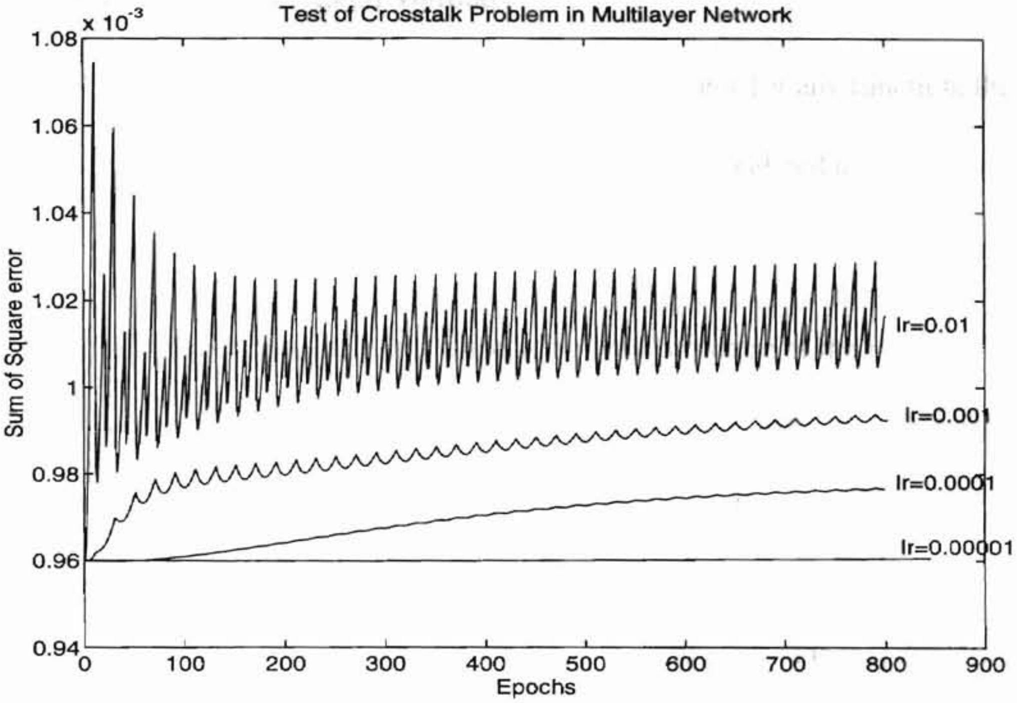


Figure 3 - 7. Test of Crosstalk Problem in Multilayer Network

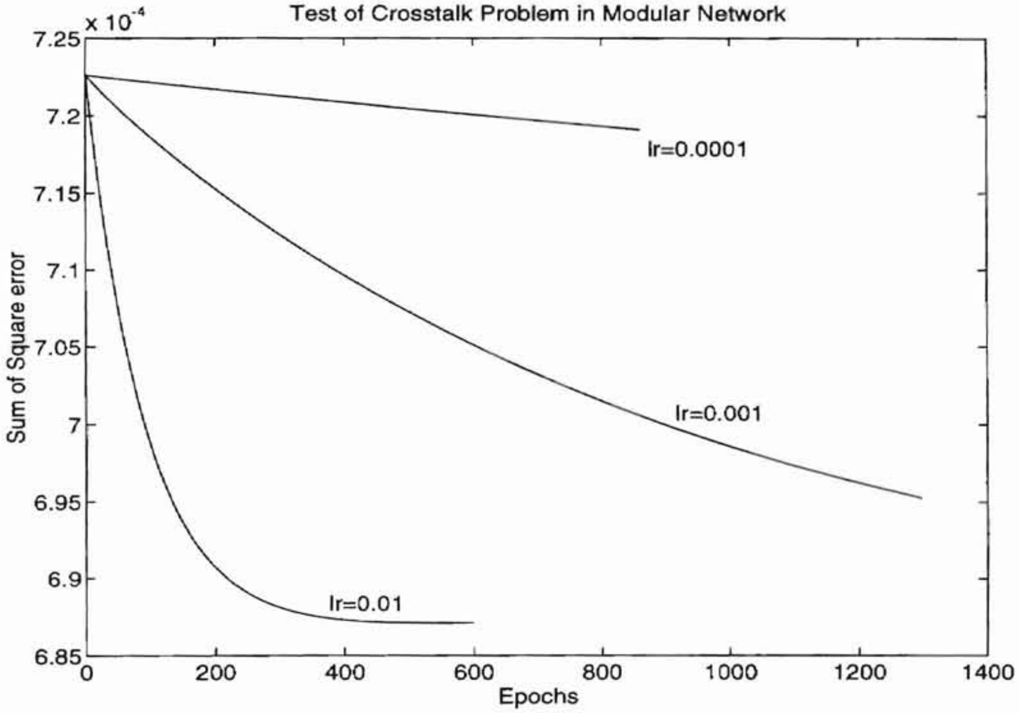


Figure 3 - 8. Test of Crosstalk Problem in Modular Network

Problems with Local and Global Methods

If we use a neural network as a universal approximator for any function, then an approximation of a prescribed input-output mapping may be realized using a *local* method or a *global* method.

A local method can capture the underlying local structure of the mapping. A model for this type of neural network would be the radial basis network, in which only a few neurons respond to any one input. This kind of realization method offers the advantage of fast learning, which requires very few training periods and offers an ability to operate in real-time. However, the disadvantage of local method is that they tend to be memory intensive and only capture the local structure, therefore they do not generalize well.

In contrast, a global method can capture the underlying global structure of the mapping. An example for this kind of neural network would be the multilayer perceptron. This kind of realization offers the advantage of better generalization performance and smaller storage requirements, but it has a relatively slow convergence and it is very difficult to interpret its representation.

The modular network provides a compromise between local and global methods, as it can capture an intermediate granularity.

Modular Network for Discontinuity Function

Consider the discontinuous function described by

$$F(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} > 0 \\ -1 & \mathbf{x} \le 0 \end{cases}, \qquad (3-4)$$

which is typical of the type of function that might be used to model coulomb friction. If we were to use a multilayer perceptron to approximate this function, the approximation may exhibit erratic behavior near the discontinuity. This erratic behavior is shown in Figure 3 - 9 with the dashed line. In this situation, it would be preferable to split the input function into two separate pieces and learn each piece separately. Hence, a modular network will provide a much better fit in this situation because it can decompose the input space into several sub-regions and then combine their individual solutions.

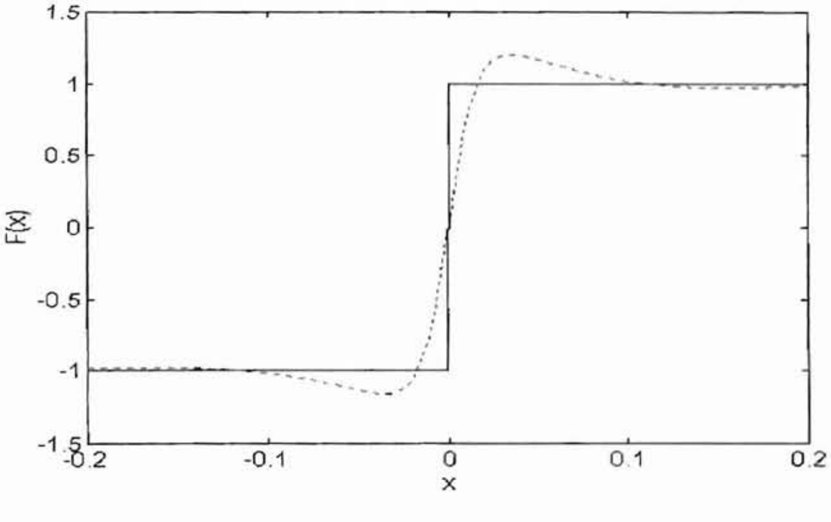


Figure 3 - 9. A discontinuous (piecewise linear) function and its approximation.

Illustration of Modularity

To illustrate how the modular network works, consider the modular network architecture shown in Figure 3 - 6. This architecture consists of 2 experts, each with a single layer linear transfer function, and a gating network. Suppose we want to approximate the absolute value function shown in Figure 3 - 5, then the nominal values for the weights and biases in the modular network are

$$w^1_{1_1} = -1; \qquad b^1_{1_1} = 0; \qquad w^1_{1_2} = 1; \qquad b^1_{1_2} = 0;$$

$$\mathbf{v} = \begin{bmatrix} +100 \\ -100 \end{bmatrix}; \qquad \mathbf{q} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

With these nominal weights, the output of the modular network will be an accurate approximation of the absolute value function. Figure 3 - 10 , Figure 3 - 11 and Figure 3 - 12 illustrate the effects of parameter changes on the modular network response. Unless otherwise noted, all the plots on the left hand side of the these figures are the gating network's output, g_1, and all the plots on the right hand side of these figures are the modular network's response. The U is varied with U=0.1 (dot line), 1 (dash-dot line), 10 (dash line), and 100 (solid line).

In Figure 3 - 10 (a) and (b), we vary the gating network's weights with $\mathbf{v} = \begin{bmatrix} +\mathrm{U} \\ -\mathrm{U} \end{bmatrix}$. This figure shows how the network weights can be used to strengthen the allocation of each region. In fact, the farther apart the +U and -U, the sharper the region is classified by the gating network. It is noteworthy that because of the softmax function, the distance between the +U and -U determines the strength to divide the region. For this reason, we will have the same network response with gating network's weights of $\mathbf{v}=\begin{bmatrix}200\\0\end{bmatrix}$, and $\mathbf{v}=\begin{bmatrix}+100\\-100\end{bmatrix}$. In Figure 3 - 10 (c) and (d), the gating network's weights vary with $\mathbf{v}=\begin{bmatrix}+\mathrm{U}\\-\mathrm{U}\end{bmatrix}$. By changing the sign of the gating network's weights, the gating network alternates its outputs and turns on the wrong expert network in each region. Consequently, the network response is the mirror image of Figure 3 - 10 (a) and (b).

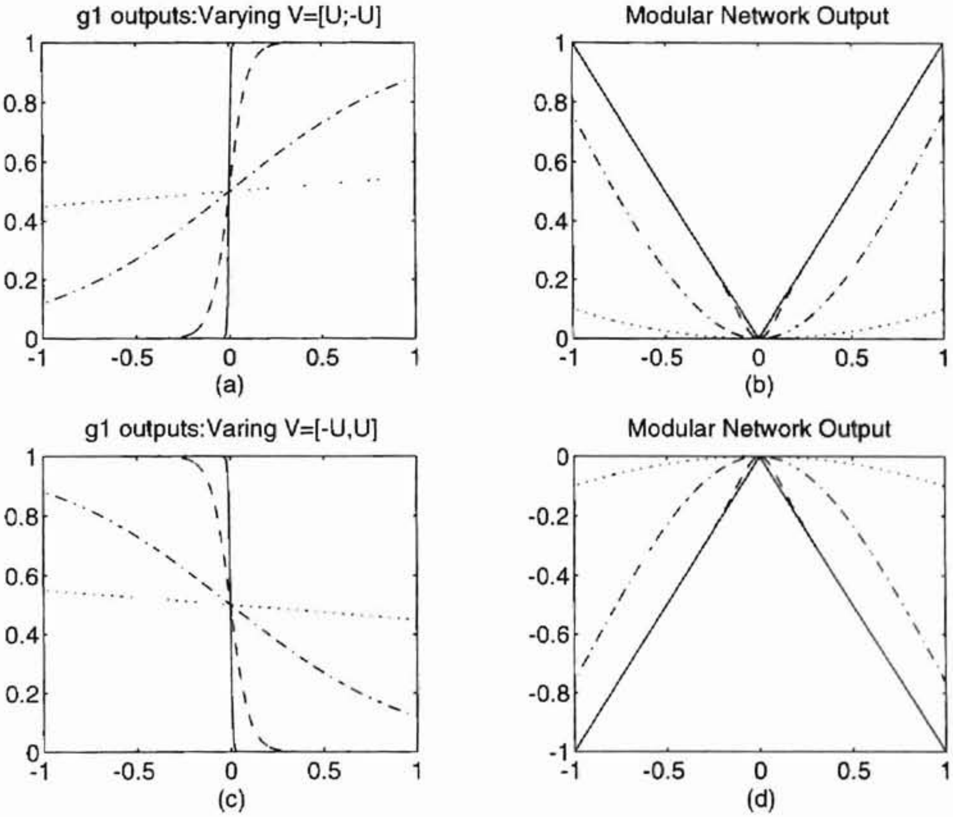


Figure 3 - 10. Effect of Parameter Changes on Network Response I

In Figure 3 - 11 the gating network's weights are fixed at $\mathbf{V} = \begin{bmatrix} +10 \\ -10 \end{bmatrix}$. In figure (a) and (b), the gating network's biases are varied with $\mathbf{q} = \begin{bmatrix} +\mathrm{U} \\ 0 \end{bmatrix}$, and in figure (c) and (d) the gating network's biases are varied with $\mathbf{q} = \begin{bmatrix} 0 \\ \mathrm{U} \end{bmatrix}$. By allowing the biases to change, this shifts the gating network's output to the left and right. Consequently, the gating network allows expert number 1 to gain more control in Figure (a) and (b) and to gain less control in Figure (c) and (d). In fact, at U=100, the gating network activates only one expert network. Due to the softmax function, the distance between the biases determines the strength of shifting the gating network's outputs. Hence, only the distance between the biases matters.

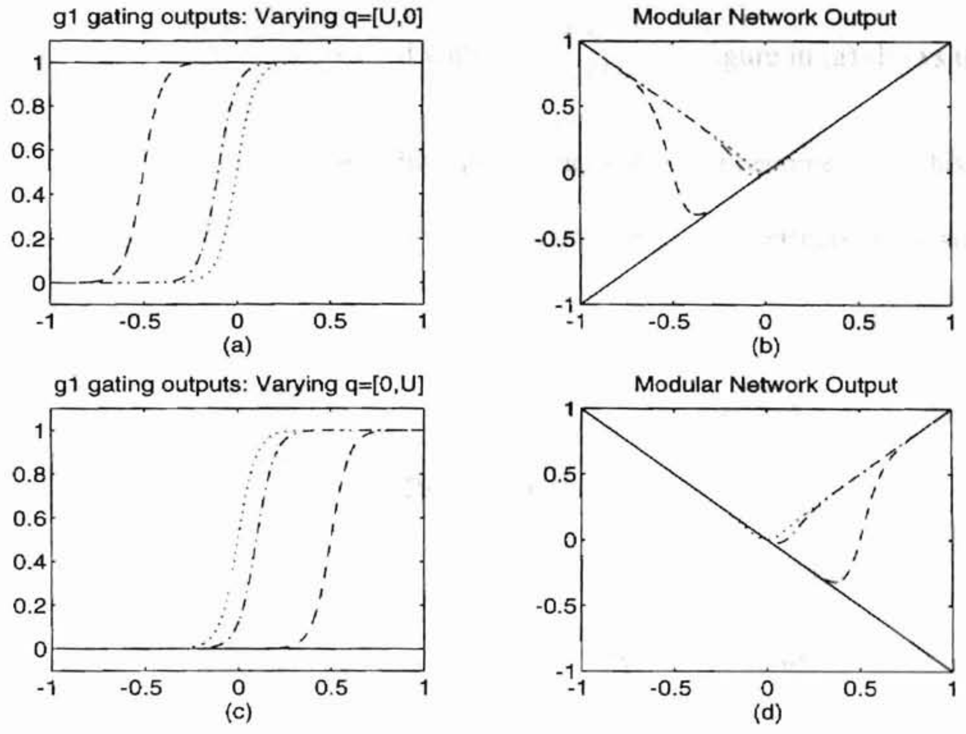


Figure 3 - 11. Effect of Parameter Changes on Network Response II

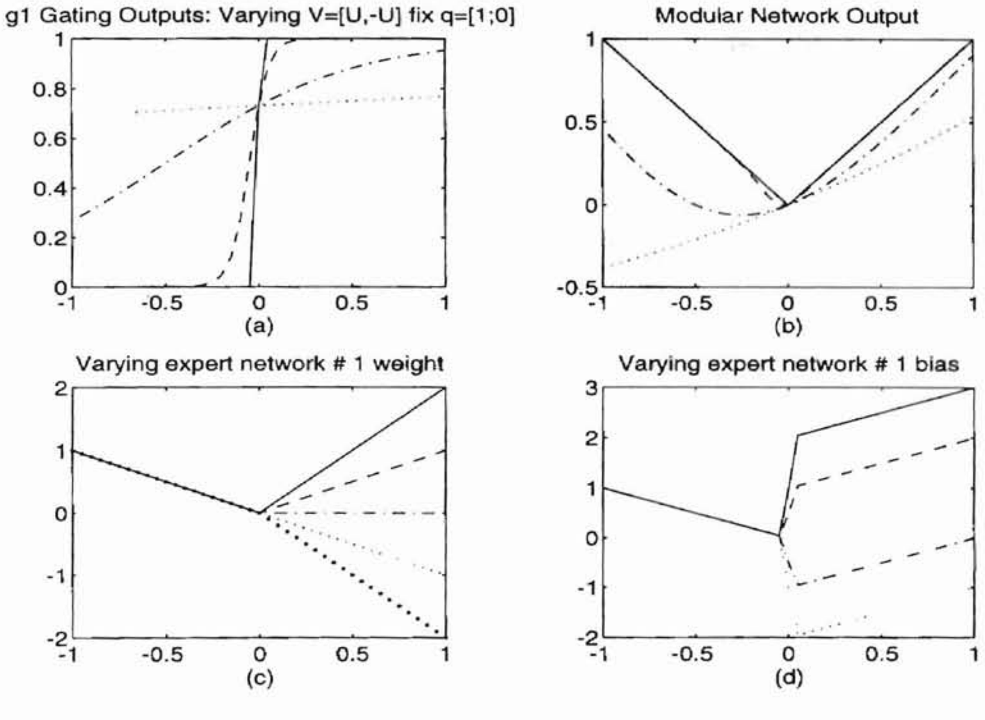


Figure 3 - 12. Effect of Parameter Changes on Network Response III

In Figure 3 - 12 (a) and (b), the gating network's biases are fixed at $\mathbf{q} = \begin{bmatrix} +1 \\ 0 \end{bmatrix}$, and the gating network's weights are varied with $\mathbf{v} = \begin{bmatrix} +\mathrm{U} \\ -\mathrm{U} \end{bmatrix}$. The figure in (a) shows that with increasing distances between the weights, the effects of shifting become less. This also means that the weights in the gating also contribute to the shifting effects and results in the network response as shown in figure (b).

In Figure 3 - 12 (c) and (d), we examine how the parameter changes in the expert network affects the network response. Both figure (c) and (d) show the modular network response. By varying the weights in expert number one by $w_{1_1}^{1} = -2,-1,0,1,2$, we change the slope of the network output response as shown in figure (c). However, this response only occurs in the region where the gating network turns on expert number one.

In Figure 3 - 12 (d), the biases of expert number one vary with $b_{1_1}^{1} = -2,-1,1,2$. As expected, the network output response shifts the slope up with positive biases and shifts the slope down with negative biases. This motion only occurs in the region where the gating network turns on expert number one.

CHAPTER IV

MODULAR NETWORK LEARNING RULES

The idea behind the modular network training is to allow both expert and gating networks to be trained simultaneously using the backpropagation algorithm to maximize the cost function--the log likelihood function. This cost function is formulated based on a statistical point of view and it is shown below:

$$J(\mathbf{x}) = \ln \sum_{i=1}^{N} g_i \exp^{-\frac{1}{2}\left(\mathbf{y}^{*}-\mathbf{y}_i\right)^T\left(\mathbf{y}^{*}-\mathbf{y}_i\right)} \qquad (4-1)$$

$\mathbf{x}$ is a parameter vector, typically represents all the weights and biases in a modular network.

$\mathbf{y}^{*}$ is the target output vector.

$\mathbf{y}_i$ is the output vector of the i^{th} expert network.

g_i is the activation of the i^{th} output of the gating network.

N is the number of expert networks.

Statistical Interpretation of Learning Algorithm in Modular Network

The objective of the learning algorithm in the modular network is to model the probability distribution of the training set, the statistical relationship between the input patterns **p** and the target patterns $\mathbf{y}^*$. It is assumed that the training patterns are generated by a number of different regressive processes in the following way. Assume that an input vector **p** is presented to the system that is being modeled, then the i^{th} expert network is chosen from a probability distribution conditioned on the input vector. According to the regressive process, the target output vector $\mathbf{y}^*$ can be generated by the i^{th} expert network:

$$\mathbf{y}^* = f_i(\mathbf{p}) + \varepsilon_i \qquad (4-2)$$

where

$\mathbf{y}^*$	the target output vector with dimension of *q*.
$f_i(\mathbf{p})$	the deterministic vector valued function of the input vector **p**.
ε_i	the Gaussian random vector with zero mean and covariance matrix σ^2I, where I is the identity matrix and σ^2 is the variance of the input vector, **p**.

The output of each expert network $\mathbf{y}_i$ is viewed as a conditional mean of a multivariate Gaussian distribution. Specifically, it is viewed as a conditional mean of the desired response $\mathbf{y}^*$ given the input vector **p** for the i^{th} expert network.

$$\begin{aligned}\mathbf{y}_i &= \mathbf{E}_i\left[\mathbf{y}^*/\mathbf{p}\right] \\ &= f_i(\mathbf{p}) \qquad (4-3)\end{aligned}$$

As shown from Wilks [10], the multivariate Gaussian distribution of the desired vector $\mathbf{y}^*$ given the input vector $\mathbf{p}$ at the i^{th} expert network may be expressed as

$$f_i\left(\mathbf{y}^*/\mathbf{p}\right) = \frac{1}{(2\pi)^{R/2}} \exp^{-\frac{1}{2}\left(\mathbf{y}^*-\mathbf{y}_i\right)^T\left(\mathbf{y}^*-\mathbf{y}_i\right)} \qquad (4-4)$$

where $-\frac{1}{2}\left(\mathbf{y}^*-\mathbf{y}_i\right)^T\left(\mathbf{y}^*-\mathbf{y}_i\right) = \left\|\mathbf{y}^*-\mathbf{y}_i\right\|$ and $\|.\|$ is the Euclidean norm of the enclosed vector. This multivariate Gaussian distribution is expressed as a conditional probability density function so that it emphasis the assumption that for a given vector **p**, the i^{th} expert network is producing the closest match to the target vector $\mathbf{y}^*$.

Base on the above assumption, the probability distribution of the target vector $\mathbf{y}^*$ may be expressed as a *mixture model* which is called *an associative Gaussian mixture model.*

$$\begin{aligned}f\left(\mathbf{y}^*/\mathbf{p}\right) &= \sum_{i=1}^{N} g_i f_i\left(\mathbf{y}^*/\mathbf{p}\right) \\ &= \frac{1}{(2\pi)^{R/2}} \sum_{i=1}^{N} g_i \exp^{-\frac{1}{2}\left(\mathbf{y}^*-\mathbf{y}_i\right)^T\left(\mathbf{y}^*-\mathbf{y}_i\right)}\end{aligned} \qquad (4-5)$$

If one were to view the output vector $\mathbf{y}_i$ of the i^{th} expert network as the synaptic weight vector $\mathbf{w}_i$ and the output of the gating network g_i as the activation of all the output neurons, with both $\mathbf{w}_i$ and g_i as the unknown free parameters, then the conditional probability density function $f(\mathbf{y}^*/\mathbf{p})$ may be view as a ***likelihood function***. However, it is preferable to work with natural logarithm of $f(\mathbf{y}^*/\mathbf{p})$, since the logarithm is a monotonic increasing function of its argument. Therefore, we may define the log-likelihood function as follows:

$$J(\mathbf{w}_i, g_i) = \ln\left[f\left(\mathbf{y}^*/\mathbf{p}\right)\right] \tag{4 - 6}$$

By substituting equation (4-5) into equation (4-6) and ignoring the constant term --$\ln(2\pi)^{R/2}$, the log-likelihood function is equivalent to the performance index or cost function of equation (4-1). By maximizing this objective cost function, the network would yield the maximum-likelihood estimate of all the unknown free parameters -- $\mathbf{y}_i$ and g_i. Take note that since $\mathbf{y}_i$ is depended on the expert's weights $\mathbf{w}_i$, and the g_i depends on the gating weights $\mathbf{v}$, the unknown free parameters can be viewed as all the weights and biases in the gating and expert networks. Several interpretations of these unknown free parameters have been given by Jacob and Jordan [9] [1]:

$\mathbf{y}_i$	expert networks' output vectors are the conditional mean of the multivariate Gaussian distributions and
g_i	gating network's output vectors are the prior probabilities of the i^{th} expert network generated by the current training patterns.

where all unknown parameters are conditioned on the input vector $\mathbf{p}$.

The activation, g_i, is selected such that the outputs of the gating network are constrained to satisfy two requirements:

$$0 \le g_i \le 1 \text{ for all } i, \text{ and} \tag{4-7}$$

$$\sum_{i=1}^{N} g_i = 1. \tag{4-8}$$

By satisfying these two constraints, we can interpret the activation, g_i, as a prior probability. To satisfy the constraints, we may define the activation, g_i, of the i^{th} output neuron of the gating network as the softmax function [5]:

$$g_i = \frac{\exp(u_i)}{\sum_{j=1}^{N} \exp(u_j)} \tag{4-9}$$

where u_i is the i^{th} output of the gating network.

Performance Optimization

To optimize the performance index $J(\mathbf{x})$ of equation (4-1), we will have to find the value of $\mathbf{x}$ which optimizes the $J(\mathbf{x})$. In other words, we will have to maximize the performance index with respects to all the weights and biases ($\mathbf{x}$) in the modular network.

$$J(\mathbf{x}) = J(x_1, x_2, \cdots, x_M). \tag{4-10}$$

We will assume that the performance index is analytical so that all the $\mathbf{x}$ derivatives exist. Then, we can represent the performance index $J(\mathbf{x})$ using Taylor series expansion about some nominal point $\mathbf{x}^*$.

$$
\begin{aligned}
J(\mathbf{x}) = J\left(\mathbf{x}^*\right) &+ \frac{\partial}{\partial x_1} J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(x_1 - x_1^*\right) + \frac{\partial}{\partial x_2} J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(x_2 - x_2^*\right) + \ldots \\
&+ \frac{\partial}{\partial x_M} J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(x_M - x_M^*\right) + \frac{1}{2}\frac{\partial^2}{\partial {x_1}^2} J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(x_1 - x_1^*\right)^2 \\
&+ \frac{1}{2}\frac{\partial^2}{\partial {x_2}^2} J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(x_2 - x_2^*\right)^2 + \ldots + \frac{1}{2}\frac{\partial^2}{\partial {x_M}^2} J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(x_M - x_M^*\right)^2 + \ldots
\end{aligned}
\qquad (4 - 11)
$$

We can also write the equation (4-11) into a matrix form as:

$$
\begin{aligned}
J(\mathbf{x}) = J\left(\mathbf{x}^*\right) &+ \nabla J(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}^*}\left(\mathbf{x} - \mathbf{x}^*\right) \\
&+ \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^*\right)^T \nabla^2 J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}\left(\mathbf{x} - \mathbf{x}^*\right) + \cdots
\end{aligned}
\qquad (4 - 12)
$$

where $\nabla J(\mathbf{x})$ is the gradient, defined as:

$$
\nabla J(\mathbf{x}) = \left[\frac{\partial}{\partial x_1} J(\mathbf{x}) \quad \frac{\partial}{\partial x_2} J(\mathbf{x}) \quad \cdots \quad \frac{\partial}{\partial x_M} J(\mathbf{x})\right]^T, \qquad (4 - 13)
$$

and $\nabla^2 J(\mathbf{x})$ is the Hessian defined as:

$$
\nabla^2 J(\mathbf{x}) = \begin{bmatrix}
\frac{\partial^2}{\partial {x_1}^2} J(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} J(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_M} J(\mathbf{x}) \\
\frac{\partial^2}{\partial x_2 \partial x_1} J(\mathbf{x}) & \frac{\partial^2}{\partial {x_2}^2} J(\mathbf{x}) & & \frac{\partial^2}{\partial x_2 \partial x_M} J(\mathbf{x}) \\
\vdots & & \ddots & \vdots \\
\frac{\partial^2}{\partial x_M \partial x_1} J(\mathbf{x}) & \frac{\partial^2}{\partial x_M \partial x_2} J(\mathbf{x}) & \ldots & \frac{\partial^2}{\partial {x_M}^2} J(\mathbf{x})
\end{bmatrix}^T . \qquad (4 - 14)
$$

By using the concept of Taylor series expansion, we will develop optimization techniques for the performance index on equation (4-1).

Since all optimization algorithms are iterative, we can begin with some initial guess, $\mathbf{x}_o$, (usually randomly selected) and as the algorithm iterates, update our initial guess in stages according to an equation of the form:

$\mathbf{x}_o$= initial guess (randomly selected), and

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k \, . \tag{4 - 15}$$

With equation (4-15), the Taylor series expansion can be rewritten to include iteration as:

$$\begin{aligned} J(\mathbf{x}_{k+1}) &= J(\mathbf{x}_k) + \nabla J(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_k}(\Delta\mathbf{x}_k) \\ &\quad + \frac{1}{2}(\Delta\mathbf{x}_k)^T \nabla^2 J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}(\Delta\mathbf{x}_k) + \cdots \end{aligned} \tag{4 - 16}$$

where $\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, or

$$J(\mathbf{x}_{k+1}) = J(\mathbf{x}_k + \Delta\mathbf{x}_k) = J(\mathbf{x}_k) + \mathbf{c}_k{}^T\Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k{}^T\mathbf{A}_k\Delta\mathbf{x}_k + \cdots \tag{4 - 17}$$

with

$$\mathbf{c}_k^T = \nabla J(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_k} \qquad \text{and} \qquad \mathbf{A}_k = \nabla^2 J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k} \, .$$

Based on the Taylor series expansion of equation (4-17), we will discuss two optimization techniques, the steepest ascent algorithm and the Marquardt-Levenberg algorithm. In the next section, we will discuss the steepest ascent algorithm and apply it to the training of the modular network. Then, in chapter IV, we will discuss the Marquardt-Levenberg techniques and incorporate them into the modular networks.

Steepest Ascent Algorithm

The steepest ascent algorithm is based on the first-order Taylor series expansion. This algorithm is the same as steepest descent except that we want the function $J(\mathbf{x})$ to increase instead of decrease at each iteration:

$$J(\mathbf{x}_{k+1}) > J(\mathbf{x}_k). \qquad (4 - 18)$$

If we expand $J(\mathbf{x}_{k+1})$ using Taylor series expansion, and consider $|\Delta\mathbf{x}_k|$ to be small, then the higher order terms in the Taylor series expansion on equation (4-17), will be negligible and the function can be approximated as:

$$J(\mathbf{x}_{k+1}) = J(\mathbf{x}_k + \Delta\mathbf{x}_k) \cong J(\mathbf{x}_k) + \mathbf{c}_k{}^T\Delta\mathbf{x}_k \qquad (4 - 19)$$

where $\mathbf{c}_k^T$ is the gradient evaluated at the old guess $\mathbf{x}_k$:

$$\mathbf{c}_k^T = \nabla J(\mathbf{x})^T\big|_{\mathbf{x}=\mathbf{x}_k} . \qquad (4 - 20)$$

For equation (4-18) to be true, we must satisfy the following equation:

$$\mathbf{c}_k{}^T\Delta\mathbf{x}_k > 0 . \qquad (4 - 21)$$

If we select

$$\Delta\mathbf{x}_k = \alpha_k\mathbf{p}_k , \qquad (4 - 22)$$

then

$$\mathbf{c}_k{}^T\Delta\mathbf{x}_k = \alpha_k\mathbf{c}_k{}^T\mathbf{p}_k > 0 . \qquad (4 - 23)$$

If we select an α_k that is greater than zero, then

$$\mathbf{c}_k{}^T\mathbf{p}_k > 0. \tag{4 - 24}$$

The vector $\mathbf{p}_k$ is called a direction vector and the α_k is called the learning rate. For any vector $\mathbf{p}_k$ that satisfies equation (4-24), the equation yields an ascent direction. Hence, if we take a small step in this ascent direction, the performance index, $J(\mathbf{x})$, is guaranteed to increase. However, the function increases most rapidly when equation (4-23) becomes most positive, therefore, we have to find a vector $\mathbf{p}_k$ which makes equation (4-24) most positive.

If we look at the inner product between the gradient and the direction vector, $\mathbf{c}_k{}^T\mathbf{p}_k$, we will notice that the $\mathbf{c}_k{}^T\mathbf{p}_k$ is most positive when the gradient, $\mathbf{c}_k{}^T$, has the same sign of the direction vector, $\mathbf{p}_k$. Hence,

$$\mathbf{p}_k = \mathbf{c}_k{}^T. \tag{4 - 25}$$

Substitute this equation (4-25) into equation, (4-22) and (4-15), yields

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{c}_k{}^T. \tag{4 - 26}$$

Equation (4-26) is the steepest ascent algorithm. It is the simplest optimization algorithm, but it can also be very slow. However, due to its simplicity, we can easily incorporate this algorithm into the modular network to give us some insight as to how well the modular network performs. In the next section, we will use this algorithm to adapt the modular network.

Adapting the Modular Network using Steepest Ascent

Recall the log-likelihood performance index:

$$J(\mathbf{x}) = \ln \sum_{i=1}^{N} g_i \exp^{-\frac{1}{2}\left(\mathbf{y}^* - \mathbf{y}_i\right)^T \left(\mathbf{y}^* - \mathbf{y}_i\right)} . \tag{4 - 27}$$

Substituting the softmax equation, (4-9), into equation (4-1), we obtain

$$J(\mathbf{x}) = \ln \sum_{i=1}^{N} \left(\frac{\exp^{(u_j)}}{\sum_{k=1}^{N} \exp^{(u_k)}} \right) \exp^{-\frac{1}{2}\left(\mathbf{y}^* - \mathbf{y}_i\right)^T \left(\mathbf{y}^* - \mathbf{y}_i\right)} . \tag{4 - 28}$$

We would like to maximize this performance index with respect to all the weights and biases, **x**, in the modular network, where

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_N \\ \mathbf{z} \end{bmatrix} \tag{4 - 29}$$

with N defined as the number of expert networks, $\mathbf{r}_i$ defined as all the weights, $\mathbf{w}_i$, and biases, $\mathbf{b}_i$, in the i^{th} expert network and $\mathbf{z}$ defined as all the weights, **v**, and biases, **q**, in the gating network.

$$\mathbf{r}_i = \left[w^1_{1,1_i} \quad w^1_{1,2_i} \quad \cdots \quad w^1_{S^1,R_i} \quad b^1_{1_i} \quad \cdots \quad b^1_{S^1_i} \quad w^2_{1,1_i} \quad \cdots \quad b^M_{S^M_i} \right]^T , \tag{4 - 30}$$

$$\mathbf{z} = \left[v^1_{1,1} \quad v^1_{1,2} \quad \cdots \quad v^1_{s^1,R} \quad q^1_1 \quad \cdots \quad q^1_{s^1} \quad v^2_{1,1} \quad \cdots \quad q^P_{s^P} \right]^T , \tag{4 - 31}$$

From equation (4-28), we can easily tell that the performance index is not a direct function of the weights and biases, **x**, of the modular network; it is, however, a direct function of the outputs of the expert networks, $\mathbf{y}_i$, and gating network, u_i. Hence, we can use the chain rule to relate the performance index with weights and biases of the modular network in the following fashion:

$$\frac{\partial J}{\partial \mathbf{x}}^T = \left[\frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1}\frac{\partial J}{\partial \mathbf{y}_1} \quad \frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_2}\frac{\partial J}{\partial \mathbf{y}_2} \quad \cdots \quad \frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_N}\frac{\partial J}{\partial \mathbf{y}_N} \quad \frac{\partial u_1}{\partial \mathbf{z}}\frac{\partial J}{\partial u_1} + \cdots + \frac{\partial u_N}{\partial \mathbf{z}}\frac{\partial J}{\partial u_N} \right]. \quad (4 - 32)$$

Take note that the above equation is not a matrix, it is one long vector. It would seem rather complicated to compute the above equation, but really what we need to calculate is

$$\frac{\partial J}{\partial \mathbf{r}_i} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\frac{\partial J}{\partial \mathbf{y}_i} \quad \text{for the } i^{\text{th}} \text{ expert network, and} \qquad (4 - 33)$$

$$\frac{\partial J}{\partial \mathbf{z}} = \sum_{i=1}^{N} \frac{\partial u_i}{\partial \mathbf{z}}\frac{\partial J}{\partial u_i} \quad \text{for the gating network.} \qquad (4 - 34)$$

Computing equation (4-32) is equivalent to computing the gradient of the performance index with respect to all weights and biases, **x**, in modular network.Once we have the gradient of the performance index, we can use the steepest ascent algorithm to update the modular network's weights and biases, **x**, in the following manner

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{c}_k{}^T \qquad \text{where} \qquad \mathbf{c}_k = \frac{\partial J}{\partial \mathbf{x}}^T . \qquad (4 - 35)$$

Equation (4-33) and (4-34) also indicate that we can adapt the expert networks and gating network simultaneously using the following pair of weight update schemes:

$$\mathbf{r}_{i_{k+1}} = \mathbf{r}_{i_k} + \alpha_k \frac{\partial J}{\partial \mathbf{r}_{i_k}} \tag{4 - 36}$$

and $$\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_k \frac{\partial J}{\partial \mathbf{z}_k}. \tag{4 - 37}$$

In the following sections, we will show how to compute equation (4-33) and (4-34) individually and update the weights using (4-36) and (4-37), respectively. However, to help in formulating the learning algorithm of the modular network, we will define the posterior probability.

Posterior Probability

The definition of the posterior probability associated with the i^{th} output of the expert network is defined as

$$h_i = \frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_i)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T(\mathbf{y}^* - \mathbf{y}_j)}}. \tag{4 - 38}$$

This probability is conditional on both the input vector, **p**, and the desired response vector $\mathbf{y}^*$. Hence, the posterior probabilities are generated by the current training pattern.

Adapting the Expert Networks

As mentioned in the previous section, to adapt the expert networks, we need to find the gradient of the performance index as:

$$\frac{\partial J}{\partial \mathbf{r}_i} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}^T \frac{\partial J}{\partial \mathbf{y}_i} \tag{4 - 39}$$

where

$$\mathbf{r}_i = \begin{bmatrix} w^1_{1,1_i} & w^1_{1,2_i} & \cdots & w^1_{S^1,R_i} & b^1_{1_i} & \cdots & b^1_{S^1_i} & w^2_{1,1_i} & \cdots & b^M_{S^M_i} \end{bmatrix}^T, \qquad (4\text{-}40)$$

$$\mathbf{y}_i = \begin{bmatrix} y_{1_i} & y_{2_i} & \cdots & y_{S^M_i} \end{bmatrix}^T. \qquad (4\text{-}41)$$

To find the second term of the equation (4-39), we will have to differentiate the performance index, *J*, with respect to the output vectors, $\mathbf{y}_i$, of the i^{th} expert network. If we do that, we will obtain the following partial derivative (see Appendix A-I for detail),

$$\frac{\partial J}{\partial \mathbf{y}_i} = h_i\left(\mathbf{y}^* - \mathbf{y}_i\right). \qquad (4\text{-}42)$$

This equation implies that, during the training process, the weights in the i^{th} expert network are updated in proportion to the posterior probability that it generated the current training patterns. Next, by choosing the expert network as a multilayer neural network, we can calculate the Jacobian term $\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}$ using the backpropagation algorithm and update the weights using steepest ascent algorithm. If we call this Jacobian term $\mathbf{K}_{(t,d)_i}$, then

$$\mathbf{K}_{(t,d)_i} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i} = \begin{bmatrix} \frac{\partial y_{1_i}}{\partial w^1_{1,1_i}} & \frac{\partial y_{1_i}}{\partial w^1_{1,2_i}} & \cdots & \frac{\partial y_{1_i}}{\partial w^1_{S^1,R_i}} & \frac{\partial y_{1_i}}{\partial b^1_{1_i}} & \cdots & \frac{\partial y_{1_i}}{\partial b^1_{S^1_i}} & \frac{\partial y_{1_i}}{\partial w^2_{1,1_i}} & \cdots & \frac{\partial y_{1_i}}{\partial b^M_{S^M_i}} \\ \frac{\partial y_{2_i}}{\partial w^1_{1,1_i}} & \frac{\partial y_{2_i}}{\partial w^1_{1,2_i}} & \cdots & \frac{\partial y_{2_i}}{\partial w^1_{S^1,R_i}} & \frac{\partial y_{2_i}}{\partial b^1_{1_i}} & \cdots & \frac{\partial y_{2_i}}{\partial b^1_{S^1_i}} & \frac{\partial y_{2_i}}{\partial w^2_{1,1_i}} & \cdots & \frac{\partial y_{2_i}}{\partial b^M_{S^M_i}} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial y_{S^M_i}}{\partial w^1_{1,1_i}} & \frac{\partial y_{S^M_i}}{\partial w^1_{1,2_i}} & \cdots & \frac{\partial y_{S^M_i}}{\partial w^1_{S^1,R_i}} & \frac{\partial y_{S^M_i}}{\partial b^1_{1_i}} & \cdots & \frac{\partial y_{S^M_i}}{\partial b^1_{S^1_i}} & \frac{\partial y_{S^M_i}}{\partial w^2_{1,1_i}} & \cdots & \frac{\partial y_{S^M_i}}{\partial b^M_{S^M_i}} \end{bmatrix}. \qquad (4\text{-}43)$$

Take note that *t* has a dimension of S^M in an i^{th} expert network, and *d* has a dimension of total number of weights and biases in i^{th} expert network.

Backpropagation in the Expert Networks

Consider Figure 4 - 1 as the i^{th} expert network in a modular network. The basic equations of the network are

$$\mathbf{y}_i^{\,0} = \mathbf{p}$$

$$\mathbf{y}_i^{\,m+1} = \mathbf{f}_i^{\,m+1}\left(\mathbf{W}_i^{\,m+1}\mathbf{y}_i^{\,m} + \mathbf{b}_i^{\,m+1}\right) \quad m = 0,1,\ldots,M-1. \qquad (4 - 44)$$

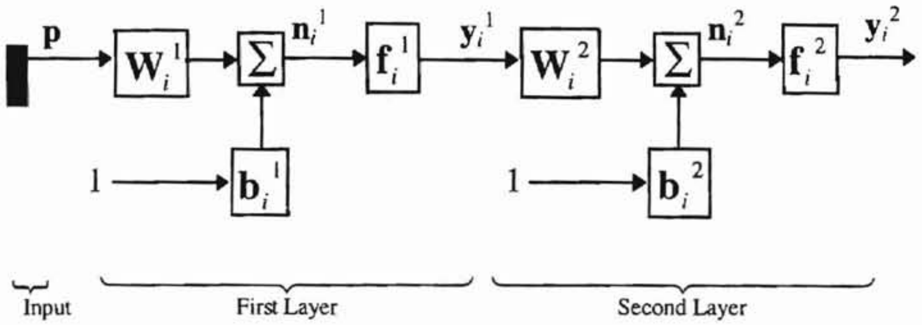


Figure 4 - 1. The i$^{\text{th}}$ Expert Network

Define

$$s^m_{j,t_i} = \frac{\partial y^M_{t_i}}{\partial n^m_{j_i}} \qquad (4 - 45)$$

as the sensitivity of the t^{th} element of the last layer output to a change in the net input of unit j in layer m of the i^{th} expert network. Then, by using the chain rule, it can be shown that:

$$\frac{\partial y^M_{t_i}}{\partial w^m_{j,k_i}} = \frac{\partial y^M_{t_i}}{\partial n^m_{j_i}} \times \frac{\partial n^m_{j_i}}{\partial w^m_{j,k_i}} \qquad (4 - 46)$$

$$\frac{\partial y^M_{t_i}}{\partial b^m_{j_i}} = \frac{\partial y^M_{t_i}}{\partial n^m_{j_i}} \times \frac{\partial n^m_{j_i}}{\partial b^m_{j_i}}. \qquad (4 - 47)$$

The first term in each of these equations is the sensitivity. The second term in each of these equations can be easily computed, since the net input to layer *m* is an explicit function of the weights and bias in that layer:

$$n_{j_i}^{m} = \sum_{k=1}^{m-1} w_{j,k_i}^{m} y_{k_i}^{m-1} + b_{j_i}^{m} \tag{4 - 48}$$

Therefore,

$$\frac{\partial n_{j_i}^{m}}{\partial w_{j,k_i}^{m}} = y_{k_i}^{m-1} \qquad \text{and} \qquad \frac{\partial n_{j_i}^{m}}{\partial b_{j_i}^{m}} = 1 . \tag{4 - 49}$$

Hence, we can compute the elements of the Jacobian using

$$\mathbf{K}_{(t,d)_i} = \frac{\partial y_{t_i}^{M}}{\partial w_{j,k_i}^{m}} = \frac{\partial y_{t_i}^{M}}{\partial n_{j_i}^{m}} \times \frac{\partial n_{j_i}^{m}}{\partial w_{j,k_i}^{m}} = s_{j,t_i}^{m} \times y_{k_i}^{m-1} , \tag{4 - 50}$$

and

$$\mathbf{K}_{(t,d)_i} = \frac{\partial y_{t_i}^{m}}{\partial b_{j_i}^{m}} = \frac{\partial y_{t_i}^{M}}{\partial n_{j_i}^{m}} \times \frac{\partial n_{j_i}^{m}}{\partial b_{j_i}^{m}} = s_{j,t_i}^{m} . \tag{4 - 51}$$

By backpropagating the sensitivity, it also can be shown that the sensitivity satisfies the following recurrence relation:

$$\mathbf{S}_i^{m} = \dot{\mathbf{F}}_i^{m}\left(\mathbf{n}_i^{m}\right)\mathbf{W}_i^{m+1^T}\mathbf{S}_i^{m+1} \tag{4 - 52}$$

where

$$\dot{\mathbf{F}}_i^m\left(\mathbf{n}_i^m\right) = \begin{bmatrix} \dot{f}_i^{\,m}\left(n_{1_i}^m\right) & 0 & \cdots & 0 \\ 0 & \dot{f}_i^{\,m}\left(n_{2_i}^m\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \dot{f}_i^{\,m}\left(n_{S^M_i}^m\right) \end{bmatrix} \tag{4 - 53}$$

and

$$\dot{f}_i^{\,m}(n) = \frac{\partial f_i^{\,m}(n)}{\partial n} \tag{4 - 54}$$

To start this recurrence relation, a boundary condition is needed. This is obtained at the last layer :

$$\mathbf{S}_i^{\,M} = \frac{\partial \mathbf{y}_i^{\,M}}{\partial \mathbf{n}_i^{\,M}} = \dot{\mathbf{F}}_i^{\,M}\left(\mathbf{n}_i^M\right) = \begin{bmatrix} f_i^{\,M}\left(n_{1_i}^M\right) & 0 & \cdots & 0 \\ 0 & f_i^{\,M}\left(n_{2_i}^M\right) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & f_i^{\,M}\left(n_{S^M_i}^M\right) \end{bmatrix}. \tag{4 - 55}$$

It is interesting that the backpropagation on the expert network does not involve any error term. The entire i^{th} expert network is trained by the posterior probabilities generated by the current training patterns.

After we have backpropagated through the i^{th} expert network and obtained the Jacobian, the next step is to multiply the Jacobian with the $\frac{\partial J}{\partial \mathbf{y}_i}$ term which we have already calculated in equation (4-42). Hence, the gradient of the performance index can be expressed in the following pair of equations,

$$\nabla J(\mathbf{x}) = \frac{\partial y_{t_i}}{\partial w_{j,k_i}^m} \times \frac{\partial J}{\partial y_{t_i}} = s_{j,t_i}^m y_{k_i}^{m-1} \times h_i\left(y_{t_i}^* - y_{t_i}^M\right), \text{ and} \tag{4 - 56}$$

$$\nabla J(\mathbf{x}) = \frac{\partial y_{t_i}}{\partial b_{j_i}^m} \times \frac{\partial J}{\partial y_{t_i}} = s_{j,t_i}^m \times h_i\left(y_{t_i}^* - y_{t_i}^M\right). \tag{4 - 57}$$

Once we have the gradient, the weights and biases of the i^{th} expert network can be updated element by element using the following steepest ascent algorithm:

$$w_{j,k_i}^m(k+1) = w_{j,k_i}^m(k) + \alpha s_{j,t_i}^m y_{k_i}^{m-1} h_i\left(y_{t_i}^* - y_{t_i}^M\right) \tag{4 - 58}$$

$$b_{j_i}^m(k+1) = b_{j_i}^m(k) + \alpha s_{j,t_i}^m h_i\left(y_{t_i}^* - y_{t_i}^M\right). \tag{4 - 59}$$

Adapting the Gating Network

Since the modular network requires the weights to be updated simultaneously in both the expert and gating networks, the next step is to update the weights in the gating network. By differentiating the log-likelihood function with respect to the output of the gating network u_i, we obtained the following partial derivative (see Appendix A-II).

$$\frac{\partial J}{\partial u_i} = h_i - g_i \tag{4 - 60}$$

This equation implies that, during the training process, the weights of the i^{th} output neuron of the gating network g_i moves toward the posterior probability h_i and allows the i^{th} expert network generates the current training patterns. As with the expert network, the gating network is also a multilayer network. The only exception is that the transfer function on last layer of the gating network is a softmax function. Since a multilayer network is used as gating network, the weights and biases in the network can be easily updated using the backpropagation algorithm in the following fashion.

$$\frac{\partial J}{\partial \mathbf{z}} = \sum_{i=1}^{N} \frac{\partial u_i}{\partial \mathbf{z}} \frac{\partial J}{\partial u_i} \qquad (4-61)$$

If we express

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_N \end{bmatrix}^T, \qquad (4-62)$$

then we can show that

$$\frac{\partial J}{\partial \mathbf{z}} = \frac{\partial \mathbf{u}}{\partial \mathbf{z}}^T \frac{\partial J}{\partial \mathbf{u}}. \qquad (4-63)$$

The above equation is similar to the gradient calculation of the expert network. In fact, the calculation of the first term, $\frac{\partial \mathbf{u}}{\partial \mathbf{z}}$, is exactly the same as the expert network; that is by backpropagation. We can think of this term as the Jacobian matrix of the $(N+1)^{\text{th}}$ expert network; that is $\mathbf{K}_{(t,d)_i}$ with $i = N+1$. Let G denote $N+1$, then we have the following Jacobian matrix for the gating network:

$$\mathbf{K}_{(t,d)_G} = \frac{\partial \mathbf{u}}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial u_1}{\partial v_{1,1}^1} & \frac{\partial u_1}{\partial v_{1,2}^1} & \cdots & \frac{\partial u_1}{\partial v_{S^1,R}^1} & \frac{\partial u_1}{\partial q_1^1} & \cdots & \frac{\partial u_1}{\partial q_{S^1}^1} & \frac{\partial u_1}{\partial v_{1,1}^2} & \cdots & \frac{\partial u_1}{\partial q_{S^M}^M} \\ \frac{\partial u_2}{\partial v_{1,1}^1} & \frac{\partial u_2}{\partial v_{1,2}^1} & \cdots & \frac{\partial u_2}{\partial v_{S^1,R}^1} & \frac{\partial u_2}{\partial q_1^1} & \cdots & \frac{\partial u_2}{\partial q_{S^1}^1} & \frac{\partial u_2}{\partial v_{1,1}^2} & \cdots & \frac{\partial u_2}{\partial q_{S^M}^M} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial u_N}{\partial v_{1,1}^1} & \frac{\partial u_N}{\partial v_{1,2}^1} & \cdots & \frac{\partial u_N}{\partial v_{S^1,R}^1} & \frac{\partial u_N}{\partial q_1^1} & \cdots & \frac{\partial u_{S^M}}{\partial q_{S^1}^1} & \frac{\partial u_{S^M}}{\partial v_{1,1}^2} & \cdots & \frac{\partial u_{S^M}}{\partial q_{S^M}^M} \end{bmatrix}. \qquad (4-64)$$

The second term, which we now express in vector form, is just a stack of individual elements that create one long vector,

$$\frac{\partial J}{\partial \mathbf{u}} = \begin{bmatrix} h_1 - g_1 \\ h_2 - g_2 \\ \vdots \\ h_N - g_N \end{bmatrix}. \qquad (4 - 65)$$

Backpropagation in Gating Network

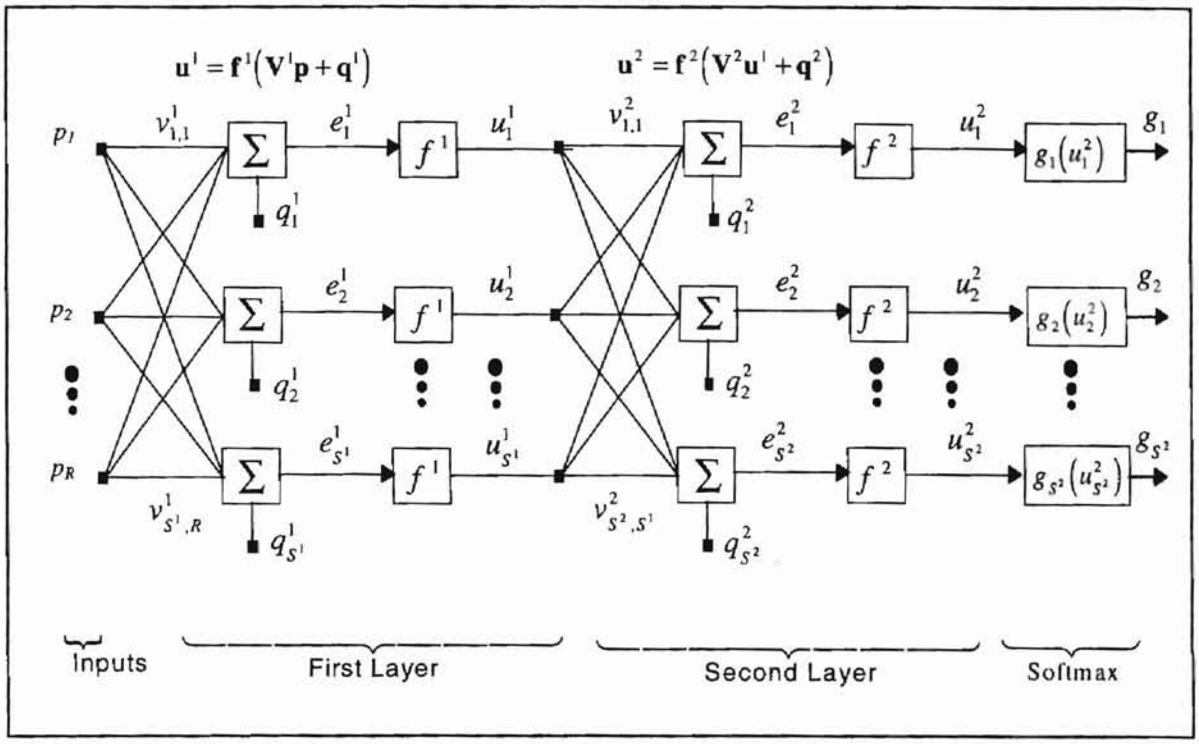


Figure 4 - 2 A Two Layered Gating Network

As shown in Figure 4 - 2, the gating network contains a multilayer network and a softmax function on the last layer. Although the gating network differs from the expert network by having a softmax function on the output layer, the backpropagation algorithm in the gating network is still exactly the same as the expert network. The reason is because we are backpropagating the output of the gating network, **u**. The only difference is the variables' name, where the weights, biases, net inputs and outputs are represented by $\mathbf{V}^m$, $\mathbf{q}^m{}_j$, $\mathbf{e}^m$, and $\mathbf{u}^m$. Now, let's summarize the equations used in backpropagation. First, we calculate the basic feedforward equations of the network:

$$\mathbf{u}^0 = \mathbf{p}$$

$$\mathbf{u}^{m+1} = \mathbf{f}^{m+1}\left(\mathbf{V}^{m+1}\mathbf{u}^m + \mathbf{q}^{m+1}\right) \quad m = 0,1,\ldots,M-1. \qquad (4-66)$$

Then, we calculate the sensitivity at the output ,M^{th}, layer; that is

$$s^M_{j,t\,G} = \frac{\partial u_t^M}{\partial e_j^M}, \qquad (4-67)$$

or in matrix form:

$$\mathbf{S}^M_G = \frac{\partial \mathbf{u}^M}{\partial \mathbf{e}^M} = \dot{\mathbf{F}}^M\left(\mathbf{e}^M\right) = \begin{bmatrix} \dot{f}^M\left(e_1{}^M\right) & 0 & \cdots & 0 \\ 0 & \dot{f}^M\left(e_2{}^M\right) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^M\left(e_{S^M}{}^M\right) \end{bmatrix} . \qquad (4-68)$$

Next, we calculate the sensitivity for the previous layer using the following recurrence relation:

$$\mathbf{S}^m_G = \dot{\mathbf{F}}^m\left(\mathbf{e}^m\right)\mathbf{V}^{m+1^T}\mathbf{S}^{m+1}_G \qquad (4-69)$$

where

$$\dot{\mathbf{F}}^m\left(\mathbf{e}^m\right) = \begin{bmatrix} \dot{f}^m\left(e_1{}^m\right) & 0 & \cdots & 0 \\ 0 & \dot{f}^m\left(e_2{}^m\right) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m\left(e_{S^m}{}^m\right) \end{bmatrix} \qquad (4-70)$$

and

$$\dot{f}^m(e) = \frac{\partial f^m(e)}{\partial e}. \qquad (4-71)$$

Repeat the recurrence relation until we find all the sensitivities in the layer. Then, by using the chain rule, we can compute the Jacobian matrix as:

$$\mathbf{K}_{(t,d)_G} = \frac{\partial u_t^M}{\partial v_{j,k}^m} = \frac{\partial u_t^M}{\partial e_j^m} \times \frac{\partial e_j^m}{\partial v_{j,k}^m} = s_{j,t_G}^m \times u_k^{m-1}, \qquad (4 - 72)$$

and

$$\mathbf{K}_{(t,d)_G} = \frac{\partial u_t^m}{\partial q_j^m} = \frac{\partial u_t^M}{\partial e_j^m} \times \frac{\partial e_j^m}{\partial q_j^m} = s_{j,t_G}^m . \qquad (4 - 73)$$

Since we know that

$$\frac{\partial J}{\partial u_i} = h_i - g_i$$

from Appendix (A-II), we can update the weights and biases using the following pair of equations:

$$v_{j,k}{}^m(k+1) = v_{j,k}{}^m(k) + \alpha s_{j,t_G}^m \left(u_k{}^{m-1}\right)\left(h_t - g_t\right) \qquad (4 - 74)$$

$$q_j{}^m(k+1) = q_j{}^m(k) + \alpha s_{j,t_G}^m \left(h_t - g_t\right). \qquad (4 - 75)$$

Like the expert network, the backpropagation on the gating network does not involve any error term. This implies that the training of the gating network is solely dependent on the term $\frac{\partial J}{\partial u_i}$ which equals $h_i - g_i$, and can be interpreted as follows. During the training process, the synaptic weights of the i^{th} output neurons of the gating network g_i move toward the posterior probability h_i and allows the i^{th} expert network to generate the current training patterns.

Example: Steepest Ascent Learning in Modular Networks

In the following, we will show an example of how the weights and biases are updated.

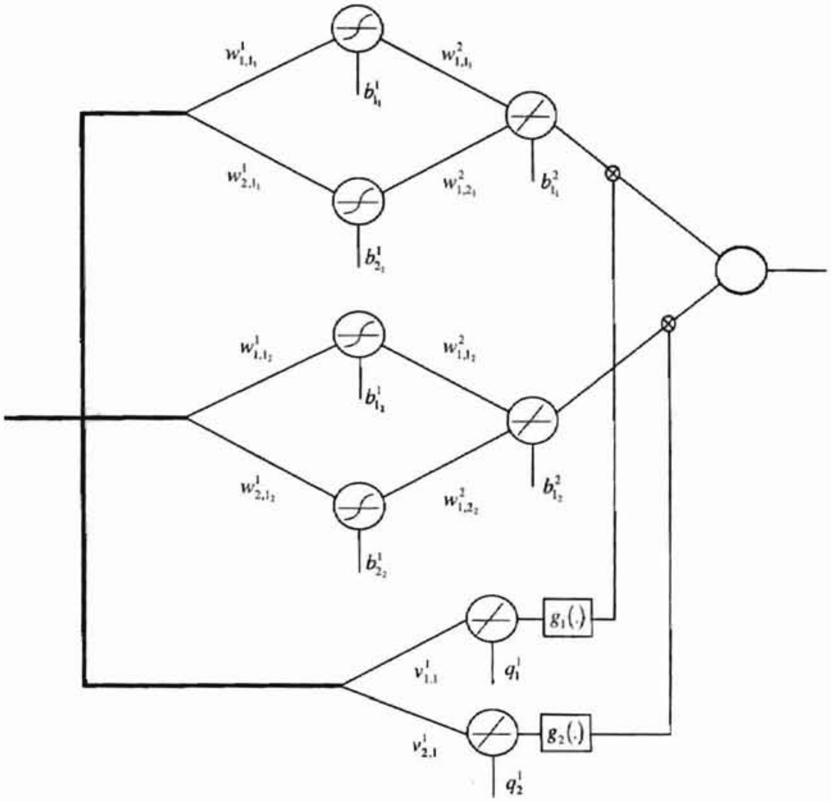


Figure 4 - 3 The Modular Network with Two 1-2-1 Experts, and 1-2 Gating

Figure 4 - 3 shows a modular network with two expert networks and a gating network. Each expert is a 1-2-1 two-layer network with hyperbolic tangent activation (**tansig**) functions in the first layer and linear activation (**purelin**) functions in the output. The gating network is a 1-2 network with a single layer and a linear output function. Assume that the input pattern, *p*, and target pattern, *t*, are

$$\mathbf{p} = \left[p_1\right] = \left[1\right] \quad \text{and} \qquad \mathbf{y}^* = \left[t_1\right] = [0].$$

The weights and the biases of the experts and gating are initialized to:

$$\mathbf{W}_1{}^1 = \begin{bmatrix} w^1_{11_1} \\ w^1_{21_1} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}; \mathbf{b}_1{}^1 = \begin{bmatrix} b^1_{1_1} \\ b^1_{2_1} \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}; \mathbf{W}_1^2 = \begin{bmatrix} w^2_{11_1} & w^2_{12_1} \end{bmatrix} = \begin{bmatrix} -1 & 1 \end{bmatrix}; \mathbf{b}_1^2 = \begin{bmatrix} b^2_{1_1} \end{bmatrix} = [0];$$

$$\mathbf{W}_2^1 = \begin{bmatrix} w^1_{11_2} \\ w^1_{21_2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \mathbf{b}_2^1 = \begin{bmatrix} b^1_{1_2} \\ b^1_{2_2} \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}; \mathbf{W}_2^2 = \begin{bmatrix} w^2_{11_2} & w^2_{12_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}; \mathbf{b}_2^2 = \begin{bmatrix} b^2_{1_2} \end{bmatrix} = [1];$$

$$\mathbf{V}^1 = \begin{bmatrix} v^1_{11} \\ v^1_{21} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}; \mathbf{q}^1 = \begin{bmatrix} q^1_1 \\ q^1_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix};$$

First, we calculate the feedforward equation of the expert and gating networks using equation (4-44) and (4-66).

<u>For expert network #1</u>

$$\mathbf{n}_1^1 = \mathbf{W}_1^1\mathbf{p} + \mathbf{b}_1^1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}[1] + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\mathbf{y}_1^1 = \text{tansig}(\mathbf{W}_1^1\mathbf{p} + \mathbf{b}_1^1) = \frac{\exp^{(\mathbf{n}_1^1)} - \exp^{(-\mathbf{n}_1^1)}}{\exp^{(\mathbf{n}_1^1)} + \exp^{(-\mathbf{n}_1^1)}} = \begin{bmatrix} 0.9640 \\ 0.7616 \end{bmatrix}$$

$$\mathbf{y}_1 = \mathbf{y}_1^2 = \text{purelin}(\mathbf{W}_1^2\mathbf{y}_1^1 + \mathbf{b}_1^2) = \begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} 0.9640 \\ 0.7616 \end{bmatrix} + [0] = [-0.2024]$$

<u>For expert network #2</u>

$$\mathbf{n}_2^1 = \mathbf{W}_2^1\mathbf{p} + \mathbf{b}_2^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}[1] + \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

$$\mathbf{y}_2^1 = \text{tansig}(\mathbf{W}_2^1\mathbf{p} + \mathbf{b}_2^1) = \frac{e^{(\mathbf{n}_2^1)} - e^{(-\mathbf{n}_2^1)}}{e^{(\mathbf{n}_2^1)} + e^{(-\mathbf{n}_2^1)}} = \begin{bmatrix} 0.9640 \\ -0.7616 \end{bmatrix}$$

$$\mathbf{y}_2 = \mathbf{y}_2^2 = \text{purelin}(\mathbf{W}_2^2\mathbf{y}_2^1 + \mathbf{b}_2^2) = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 0.9640 \\ -0.7616 \end{bmatrix} + [1] = [1.2024]$$

<u>For gating network</u>

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \text{purelin}\left(\mathbf{V}^1\mathbf{p} + \mathbf{q}^1\right) = \begin{bmatrix} v_{11}^1 \\ v_{21}^1 \end{bmatrix}[p_1] + \begin{bmatrix} q_1^1 \\ q_2^1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}[1] + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$g_i = \frac{e^{u_i}}{\sum_{j=1}^{2} e^{u_j}}; \quad \mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = \frac{e^{\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}}}{e^{[0.5]} + e^{[0.5]}} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix};$$

The output of the modular network yields

$$\mathbf{y} = g_1\mathbf{y}_1 + g_2\mathbf{y}_2 = 0.5 \times (-0.2024) + 0.5 \times (1.2024) = 0.5.$$

To find the posterior probability, equation (4 - 38)

$$h_i = \frac{g_i e^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)}}{\sum_{i=1}^{2} g_i e^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)}},$$

we first find the $-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)$

$$-\frac{1}{2}(t_1 - \mathbf{y}_1)^T (t_1 - \mathbf{y}_1) = [-0.02045]; \qquad -\frac{1}{2}(t_1 - \mathbf{y}_2)^T (t_1 - \mathbf{y}_2) = [-0.723]$$

then,

$$g_1 e^{-\frac{1}{2}(t_1 - \mathbf{y}_1)^T (t_1 - \mathbf{y}_1)} = [0.49]; \qquad g_2 e^{-\frac{1}{2}(t_1 - \mathbf{y}_2)^T (t_1 - \mathbf{y}_2)} = [0.2427].$$

So,

$$h_1 = \frac{0.49}{0.49 + 0.2427} = 0.6687 \text{ , and}$$

$$h_2 = \frac{0.2427}{0.49 + 0.2427} = 0.3313 .$$

For first layer, the derivative of the hyperbolic tangent (tansig) transfer function is

$$\dot{f}^1(n) = \frac{d}{dn}\left(\frac{e^n - e^{-n}}{e^n + e^{-n}}\right) = \frac{(e^n + e^{-n})(e^n + e^{-n}) - (e^n - e^{-n})(e^n - e^{-n})}{(e^n + e^{-n})^2} = 1 - \left(\frac{e^n - e^{-n}}{e^n + e^{-n}}\right)^2 = 1 - (a^1)^2$$

and for second layer, the derivative of linear (purelin) transfer function is

$$\dot{f}^2(n) = \frac{d}{dn}(n) = 1.$$

To calculate the expert Jacobian, $\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}$, we would perform the backpropagation. The starting point is the second layer on each expert network.

Backpropagation on Expert Network # 1

Starting from the last layer, calculate the sensitivities using (4-55),

$$\mathbf{S}_1^2 = \dot{\mathbf{F}}_1^2\left(\mathbf{n}_1^2\right) = 1.$$

Then, relate it using the recurrence relation on equation (4-52),

$$\mathbf{S}_1^1 = \dot{\mathbf{F}}_1^1\left(\mathbf{n}_1^1\right)\left(\mathbf{W}_1^2\right)^T \mathbf{S}_1^2 = \begin{bmatrix} 1-a_{1_1}^1 & 0 \\ 0 & 1-a_{1_2}^1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} [1] = \begin{bmatrix} 1-(0.9640)^2 & 0 \\ 0 & 1-(0.7616)^2 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} [1] \, .$$

$$\mathbf{S}_1^1 = \begin{bmatrix} -0.0707 \\ 0.42 \end{bmatrix}$$

We obtain the Jacobian matrix using (4-50) and (4-51) as follows,

$$\frac{\partial \mathbf{y}_1}{\partial \mathbf{W}_1^1} = \mathbf{S}_1^1 \mathbf{p} = \begin{bmatrix} -0.0707 \\ 0.42 \end{bmatrix} [1] = \begin{bmatrix} -0.0707 \\ 0.42 \end{bmatrix}$$

$$\frac{\partial \mathbf{y}_1}{\partial \mathbf{b}_1^1} = \mathbf{S}_1^1 = \begin{bmatrix} -0.0707 \\ 0.42 \end{bmatrix}$$

$$\frac{\partial \mathbf{y}_1}{\partial \mathbf{W}_1^2} = \mathbf{S}_1^2 \mathbf{y}_1^1 = [1] \begin{bmatrix} 0.9640 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.9640 \\ 0 \end{bmatrix}$$

$$\frac{\partial \mathbf{y}_1}{\partial \mathbf{b}_1^2} = \mathbf{S}_1^2 = [1]$$

$$\mathbf{K}_1 = \frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1} = \left[\frac{\partial y_1^2}{\partial w_{11_1}^1} \quad \frac{\partial y_1^2}{\partial w_{21_1}^1} \quad \frac{\partial y_1^2}{\partial b_{1_1}^1} \quad \frac{\partial y_1^2}{\partial b_{2_1}^1} \quad \frac{\partial y_1^2}{\partial w_{11_1}^2} \quad \frac{\partial y_1^2}{\partial w_{12_1}^1} \quad \frac{\partial y_1^2}{\partial b_{1_1}^2} \right].$$

$$= \left[-0.0707 \quad 0.42 \quad -0.0707 \quad 0.42 \quad 0.9640 \quad 0 \quad 1 \right]$$

Backpropagation on Expert Network #2 will be the same as in Expert Network #1,

$$\mathbf{S}_2^2 = \dot{\mathbf{F}}_2^2\left(\mathbf{n}_2^2\right) = 1.$$

$$\mathbf{S}_2^1 = \dot{\mathbf{F}}_2^1\left(\mathbf{n}_2^1\right)\left(\mathbf{W}_2^2\right)^T \mathbf{S}_2^2 = \begin{bmatrix} 1 - a_{2_1}^1 & 0 \\ 0 & 1 - a_{2_2}^1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} [1] = \begin{bmatrix} 1 - (0.9640)^2 & 0 \\ 0 & 1 - (-0.7616)^2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} [1]$$

$$\mathbf{S}_1^1 = \begin{bmatrix} 0.0707 \\ 0.42 \end{bmatrix}$$

$$\frac{\partial \mathbf{y}_2}{\partial \mathbf{W}_2^1} = \mathbf{S}_2^1 \mathbf{p}^T = \begin{bmatrix} 0.0707 \\ 0.42 \end{bmatrix} [1]$$

$$\frac{\partial \mathbf{y}_2}{\partial \mathbf{b}_2^1} = \mathbf{S}_2^1 = \begin{bmatrix} 0.0707 \\ 1 \end{bmatrix}$$

$$\frac{\partial \mathbf{y}_2}{\partial \mathbf{W}_2^2} = \mathbf{S}_2^2 \mathbf{y}_2^{1\,T} = [1] \begin{bmatrix} 0.9640 \\ 0 \end{bmatrix}^T = \begin{bmatrix} 0.9640 & 0 \end{bmatrix}$$

$$\frac{\partial \mathbf{y}_2}{\partial \mathbf{b}_2^2} = \mathbf{S}_2^2 = [1].$$

Once we find the Jacobian matrix, find the $\frac{\partial J}{\partial \mathbf{y}_i}$:

$$\frac{\partial J}{\partial \mathbf{y}_1} = h_1\left(\mathbf{t} - \mathbf{y}_1\right) = [0.6687]\left([0] - [-0.2024]\right) = [0.1353]$$

$$\frac{\partial J}{\partial \mathbf{y}_2} = h_2(\mathbf{t} - \mathbf{y}_2) = [0.3313]([0] - [1.2024]) = [-0.3984].$$

Now, we can compute the elements of the gradient, $\frac{\partial J}{\partial \mathbf{W}_i^k}$:

For expert network #1:

$$\frac{\partial J}{\partial \mathbf{W}_1^1} = \left(\frac{\partial J}{\partial \mathbf{y}_1}\right)\frac{\partial \mathbf{y}_1}{\partial \mathbf{W}_1^1} = [0.1353]\begin{bmatrix} -0.0707 \\ 0.42 \end{bmatrix} = \begin{bmatrix} -0.0096 \\ 0.0568 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{b}_1^1} = \left(\frac{\partial J}{\partial \mathbf{y}_1}\right)\frac{\partial \mathbf{y}_1}{\partial \mathbf{b}_1^1} = [0.1353]\begin{bmatrix} -0.0707 \\ 0.42 \end{bmatrix} = \begin{bmatrix} -0.0096 \\ 0.0568 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{W}_1^2} = \left(\frac{\partial J}{\partial \mathbf{y}_1}\right)\frac{\partial \mathbf{y}_1}{\partial \mathbf{W}_1^2} = [0.1353]\begin{bmatrix} 0.9640 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1304 \\ 0 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{b}_1^2} = \left(\frac{\partial J}{\partial \mathbf{y}_1}\right)\frac{\partial \mathbf{y}_1}{\partial \mathbf{b}_1^2} = [0.1353][1] = [0.1353]$$

For expert network #2:

$$\frac{\partial J}{\partial \mathbf{W}_2^1} = \left(\frac{\partial J}{\partial \mathbf{y}_2}\right)\frac{\partial \mathbf{y}_2}{\partial \mathbf{W}_2^1} = [-0.3984]\begin{bmatrix} 0.0707 \\ 0.42 \end{bmatrix} = \begin{bmatrix} -0.02817 \\ -0.1673 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{b}_2^1} = \left(\frac{\partial J}{\partial \mathbf{y}_2}\right)\frac{\partial \mathbf{y}_1}{\partial \mathbf{b}_2^1} = [-0.3984]\begin{bmatrix} 0.0707 \\ 0.42 \end{bmatrix} = \begin{bmatrix} -0.02817 \\ -0.1673 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{W}_2^2} = \left(\frac{\partial J}{\partial \mathbf{y}_2}\right)\frac{\partial \mathbf{y}_2}{\partial \mathbf{W}_2^2} = [-0.3984]\begin{bmatrix} 0.9640 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.3841 \\ 0 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{b}_2^2} = \left(\frac{\partial J}{\partial \mathbf{y}_2}\right)\frac{\partial \mathbf{y}_2}{\partial \mathbf{b}_2^2} = [-0.3984][1] = [-0.3984]$$

Using steepest ascent, the weights in the experts are updated as follows:

$$\mathbf{W}_{i\ new}^{k} = \mathbf{W}_{i\ old}^{k} + \alpha\frac{\partial J}{d\mathbf{W}_i^k} \qquad \text{and}$$

$$\mathbf{b}_{i\ new}^{k} = \mathbf{b}_{i\ old}^{k} + \alpha\frac{\partial J}{d\mathbf{b}_i^k}$$

Meanwhile, the gradient for the gating network's is computed as follows:

$$\mathbf{S} = \dot{\mathbf{F}}(\mathbf{u}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \text{ with}$$

$$\mathbf{s}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{s}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{V}^1} = \mathbf{s}_1 p^T + \mathbf{s}_2 p^T = \begin{bmatrix} 1 \\ 0 \end{bmatrix}[1]^T + \begin{bmatrix} 0 \\ 1 \end{bmatrix}[1]^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{q}^1} = \mathbf{s}_1.1^T + \mathbf{s}_2.1^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{u}} = \begin{bmatrix} h_1 - g_1 \\ h_2 - g_2 \end{bmatrix} = \begin{bmatrix} 0.6687 - 0.5 \\ 0.3313 - 0.5 \end{bmatrix} = \begin{bmatrix} 0.1687 \\ -0.1687 \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{V}^1} = \frac{\partial J}{\partial u_i}\frac{\partial u_i}{\partial \mathbf{V}^1} = (h_i - g_i)p^T$$

$$\frac{\partial J}{\partial v_1^1} = \frac{\partial J}{\partial u_1}\frac{\partial u_1}{\partial v_1^1} = \left(h_1 - g_1\right)p^T = ([0.6687] - [0.5])[1] = [0.1687]$$

$$\frac{\partial J}{\partial v_2^1} = \frac{\partial J}{\partial u_2}\frac{\partial u_2}{\partial v_2^1} = \left(h_2 - g_2\right)p^T = ([0.3313] - [0.5])[1] = [-0.1687]$$

$$\frac{\partial J}{\partial \mathbf{V}^1} = \begin{bmatrix} 0.1687 \\ -0.1687 \end{bmatrix}$$

Using steepest ascent, the weights in the gating network are updated as follows:

$$\mathbf{V}^1{}_{new} = \mathbf{V}^1{}_{old} + \alpha \frac{\partial J}{\partial \mathbf{V}^1}$$

and

$$\mathbf{q}^1_{new} = \mathbf{q}^1_{old} + \alpha \frac{\partial J}{\partial \mathbf{q}^1}$$

After the weights and biases are updated, the process is repeated for the next iteration.

CHAPTER V

MARQUARDT-LEVENBERG OPTIMIZATION

Chapter IV has provided a basic description of the modular network architecture and its learning rule using the steepest ascent method. Unfortunately, steepest ascent is the slowest optimization method. In this section, we will describe a technique to speed up the learning process of the modular network using the Marquardt-Levenberg algorithm.

Marquardt-Levenberg in Multilayer Network

The Marquardt-Levenberg algorithm provides very fast training for multilayer perceptrons. It has been shown to be approximately 20 times faster than the steepest descent method [11] in a small multilayer perceptron. Since the Marquardt-Levenberg algorithm has worked so well for multilayer perceptrons, we would like to incorporate the Marquardt-Levenberg optimization technique into modular network training. The Marquardt-Levenberg modification for modular network training will be described in this Chapter.

Marquardt-Levenberg Technique

Unlike the steepest descent algorithm, the Marquardt-Levenberg algorithm is a variation of Newton's method. Hence, to explain the Marquardt-Levenberg Algorithm, we will start by explaining how Newton's method works. Newton's method is based on the second order Taylor series:

$$J(\mathbf{x}_{k+1}) = J(\mathbf{x}_k + \Delta\mathbf{x}_k) = J(\mathbf{x}_k) + \mathbf{c}_k{}^T\Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k{}^T\mathbf{A}_k\Delta\mathbf{x}_k \qquad (5-1)$$

with

$$\mathbf{c}_k^T = \nabla J(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_k} \quad \text{and} \qquad (5-2)$$

$$\mathbf{A}_k = \nabla^2 J(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}. \qquad (5-3)$$

The basic idea behind Newton's method is to locate the stationary point of the quadratic approximation for $J(\mathbf{x})$. In locating the stationary point, we will take the gradient of this function with respect to $\Delta\mathbf{x}_k$, and set it equal to zero:

$$\frac{\partial J(\mathbf{x}_{k+1})}{\partial \Delta\mathbf{x}_k} = \mathbf{c}_k + \mathbf{A}_k\Delta\mathbf{x}_k = 0. \qquad (5-4)$$

Solving for $\Delta\mathbf{x}_k$,

$$\Delta\mathbf{x}_k = -\mathbf{A}_k{}^{-1}\mathbf{c}_k. \qquad (5-5)$$

Hence, Newton's method is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k{}^{-1}\mathbf{c}_k. \qquad (5-6)$$

If $J(\mathbf{x})$ is a function with a strong maximum, then $\mathbf{A}_k$ is negative definite and Newton's method will maximize $J(\mathbf{x})$. However, one problem with Newton's method is that the

Hessian matrix $\mathbf{A}_k$ may not be invertible. To avoid this, the Marquardt-Levenberg modification is introduced [12].

$$\mathbf{G}_k = \mathbf{A}_k - \mu_k \mathbf{I}. \qquad (5-7)$$

and the Marquardt-Levenberg modification to Newton's method is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\mathbf{A}_k - \mu_k \mathbf{I}\right)^{-1} \mathbf{c}_k. \qquad (5-8)$$

To see how the $\mathbf{G}_k$ can be made invertible, suppose that the eigenvalues and eigenvectors of $\mathbf{A}_k$ are $\{-\lambda_1, -\lambda_2, \ldots, -\lambda_n\}$ and $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n\}$. Then,

$$\mathbf{G}_k \mathbf{z}_i = \left(\mathbf{A}_k - \mu_k \mathbf{I}\right)\mathbf{d}_i = \mathbf{A}_k \mathbf{d}_i - \mu_k \mathbf{d}_i = -\lambda_i \mathbf{d}_i - \mu_k \mathbf{d}_i = \left(-\lambda_i - \mu_k\right)\mathbf{d}_i . \qquad (5-9)$$

From this result, we can see that $\mathbf{G}_k$ has the eigenvalues of $\left(-\lambda_i - \mu_k\right)$ and has the same eigenvectors as $\mathbf{A}_k$. By increasing μ_k until $\left(-\lambda_i - \mu_k\right) < 0$, $\mathbf{G}_k$ can be made negative definite and will be invertible.

The μ_k has a very meaningful interpretation. As μ_k increases to very large value, the Marquardt-Levenberg algorithm approaches the steepest ascent method, and as μ_k decreases to zero, the algorithm becomes Newton's method. This Marquardt-Levenberg modification provides a nice compromise between the speed of Newton's method and the guaranteed convergence of steepest ascent.

Marquardt-Levenberg Modification to Modular Network

To use the Marquardt-Levenberg algorithm for Modular Networks, we have to modify the performance index, so that it will work on a window of data. Hence, the notations use in the Chapter are slightly different than in Chapter IV. For a complete listing of symbol notation, see Appendix D. Since the performance index of the modular network is the log-likelihood function described in equation (3-1), and proportional to the sum of the log likelihood function over the training set, we can define the performance index as the sum of the log likelihood functions over the training set,

$$J = \sum_{q=1}^{Q} \ln \sum_{i=1}^{N} g_{i,q} \exp^{-\frac{1}{2}\left(\mathbf{y}_q^* - \mathbf{y}_{i,q}\right)^T \left(\mathbf{y}_q^* - \mathbf{y}_{i,q}\right)} . \qquad (5-10)$$

Let

$$J_q = \ln \sum_{i=1}^{N} g_{i,q} \exp^{-\frac{1}{2}\left(\mathbf{y}_q^* - \mathbf{y}_{i,q}\right)^T \left(\mathbf{y}_q^* - \mathbf{y}_{i,q}\right)} , \qquad (5-11)$$

then, the sum of the log-likelihood function over the training set becomes

$$J = \sum_{q=1}^{Q} J_q . \qquad (5-12)$$

The J_q is the performance index at q^{th} data. Since the performance index is a scalar, the subscript will always denotes the q^{th} data. To derive the Marquardt-Levenberg modification for Modular Networks, we need to find the Jacobian and Hessian of the performance index with respect to the all the weights in the network.

Gradient Calculation

We first note from appendix B that the total weights and biases of the modular network are defined as **x**, which contains the weights' and biases' of the experts, $\mathbf{r}_1 \ldots \mathbf{r}_N$, and gating, **z**, such that

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_N \\ \mathbf{z} \end{bmatrix}, \qquad (5-13)$$

where

$$\mathbf{r}_i = \begin{bmatrix} w^1_{1,1\,,i} & w^1_{1,2\,,i} & \cdots & w^1_{S^1,R\,,i} & b^1_{1\,,i} & \cdots & b^1_{S^1\,,i} & w^2_{1,1\,,i} & \cdots & b^M_{S^M\,,i} \end{bmatrix}^T \qquad (5-14)$$

contains the weights and biases in i^{th} expert network and

$$\mathbf{z} = \begin{bmatrix} v^1_{1,1} & v^1_{1,2} & \cdots & v^1_{S^1,R} & q^1_1 & \cdots & q^1_{S^1} & v^2_{1,1} & \cdots & q^M_{S^M} \end{bmatrix}^T. \qquad (5-15)$$

Then, the total gradient for the modular network is

$$\frac{\partial J}{\partial \mathbf{x}} = \begin{bmatrix} \sum_{q=1}^{Q} \frac{\partial J_q}{\partial \mathbf{r}_1} \\ \sum_{q=1}^{Q} \frac{\partial J_q}{\partial \mathbf{r}_2} \\ \vdots \\ \sum_{q=1}^{Q} \frac{\partial J_q}{\partial \mathbf{r}_N} \\ \sum_{q=1}^{Q} \frac{\partial J_q}{\partial \mathbf{z}} \end{bmatrix}. \qquad (5-16)$$

Now, define the total output of the network for the q^{th} input as $\mathbf{m}_q$, we have

$$\mathbf{m}_q = \begin{bmatrix} \mathbf{y}_{1,q} \\ \mathbf{y}_{2,q} \\ \vdots \\ \mathbf{y}_{N,q} \\ \mathbf{u}_q \end{bmatrix} \text{ where } \mathbf{y}_{i,q} = \begin{bmatrix} y_{1_{i,q}} \\ y_{2_{i,q}} \\ \vdots \\ y_{S^M{}_{i,q}} \end{bmatrix} \text{ and } \mathbf{u}_q = \begin{bmatrix} u_{1,q} \\ u_{2,q} \\ \vdots \\ u_{N,q} \end{bmatrix}. \tag{5 - 17}$$

Since the total weights of the network, **x**, is an indirect function of the performance index, *J*, but is a direct function of the total output of the modular network, $\mathbf{m}_q$, we use the chain rule to relate them,

$$\frac{\partial J}{\partial \mathbf{x}} = \sum_{q=1}^{Q} \frac{\partial \mathbf{m}_q}{\partial \mathbf{x}}^T \frac{\partial J_q}{\partial \mathbf{m}_q}. \tag{5 - 18}$$

If we define

$$\mathbf{j} = \begin{bmatrix} J_1 & J_2 & \cdots & J_Q \end{bmatrix}^T, \tag{5 - 19}$$

then we can express equation (5-19) as

$$\frac{\partial J}{\partial \mathbf{x}} = \frac{\partial \mathbf{m}}{\partial \mathbf{x}}^T \frac{\partial \mathbf{j}}{\partial \mathbf{m}}. \tag{5 - 20}$$

Expanding this equation, we get

$$\frac{\partial J}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1} & \frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_2} & \cdots & \frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_N} & \frac{\partial \mathbf{y}_1}{\partial \mathbf{z}} \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_1} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_2} & & \frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_N} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{z}} \\ \vdots & & \ddots & \vdots & \vdots \\ \frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_1} & \frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_2} & \cdots & \frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_N} & \frac{\partial \mathbf{y}_N}{\partial \mathbf{z}} \\ \frac{\partial \mathbf{u}}{\partial \mathbf{r}_1} & \frac{\partial \mathbf{u}}{\partial \mathbf{r}_2} & \cdots & \frac{\partial \mathbf{u}}{\partial \mathbf{r}_N} & \frac{\partial \mathbf{u}}{\partial \mathbf{z}} \end{bmatrix}^T \begin{bmatrix} \frac{\partial \mathbf{j}}{\partial \mathbf{y}_1} \\ \frac{\partial \mathbf{j}}{\partial \mathbf{y}_2} \\ \vdots \\ \frac{\partial \mathbf{j}}{\partial \mathbf{y}_N} \\ \frac{\partial \mathbf{j}}{\partial \mathbf{u}} \end{bmatrix}. \tag{5 - 21}$$

The term $\frac{\partial \mathbf{j}}{\partial \mathbf{m}}$ is the derivative of the performance index with respect to the total outputs of each expert and gating network. This term is computed as

$$\frac{\partial \mathbf{j}}{\partial \mathbf{m}}^T = \left[\frac{\partial \mathbf{j}}{\partial \mathbf{y}_1} \ \ \frac{\partial \mathbf{j}}{\partial \mathbf{y}_2} \ \ \cdots \ \ \frac{\partial \mathbf{j}}{\partial \mathbf{y}_N} \ \ \frac{\partial \mathbf{j}}{\partial \mathbf{u}} \right]. \qquad (5-22)$$

where

$$\frac{\partial \mathbf{j}}{\partial \mathbf{y}_i} \equiv \left[\frac{\partial J_1}{\partial y_{1_{i,1}}} \ \ \frac{\partial J_1}{\partial y_{2_{i,1}}} \ \ \cdots \ \ \frac{\partial J_1}{\partial y_{S^M{}_{i,1}}} \ \ \frac{\partial J_2}{\partial y_{1_{i,2}}} \ \ \cdots \ \ \frac{\partial J_Q}{\partial y_{S^M{}_{i,Q}}} \right] \qquad (5-23)$$

and

$$\frac{\partial \mathbf{j}}{\partial \mathbf{u}} \equiv \left[\frac{\partial J_1}{\partial u_{1,1}} \ \ \frac{\partial J_1}{\partial u_{2,1}} \ \ \ \ \frac{\partial J_1}{\partial u_{N,1}} \ \ \frac{\partial J_2}{\partial u_{1,2}} \ \ \cdots \ \ \frac{\partial J_Q}{\partial u_{N,Q}} \right]. \qquad (5-24)$$

The $\frac{\partial \mathbf{j}}{\partial \mathbf{y}_i}$ and $\frac{\partial \mathbf{j}}{\partial \mathbf{u}}$ are supposed to be matrix but due to the arrangement in the Jacobian matrix, $\frac{\partial \mathbf{m}}{\partial \mathbf{x}}$, we have to pile them into vector form. The term $\frac{\partial \mathbf{m}}{\partial \mathbf{x}}$ is the Jacobian matrix; the partial derivative of the total outputs of each expert and gating network with respect to all the modular network's weights and biases. Since the individual expert network and gating network are independent of each other, the partial derivative of an i^{th} expert network output or a gating network output that is not with respect to its own networks' weights and biases is equal to zero. Hence, all the off-diagonal terms in the Jacobian matrix are equal to zero.

$$\frac{\partial \mathbf{m}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1} & 0 & \cdots & 0 & 0 \\ 0 & \frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_2} & & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_N} & \\ 0 & 0 & \cdots & & \frac{\partial \mathbf{u}}{\partial \mathbf{z}} \end{bmatrix}. \qquad (5 - 25)$$

Since the off-diagonal terms are zeros, we can calculate the gradient with the following matrix,

$$\frac{\partial J}{\partial \mathbf{x}} = \begin{bmatrix} \left(\frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1}\right)^T \frac{\partial \mathbf{j}}{\partial \mathbf{y}_1} \\ \left(\frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_2}\right)^T \frac{\partial \mathbf{j}}{\partial \mathbf{y}_2} \\ \vdots \\ \left(\frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_N}\right)^T \frac{\partial \mathbf{j}}{\partial \mathbf{y}_N} \\ \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \frac{\partial \mathbf{j}}{\partial \mathbf{u}} \end{bmatrix}. \qquad (5 - 26)$$

Similar to Chapter IV, we will call the Jacobian matrix in expert i as $\mathbf{K}_{(t,d)_i}$ and the Jacobian matrix in gating as $\mathbf{K}_{(t,d)_G}$ where we can view the gating as the $(N+1)^{\text{th}}$ expert; that is $i = G$ as N+1. To find each diagonal term of the Jacobian matrix in equation (5-22), we will need to use the backpropagation algorithm as described in Chapter IV with a little modification. This modification is instead of finding the Jacobian for one training data point, we need to find the Jacobian for a set of training data. Hence, the Jacobian matrix for the i^{th} expert network will be

$$\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}=\begin{bmatrix}
\frac{\partial y_{1_{i,1}}}{\partial w^{1}_{1,1_{i}}} & \frac{\partial y_{1_{i,1}}}{\partial w^{1}_{1,2_{i}}} & \cdots & \frac{\partial y_{1_{i,1}}}{\partial w^{1}_{S^1,R_{i}}} & \frac{\partial y_{1_{i,1}}}{\partial b^{1}_{1_{i}}} & \cdots & \frac{\partial y_{1_{i,1}}}{\partial b^{1}_{S^1_{i}}} & \frac{\partial y_{1_{i,1}}}{\partial w^{2}_{1,1_{i}}} & \cdots & \frac{\partial y_{1_{i,1}}}{\partial b^{1}_{S^M_{i}}} \\
\frac{\partial y_{2_{i,1}}}{\partial w^{1}_{1,1_{i}}} & \frac{\partial y_{2_{i,1}}}{\partial w^{1}_{1,2_{i}}} & \cdots & \frac{\partial y_{2_{i,1}}}{\partial w^{1}_{S^1,R_{i}}} & \frac{\partial y_{2_{i,1}}}{\partial b^{1}_{1_{i}}} & & \frac{\partial y_{2_{i,1}}}{\partial b^{1}_{S^1_{i}}} & \frac{\partial y_{2_{i,1}}}{\partial w^{2}_{1,1_{i}}} & & \frac{\partial y_{2_{i,1}}}{\partial b^{1}_{S^M_{i}}} \\
\vdots & & & \vdots & & & \vdots & & & \vdots \\
\frac{\partial y_{{S^M}_{i,1}}}{\partial w^{1}_{1,1_{i}}} & \frac{\partial y_{{S^M}_{i,1}}}{\partial w^{1}_{1,2_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,1}}}{\partial w^{1}_{S^1,R_{i}}} & \frac{\partial y_{{S^M}_{i,1}}}{\partial b^{1}_{1_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,1}}}{\partial b^{1}_{S^1_{i}}} & \frac{\partial y_{{S^M}_{i,1}}}{\partial w^{2}_{1,1_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,1}}}{\partial b^{1}_{S^M_{i}}} \\
\frac{\partial y_{1_{i,2}}}{\partial w^{1}_{1,1_{i}}} & \frac{\partial y_{1_{i,2}}}{\partial w^{1}_{1,2_{i}}} & & \frac{\partial y_{1_{i,2}}}{\partial w^{1}_{S^1,R_{i}}} & \frac{\partial y_{1_{i,2}}}{\partial b^{1}_{1_{i}}} & & \frac{\partial y_{1_{i,2}}}{\partial b^{1}_{S^1_{i}}} & \frac{\partial y_{1_{i,2}}}{\partial w^{2}_{1,1_{i}}} & & \frac{\partial y_{1_{i,2}}}{\partial b^{1}_{S^M_{i}}} \\
\vdots & & & \vdots & & & \vdots & & & \vdots \\
\frac{\partial y_{{S^M}_{i,2}}}{\partial w^{1}_{1,1_{i}}} & \frac{\partial y_{{S^M}_{i,2}}}{\partial w^{1}_{1,2_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,2}}}{\partial w^{1}_{S^1,R_{i}}} & \frac{\partial y_{{S^M}_{i,2}}}{\partial b^{1}_{1_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,2}}}{\partial b^{1}_{S^1_{i}}} & \frac{\partial y_{{S^M}_{i,2}}}{\partial w^{2}_{1,1_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,2}}}{\partial b^{1}_{S^M_{i}}} \\
\vdots & & & \vdots & & & \vdots & & & \vdots \\
\frac{\partial y_{{S^M}_{i,Q}}}{\partial w^{1}_{1,1_{i}}} & \frac{\partial y_{{S^M}_{i,Q}}}{\partial w^{1}_{1,2_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,Q}}}{\partial w^{1}_{S^1,R_{i}}} & \frac{\partial y_{{S^M}_{i,Q}}}{\partial b^{1}_{1_{i}}} & & \frac{\partial y_{{S^M}_{i,Q}}}{\partial b^{1}_{S^1_{i}}} & \frac{\partial y_{{S^M}_{i,Q}}}{\partial w^{2}_{1,1_{i}}} & \cdots & \frac{\partial y_{{S^M}_{i,Q}}}{\partial b^{1}_{S^M_{i}}}
\end{bmatrix},\text{(5 - 27)}$$

and the Jacobian matrix for the gating network will be

$$\frac{\partial \mathbf{u}}{\partial \mathbf{z}}=\begin{bmatrix}
\frac{\partial u_{1,1}}{\partial v^{1}_{1,1}} & \frac{\partial u_{1,1}}{\partial v^{1}_{1,2}} & \cdots & \frac{\partial u_{1,1}}{\partial v^{1}_{S^1,R}} & \frac{\partial u_{1,1}}{\partial q^{1}_{1}} & \cdots & \frac{\partial u_{1,1}}{\partial q^{1}_{S^1}} & \frac{\partial u_{1,1}}{\partial v^{2}_{1,2}} & \cdots & \frac{\partial u_{1,1}}{\partial q^{M}_{S^M}} \\
\frac{\partial u_{1,2}}{\partial v^{1}_{1,1}} & \frac{\partial u_{1,2}}{\partial v^{1}_{1,2}} & \cdots & \frac{\partial u_{1,2}}{\partial v^{1}_{S^1,R}} & \frac{\partial u_{1,2}}{\partial q^{1}_{1}} & & \frac{\partial u_{1,2}}{\partial q^{1}_{S^1}} & \frac{\partial u_{1,2}}{\partial v^{2}_{1,2}} & & \frac{\partial u_{1,2}}{\partial q^{M}_{S^M}} \\
\vdots & & & \vdots & & & \vdots & & & \vdots \\
\frac{\partial u_{1.Q}}{\partial v^{1}_{1,1}} & \frac{\partial u_{1.Q}}{\partial v^{1}_{1,2}} & \cdots & \frac{\partial u_{1,Q}}{\partial v^{1}_{S^1,R}} & \frac{\partial u_{1,Q}}{\partial q^{1}_{1}} & \cdots & \frac{\partial u_{1,Q}}{\partial q^{1}_{S^1}} & \frac{\partial u_{1,Q}}{\partial v^{2}_{1,2}} & \cdots & \frac{\partial u_{1,Q}}{\partial q^{M}_{S^M}} \\
\frac{\partial u_{2,1}}{\partial v^{1}_{1,1}} & \frac{\partial u_{2,1}}{\partial v^{1}_{1,2}} & & \frac{\partial u_{2,1}}{\partial v^{1}_{S^1,R}} & \frac{\partial u_{2,1}}{\partial q^{1}_{1}} & & \frac{\partial u_{2,1}}{\partial q^{1}_{S^1}} & \frac{\partial u_{2,1}}{\partial v^{2}_{1,2}} & & \frac{\partial u_{2,1}}{\partial q^{M}_{S^M}} \\
\vdots & & & \vdots & & & \vdots & & & \vdots \\
\frac{\partial u_{2,Q}}{\partial v^{1}_{1,1}} & \frac{\partial u_{2,Q}}{\partial v^{1}_{1,2}} & \cdots & \frac{\partial u_{2,Q}}{\partial v^{1}_{S^1,R}} & \frac{\partial u_{2,Q}}{\partial q^{1}_{1}} & \cdots & \frac{\partial u_{2,Q}}{\partial q^{1}_{S^1}} & \frac{\partial u_{2,Q}}{\partial v^{2}_{1,2}} & \cdots & \frac{\partial u_{2,Q}}{\partial q^{M}_{S^M}} \\
\vdots & & & \vdots & & & \vdots & & & \vdots \\
\frac{\partial u_{S^M,Q}}{\partial v^{1}_{1,1}} & \frac{\partial u_{S^M,Q}}{\partial v^{1}_{1,2}} & \cdots & \frac{\partial u_{S^M,Q}}{\partial v^{1}_{S^1,R}} & \frac{\partial u_{S^M,Q}}{\partial q^{1}_{1}} & & \frac{\partial u_{S^M,Q}}{\partial q^{1}_{S^1}} & \frac{\partial u_{S^M,Q}}{\partial v^{2}_{1,2}} & \cdots & \frac{\partial u_{S^M,Q}}{\partial q^{M}_{S^M}}
\end{bmatrix}.\text{(5 - 28)}$$

When compare these Jacobian matrices on equation (5-27) and (5-28) with equations (4-43) and (4-62), we have modified them to include all the Jacobian terms for a entire training set by stacking them in one column. Each Jacobian matrix element for the i^{th} expert network can be calculated using the following pair of equations,

$$\mathbf{K}_{(t,d)_i} = \frac{\partial y_{t_{i,q}}}{\partial w^m_{l,k_i}} = \frac{\partial y_{t_{i,q}}}{\partial n^m_{l_{i,q}}} \times \frac{\partial n^m_{l_{i,q}}}{\partial w^m_{l,k_i}} = s^m_{l,t_{i,q}} \times y^{m-1}_{k_{i,q}}, \qquad (5-29)$$

$$\mathbf{K}_{(t,d)_i} = \frac{\partial y_{t_{i,q}}}{\partial b^m_{l_i}} = \frac{\partial y_{t_{i,q}}}{\partial n^m_{l_{i,q}}} \times \frac{\partial n^m_{l_{i,q}}}{\partial b^m_{l_i}} = s^m_{l,t_{i,q}}, \qquad (5-30)$$

where

$y_{t_{i,q}} = y^M_{t_{i,q}}$ is the t^{th} element of the output in the last layer of the i^{th} expert network and $y^{m-1}_{k_{i,q}}$ is the k^{th} element of the output in layer m-1 of the i^{th} expert network when input p_q is presented to the network.

Similarly, each Jacobian matrix element for the gating network can be calculated using the following pair of equations,

$$\mathbf{K}_{(t,d)_G} = \frac{\partial u^M_{t_q}}{\partial v^m_{l,k}} = \frac{\partial u^M_{t_q}}{\partial a^m_{l_q}} \times \frac{\partial a^m_{l_q}}{\partial v^m_{l,k}} = s^m_{l,t_{G,q}} \times u^{m-1}_{k_q}, \qquad (5-31)$$

$$\mathbf{K}_{(t,d)_G} = \frac{\partial u^m_{t_q}}{\partial q^m_l} = \frac{\partial u^M_{t_q}}{\partial a^m_{l_q}} \times \frac{\partial a^m_{l_q}}{\partial q^m_l} = s^m_{l,t_{G,q}}, \qquad (5-32)$$

where

$u_{t_q} = u_{t_q}^M$ is the t^{th} element of the output in the last layer of the gating network and

$u_{k_q}^{m-1}$ is the k^{th} element of the output in layer m-1 of the gating network when input p_q is presented to the network.

Hessian Matrix Computation

The Hessian matrix is calculated by taking the second derivative of the performance index, $J(\mathbf{x})$, with respect to the modular network parameters $\mathbf{x}$. Since the parameters are composed of the weights and biases in each expert and gating, the Hessian matrix will gives

$$\frac{\partial^2 J}{\partial \mathbf{x}^2} = \begin{bmatrix} \frac{\partial^2 J}{\partial \mathbf{r}_1{}^2} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{r}_1} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{r}_1} & \frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_1} \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{r}_2} & \frac{\partial^2 J}{\partial \mathbf{r}_2{}^2} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{r}_2} & \frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_2} \\ \vdots & \vdots & \ddots & & \vdots \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{r}_N} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{r}_N} & & \frac{\partial^2 J}{\partial \mathbf{r}_N{}^2} & \frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_N} \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{z}} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{z}} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{z}} & \frac{\partial^2 J}{\partial \mathbf{z}^2} \end{bmatrix}. \qquad (5-33)$$

As seen in the Hessian matrix, we can divide the matrix into four sub-matrices; that are the second derivative of the weights and biases in experts-experts, experts-gating, gating-experts and gating-gating.

Recall the calculation of gradient on equation (5-20),

$$\frac{\partial J}{\partial \mathbf{x}} = \frac{\partial \mathbf{m}}{\partial \mathbf{x}}^T \frac{\partial \mathbf{j}}{\partial \mathbf{m}}. \qquad (5-34)$$

By applying the results that the off-diagonal terms in the Jacobian matrix $\frac{\partial \mathbf{m}}{\partial \mathbf{x}}$ are equally to zero, we can also denote them with the following pair of equations:

$$\frac{\partial J}{\partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \frac{\partial \mathbf{j}}{\partial \mathbf{y}_i}, \qquad (5-35)$$

$$\frac{\partial J}{\partial \mathbf{z}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \frac{\partial \mathbf{j}}{\partial \mathbf{u}}. \qquad (5-36)$$

Hence, to calculate the Hessian matrix, we can compute the derivative of modular network's weights and biases with respect to the above pair of equations. It turns up that we have four second derivative terms to compute; that are $\frac{\partial^2 J}{\partial \mathbf{r}_j \partial \mathbf{r}_i}$, $\frac{\partial^2 J}{\partial \mathbf{z}^2}$, $\frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_i}$,and $\frac{\partial^2 J}{\partial \mathbf{r}_j \partial \mathbf{z}}$. These four second derivative terms are also the sub-matrices in the Hessian matrix. The detail of how these four second derivative terms are computed is shown in Appendix C. The results of these terms are summarized below:

$$\begin{aligned} \frac{\partial^2 J}{\partial \mathbf{r}_l \partial \mathbf{r}_i} &= \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{y}_l}\right)\left(\frac{\partial \mathbf{y}_l}{\partial \mathbf{r}_l}\right) & \text{if} \quad i \neq l \\ &= \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{y}_l}\right)\left(\frac{\partial \mathbf{y}_l}{\partial \mathbf{r}_l}\right) + \mathbf{H}(\mathbf{x}) & \text{if} \quad i = l \end{aligned} \qquad (5-37)$$

where

$$\mathbf{H}(\mathbf{x}) = \sum_j \frac{\partial^2 y_{i_j}}{\partial r_{i_m} \partial r_{i_k}} \frac{\partial J}{\partial y_{i_j}} = \frac{\partial^2 \mathbf{y}_i}{\partial \mathbf{r}_i^{\,2}} \frac{\partial J}{\partial \mathbf{y}_i}. \qquad (5-38)$$

$$\frac{\partial^2 J}{\partial \mathbf{z}^2} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{u}^2}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right) + \mathbf{B}(\mathbf{x}) \tag{5 - 39}$$

where

$$\mathbf{B}(\mathbf{x}) = \sum_j \frac{\partial^2 u_j}{\partial z_m \partial z_k} \frac{\partial J}{\partial u_j} = \frac{\partial^2 \mathbf{u}}{\partial \mathbf{z}^2} \frac{\partial J}{\partial \mathbf{u}}. \tag{5 - 40}$$

$$\frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{u}}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right). \tag{5 - 41}$$

$$\frac{\partial^2 J}{\partial \mathbf{r}_l \partial \mathbf{z}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_l \partial \mathbf{u}}\right)\left(\frac{\partial \mathbf{y}_l}{\partial \mathbf{r}_l}\right). \tag{5 - 42}$$

It is interesting to note that all four terms have a very similar solution. In fact, the Hessian matrix is different only on the diagonal terms where there are $\mathbf{H}(\mathbf{x})$ and $\mathbf{B}(\mathbf{x})$ added. If we assume the $\mathbf{H}(\mathbf{x})$ and $\mathbf{B}(\mathbf{x})$ terms are small and negligible, then equations (5-37) and (5-39) become

$$\frac{\partial^2 J}{\partial \mathbf{r}_l \partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{y}_l}\right)\left(\frac{\partial \mathbf{y}_l}{\partial \mathbf{r}_l}\right) \tag{5 - 43}$$

and

$$\frac{\partial^2 J}{\partial \mathbf{z}^2} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{u}^2}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right). \tag{5 - 44}$$

If we substitute these sub-matrices in equations (5-41), (5-42), (5-43), and (5-44) back into the Hessian matrix in equation (5-33) and reorganized them, we can obtain

$$\frac{\partial^2 J}{\partial \mathbf{x}^2} = \left(\frac{\partial \mathbf{m}}{\partial \mathbf{x}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{m}^2}\right) \frac{\partial \mathbf{m}}{\partial \mathbf{x}} \tag{5 - 45}$$

where

$$\frac{\partial^2 J}{\partial \mathbf{m}^2} = \begin{bmatrix} \frac{\partial^2 J}{\partial \mathbf{y}_1{}^2} & \frac{\partial^2 J}{\partial \mathbf{y}_2 \partial \mathbf{y}_1} & \cdots & \frac{\partial^2 J}{\partial \mathbf{y}_N \partial \mathbf{y}_1} & \frac{\partial^2 J}{\partial \mathbf{u} \partial \mathbf{y}_1} \\ \frac{\partial^2 J}{\partial \mathbf{y}_1 \partial \mathbf{y}_2} & \frac{\partial^2 J}{\partial \mathbf{y}_2{}^2} & \cdots & \frac{\partial^2 J}{\partial \mathbf{y}_N \partial \mathbf{y}_2} & \frac{\partial^2 J}{\partial \mathbf{u} \partial \mathbf{y}_2} \\ \vdots & \vdots & \ddots & & \vdots \\ \frac{\partial^2 J}{\partial \mathbf{y}_1 \partial \mathbf{y}_N} & \frac{\partial^2 J}{\partial \mathbf{y}_2 \partial \mathbf{y}_N} & & \frac{\partial^2 J}{\partial \mathbf{y}_N{}^2} & \frac{\partial^2 J}{\partial \mathbf{u} \partial \mathbf{y}_N} \\ \frac{\partial^2 J}{\partial \mathbf{y}_1 \partial \mathbf{u}} & \frac{\partial^2 J}{\partial \mathbf{y}_2 \partial \mathbf{u}} & \cdots & \frac{\partial^2 J}{\partial \mathbf{y}_N \partial \mathbf{u}} & \frac{\partial^2 J}{\partial \mathbf{u}^2} \end{bmatrix} \tag{5 - 46}$$

and

$$\frac{\partial \mathbf{m}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1} & 0 & \cdots & 0 & 0 \\ 0 & \frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_2} & & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_N} & \\ 0 & 0 & \cdots & & \frac{\partial \mathbf{u}}{\partial \mathbf{z}} \end{bmatrix}. \tag{5 - 47}$$

Each individual terms in the $\frac{\partial^2 J}{\partial \mathbf{m}^2}$ matrix are calculated as

$$\begin{aligned} \frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) &= h_i(-\mathbf{I}) + \left(\mathbf{y}^* - \mathbf{y}_i\right)\left(h_k - h_i h_k\right)\left(\mathbf{y}^* - \mathbf{y}_k\right)^T \qquad \text{if } i = k \\ &= -\left(\mathbf{y}^* - \mathbf{y}_i\right)\left(h_i h_k\right)\left(\mathbf{y}^* - \mathbf{y}_k\right)^T \qquad \text{if } i \neq k \end{aligned} \tag{5 - 48}$$

$$\begin{aligned}\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial u_i}\right) &= \left(h_i - h_i^{\,2}\right) - \left(g_i - g_i^{\,2}\right) && \text{if } \; i = j \\ &= h_i h_j + g_i g_j && \text{if } \; i \neq j\end{aligned} \qquad (5\text{-}49)$$

$$\begin{aligned}\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right) &= \left(h_i - h_i^{2}\right)\left(\mathbf{y}^{*} - \mathbf{y}_i\right)^{\mathrm{T}} && \text{if } \; i = j \\ &= -h_i h_j \left(\mathbf{y}^{*} - \mathbf{y}_j\right)^{\mathrm{T}} && \text{if } \; i \neq j\end{aligned} \qquad (5\text{-}50)$$

$$\begin{aligned}\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) &= \left(h_i - h_i^{2}\right)\left(\mathbf{y}^{*} - \mathbf{y}_i\right) && \text{if } \; i = j \\ &= -h_i h_j \left(\mathbf{y}^{*} - \mathbf{y}_i\right) && \text{if } \; i \neq j\end{aligned} \qquad (5\text{-}51)$$

The complete calculation detail are shown in Appendix C-V to Appendix C-VIII. It is interesting to note that the $\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right)$ and $\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right)$ are mirror image of each other. In fact, the $\frac{\partial^2 J}{\partial \mathbf{m}^2}$ matrix has all the upper right off diagonal terms mirror imaging all the lower left off diagonal terms. Let $\mathbf{e}_j \mathbf{e}_i^{\,T} = \left(\mathbf{y}^{*} - \mathbf{y}_j\right)\left(\mathbf{y}^{*} - \mathbf{y}_i\right)^{T}$, then

$$\frac{\partial^2 J}{\partial \mathbf{m}^2} = \left[\begin{array}{cccc}
-h_1\mathbf{I} + \left(h_1 - h_1^{\ 2}\right)\mathbf{e}_1^{\ 2^T} & -\left(h_2h_1\right)\mathbf{e}_2\mathbf{e}_1^{\ T} & \cdots & -\left(h_Nh_1\right)\mathbf{e}_N\mathbf{e}^{T} \\
-\left(h_1h_2\right)\mathbf{e}_1\mathbf{e}_2^{\ T} & -h_2\mathbf{I} + \left(h_2 - h_2^{\ 2}\right)\mathbf{e}_2^{\ 2^T} & & -\left(h_Nh_2\right)\mathbf{e}_N\mathbf{e}_2^{\ T} \\
\vdots & & \ddots & \vdots \\
-\left(h_1h_N\right)\mathbf{e}_1\mathbf{e}_N^{\ T} & -\left(h_2h_N\right)\mathbf{e}_2\mathbf{e}_N^{\ T} & \cdots & -h_N\mathbf{I} + \left(h_N - h_N^{\ 2}\right)\mathbf{e}_N^{\ 2^T} \cdots \\
\left(h_1 - h_1^{\ 2}\right)\mathbf{e}_1^{\ T} & -\left(h_2h_1\right)\mathbf{e}_1^{\ T} & \cdots & -\left(h_Nh_1\right)\mathbf{e}_1^{\ T} \\
-\left(h_1h_2\right)\mathbf{e}_2^{\ T} & \left(h_2 - h_2^{\ 2}\right)\mathbf{e}_2^{\ T} & & -\left(h_Nh_2\right)\mathbf{e}_2^{\ T} \\
\vdots & & \ddots & \vdots \\
-\left(h_1h_N\right)\mathbf{e}_N^{\ T} & -\left(h_2h_N\right)\mathbf{e}_N^{\ T} & \cdots & \left(h_N - h_N^{\ 2}\right)\mathbf{e}_N^{\ T}
\end{array}\right.$$

$$\left.\begin{array}{cccc}
\left(h_1 - h_1^{\ 2}\right)\mathbf{e}_1 & -\left(h_1h_2\right)\mathbf{e}_2 & \cdots & -\left(h_1h_N\right)\mathbf{e}_N \\
-\left(h_2h_1\right)\mathbf{e}_1 & \left(h_2 - h_2^{\ 2}\right)\mathbf{e}_2 & & -\left(h_2h_N\right)\mathbf{e}_N \\
\vdots & & \ddots & \vdots \\
-\left(h_Nh_1\right)\mathbf{e}_1 & -\left(h_Nh_2\right)\mathbf{e}_2 & \cdots & \left(h_N - h_N^{\ 2}\right)\mathbf{e}_N \\
\left(h_1 - h_1^{\ 2}\right) - \left(g_1 - g_1^{\ 2}\right) & h_1h_2 + g_1g_2 & \cdots & h_1h_N + g_1g_N \\
h_2h_1 + g_2g_1 & \left(h_2 - h_2^{\ 2}\right) - \left(g_2 - g_2^{\ 2}\right) & & h_2h_N + g_2g_N \\
\vdots & & \ddots & \vdots \\
h_Nh_1 + g_Ng_1 & h_Nh_2 + g_Ng_2 & \cdots & \left(h_N - h_N^{\ 2}\right) - \left(g_N - g_N^{\ 2}\right)
\end{array}\right]. \qquad (\ 5\text{ - }52\ )$$

However, since we need the Hessian matrix for a set of training data, we have to batch them; that is

$$\frac{\partial^2 J}{\partial \mathbf{x}^2} = \sum_{q=1}^{Q} \left(\frac{\partial \mathbf{m}_q}{\partial \mathbf{x}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{m}_q^{\ 2}}\right) \frac{\partial \mathbf{m}_q}{\partial \mathbf{x}}. \qquad (\ 5\text{ - }53\ )$$

One way of doing this is to use a for loop and summing each of the Hessian matrix produced by each data point q. To do this, we first calculate the experts and gating Jacobian, on equation (4-43) and (4-64), at q^{th} data point and substitute them into total modular network Jacobian on equation (5-25). Then, calculate the $\dfrac{\partial^2 J}{\partial \mathbf{m}_q^{\ 2}}$ term at q^{th} data point and multiply them together using equation (5-53) and repeat the process until the last data point. This method will be suitable for implementation in C++ or FORTRAN.

Another way of doing this is to utilize the sparse matrices in MATLAB. To do this, we can use the experts and gating Jacobian calculation on equation (5-27) and (5-28). Then, substitute them into the total Jacobian matrix on equation (5-25). The $\frac{\partial^2 J}{\partial \mathbf{m}^2}$ is calculated in such a way that it includes a window of data, stacking them diagonally. For example

$$\frac{\partial}{\partial \mathbf{y}_2}\left(\frac{\partial J}{\partial \mathbf{y}_1}\right)=\begin{bmatrix} -\left(h_{2,1}h_{1,1}\right)\mathbf{e}_{2,1}\mathbf{e}_{1,1}{}^T & \cdots & 0 & 0 \\ \vdots & -\left(h_{2,2}h_{1,2}\right)\mathbf{e}_{2,2}\mathbf{e}_{1,2}{}^T & & 0 \\ 0 & & \ddots & \vdots \\ 0 & 0 & \cdots & -\left(h_{2,Q}h_{1,Q}\right)\mathbf{e}_{2,Q}\mathbf{e}_{1,Q}{}^T \end{bmatrix}. \quad (5-54)$$

Once the $\frac{\partial^2 J}{\partial \mathbf{m}^2}$ matrix is made available, then find the Hessian matrix using equation (5-45). Take note that utilizing the sparse matrices in this algorithm is essential otherwise the $\frac{\partial^2 J}{\partial \mathbf{m}^2}$ needs to store all the zeros and it will be memory intensive. By utilizing the sparse matrices, the zeros is indicated by just the index and save a lot of memory.

Marquardt-Levenberg Modification to Approximated Newton's Method

Since all the off-diagonal terms are exactly the solution of the second derivative, we are using the exact Newton's methods on all the off diagonal terms. Meanwhile, since we assume the **H**(**x**) and **B**(**x**) to be small, we are using the approximated Newtons' methods on all the diagonal terms. Hence, we will be using the Marquardt-Levenberg modification to the approximated Newton's methods in the modular network:

$$\Delta\mathbf{x} = -\left[\left(\frac{\partial\mathbf{m}}{\partial\mathbf{x}}\right)^T\left(\frac{\partial^2 J}{\partial\mathbf{m}^2}\right)\frac{\partial\mathbf{m}}{\partial\mathbf{x}} - \mu\mathbf{I}\right]^{-1}\frac{\partial\mathbf{m}^T}{\partial\mathbf{x}}\frac{\partial\mathbf{j}}{\partial\mathbf{m}}. \qquad (5-55)$$

The parameter μ is multiplied by some factor β whenever a step would result in an increased in $J(\mathbf{x})$. When a step reduces $J(\mathbf{x})$, μ is divided by β. When μ is large, the algorithm becomes steepest ascent method and when μ is small, the algorithm becomes approximated Newton's method.

CHAPTER VI

MODULAR NETWORK ALGORITHMS COMPARISON

Comparison of Learning Algorithms for the Modular Network

To test the capability of Marquardt-Levenberg (**ML**) modification for the modular network, we have tested the algorithm on five simple function approximation problems using several network architectures. Two other learning algorithms, **S**teepest **A**scent (**SA**) and **R**esilient back**prop**agation (**Rprop**), were used on the same test problems to provide a baseline performance comparison. Since the **Rprop** algorithm has not been discussed, we will explain how the algorithm works in the following,

Resilient Backpropagation (Rprop) Algorithm

Rprop is the most recent gradient based learning algorithm [13]. Its convergence speed in multilayer networks is equivalent or slightly better than **Quickprop**. The basic principle of **Rprop** is to eliminate the harmful influence of the size of the gradient on the weight step. Only the sign of the derivative is considered to indicate the direction of the weight update. The size of the weight change is exclusively determined by a weight-specific, so called 'update-value' $\Delta_{i,j}$:

$$\Delta w_{i,j}(k)=\begin{cases}-\Delta_{i,j}(k)\ , \text{ if} & \dfrac{\partial J(k)}{\partial w_{i,j}}>0\\ +\Delta_{i,j}(k)\ , \text{ if} & \dfrac{\partial J(k)}{\partial w_{i,j}}<0\\ 0\quad , \text{ if} & \text{else}\end{cases} \qquad (6-1)$$

The second step of **Rprop** learning is to determine the new update-values $\Delta_{i,j}(k)$ based on the sign dependent adaptation process.

$$\Delta_{i,j}(k)=\begin{cases}\eta^{+}\times\Delta_{i,j}(k-1)\ , \text{ if} & \dfrac{\partial J(k-1)}{\partial w_{i,j}}\times\dfrac{\partial J(k)}{\partial w_{i,j}}>0\\ \eta^{-}\times\Delta_{i,j}(k-1)\ , \text{ if} & \dfrac{\partial J(k-1)}{\partial w_{i,j}}\times\dfrac{\partial J(k)}{\partial w_{i,j}}<0\\ \Delta_{i,j}(k-1)\ , & \text{else}\end{cases} \qquad (6-2)$$

where $0<\eta^{-}<1<\eta^{+}$.

All update-values are initialized to a value Δ_{o}. The choice of this value is rather uncritical for the multilayer network, a typical value is $\Delta_{o}=0.1$. To prevent the weight changes from becoming too large, a maximum upper bound $\Delta_{\max}$ is set for each $\Delta_{i,j}$; a typical value is $\Delta_{\max}=50$. The increase and decrease factors are typically fixed at $\eta^{+}=1.15$ and $\eta^{-}=0.7$. These values are based on empirical tests.

Evaluation Methods

To accurately evaluate the speed of these three algorithms, we evaluate the performances in two stages. In the first stage, ten random initial weights and biases are selected and used throughout each test problem for the three learning algorithms. Take note that the parameters in each algorithm are set to a fixed number which are used throughout the test problem in this stage. For the **SA** algorithm, the learning rate, α, is set to 0.1. For **Rprop**, the typical values that were described previously are used: $\eta^{+} = 1.15$, $\eta^{-} = 0.7$, $\Delta_{o} = 0.1$ and $\Delta_{\max} = 50$. For the **ML** algorithm, the initial μ is set to 100, and the increasing and decreasing factor β is set to 5. The average number of epochs and flops are calculated for those trials which converge.

In the second stage, we select four sets of initial weights and biases that give the best results in the first stage for each problem. Then, we try to fine tune the parameters in each algorithm for each specific initial weights and biases. It required many runs to fine tune these parameters to give the best possible results for each set of random initial weights and biases. The results are plotted in terms of the sum of squared error versus the number of floating point operations as shown in Figure 6 - 1 to Figure 6 - 4. Also, the average number of epochs, average number of floating point operations and relative speed are summarized for each problem. In both stages, the relative speeds are obtained by dividing the average number of flops for each algorithm by the smallest average number of flops.

This second stage evaluation will provide us with an accurate evaluation of the speed comparison on these algorithms. This is because the first stage comparison might contain a large fluctuation, since some initial weights and biases might take longer to converge. Also, the second stage will provide us some measure of how diversified the first stage is. Since the **SA** algorithm is very slow in converging, we excluded the **SA** algorithm for the second stage evaluation.

Training Set

The training set used throughout the first four problems is an absolute value function. This absolute value function is chosen because it contains a sharp change at the point (0,0) which modular network is very good at capturing. This training set consists of 21 input/output pairs which cover the interval [-1,1]. Throughout these first four problems, two expert networks and one gating network will be used. We will compare the speed of the three learning algorithms while varying numbers of layers in the expert and/or the gating network.

On the fifth problem, we will use a sawtooth function as the training set. This sawtooth function consists of 21 input/output pair which cover the interval [-1,1] and it is shown in the top left plot of Figure 6 - 5 with a + mark.

Problem #1

In this test problem, each expert network has a 1-1 architecture with linear output function, and the gating network has a 1-2 architecture. The stopping criterion is set at sum of squared error of 10^{-4}.

Table 6 - 1 summaries the first stage evaluation result of the three methods for an average of ten trials. When we compare the average number of epochs required to converge, the **ML** shows about 5 times fewer epochs than the **Rprop** and about 1100 times fewer epochs than **SA**. However, the average number of epochs provides limited information, since the three algorithms do not have the same number of floating point operations (flops) per each iteration. As shown in the last column, the **ML** algorithm is about 1.665 times faster than the **Rprop** and about 385 times faster than the **SA** method for the 10^{-4} stopping criteria.

	Epochs	***Floating Point Operation***	***# of Successes***	***Relative Speed***
ML	16.6	266685.6	10/10	1
Rprop	77	444229.8	10/10	1.665
SA	19567.2	102842201.5	10/10	385.6

Table 6 - 1. First Stage Algorithms Comparison for Problem #1

In the second stage, we fine tuned the four best sets of initial weights and biases that we obtained in the first stage. The results are summarized in Figure 6 - 1 and Table 6 - 2. In Figure 6 - 1, the sum of squared errors versus the number of floating point operations for each algorithm are plotted (solid line – **ML** , dashed line – **Rprop**). As shown, **Rprop** is initially faster than **ML** but **ML** takes over **Rprop** later. This is not surprising because **ML** takes about 3 times more flops than the **Rprop** for each epoch. As shown in Table 6 - 2, the average number of epochs and average number of floating point operations for both algorithms show an improved performance after fine tuning the parameters. The relative speed comparison in term of flops shows that **ML** is still about 1.66 times faster than the **Rprop**. This indicates that the results obtained in the first stage are quite accurate.

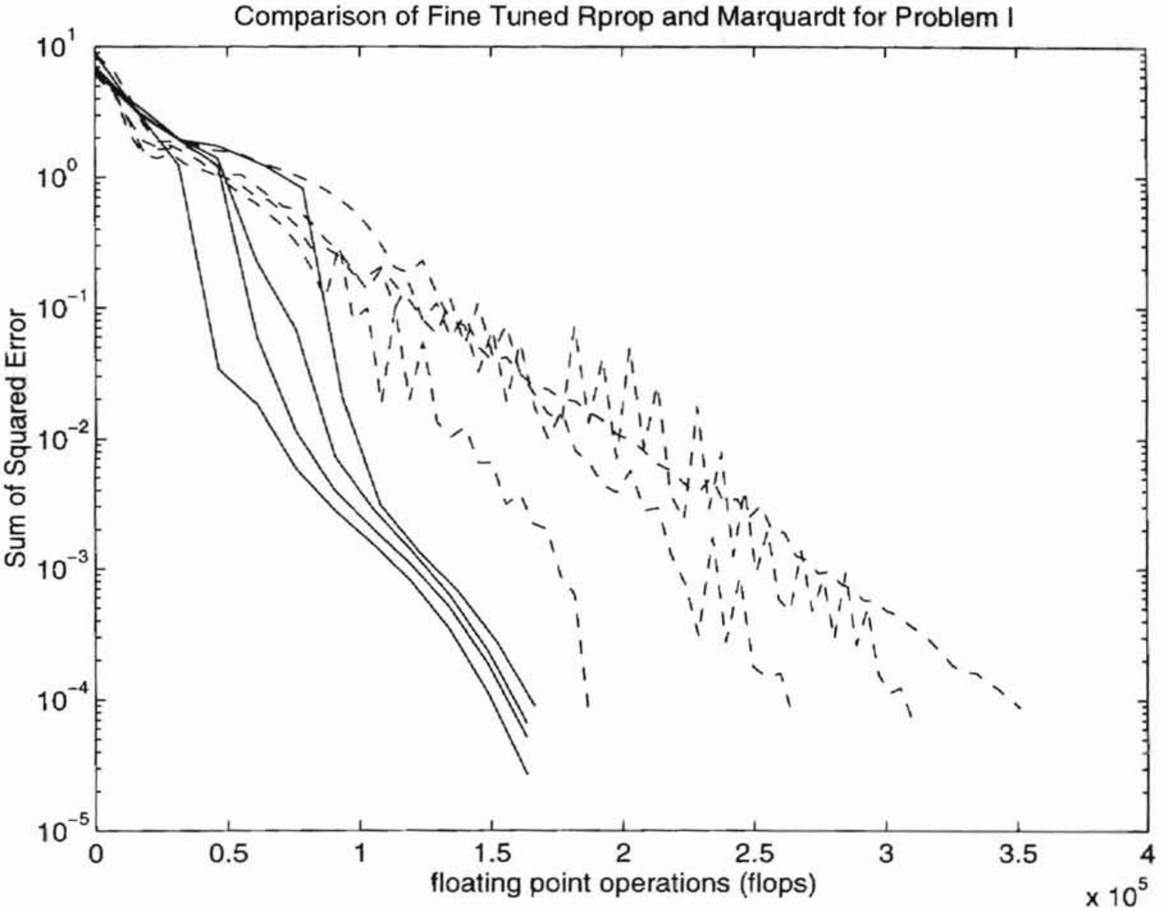


Figure 6 - 1. Comparison of ML and Rprop algorithms

	Epochs	*Floating Point Operation*	*Relative Speed*
ML	11	170623.25	1
Rprop	54.75	282621.5	1.66

Table 6 - 2. Second Stage Algorithms Comparison for Problem #1

Problem #2, Increasing the Size of the Gating Network

In this test problem, we took the modular network architecture of problem #1 and added a tan-sigmoidal hidden layer to the gating network. Hence, the gating network has a 1-2-2 architecture. The stopping criterion is set at a sum of squared error of 10^{-4}. Table 6 - 3 summarizes the first stage results. As shown in Table 6 - 3, the **ML** is still approximately twice as fast as **Rprop** and about 124 times as fast as **SA** when comparing the relative speed in term of flops. When comparing the average number of epochs taken to converge with problem #1, it seems that by adding a hidden layer, the algorithms take fewer epochs to converge, except for the **Rprop**.

	Epochs	***Floating Point Operation***	***# of Successes***	***Relative Speed***
ML	12	381020.14	10/10	1
Rprop	87.11	786340.33	10/10	2.063
SA	5660.2	47277739.2	8/10	124.08

Table 6 - 3. First Stage Algorithms comparison for Problem #2

However, after going through the second stage to fine tune the parameters, Table 6 - 4 reveals that **Rprop** takes an average of 41 epochs to converge for the four initial weights selected from stage one. That is less than the second stage average number of epochs in problem #1. This result indicates that the **Rprop** does take fewer epochs to converge in problem #2 than problem #1. It also implies that adding a hidden layer to the gating network improves the convergence.

When comparing the average number of flops it takes to converge in stage one and two, the **Rprop** has shown an improvement from 786340.33 to 354798.25. This could indicate the first stage **Rprop** results are not very accurate. This inaccuracy is also reflected in the relative speed, where the **ML** is shown to be about 1.48 times faster than the **Rprop** in the second stage, rather than 2.06 times in the first stage.

While Table 6 - 4 summarizes the average results, Figure 6 - 2 shows the sum of squared error versus the flops for four sets of initial weights and biases selected from the first stage. Again, a similar phenomenon occurs, that is **Rprop** leads in the SSE in the beginning and **ML** takes over later.

	Epochs	***Floating Point Operation***	***Relative Speed***
ML	7.75	239470	1
Rprop	41	354798.25	1.48

Table 6 - 4. Second Stage Algorithms Comparison for Problem #2

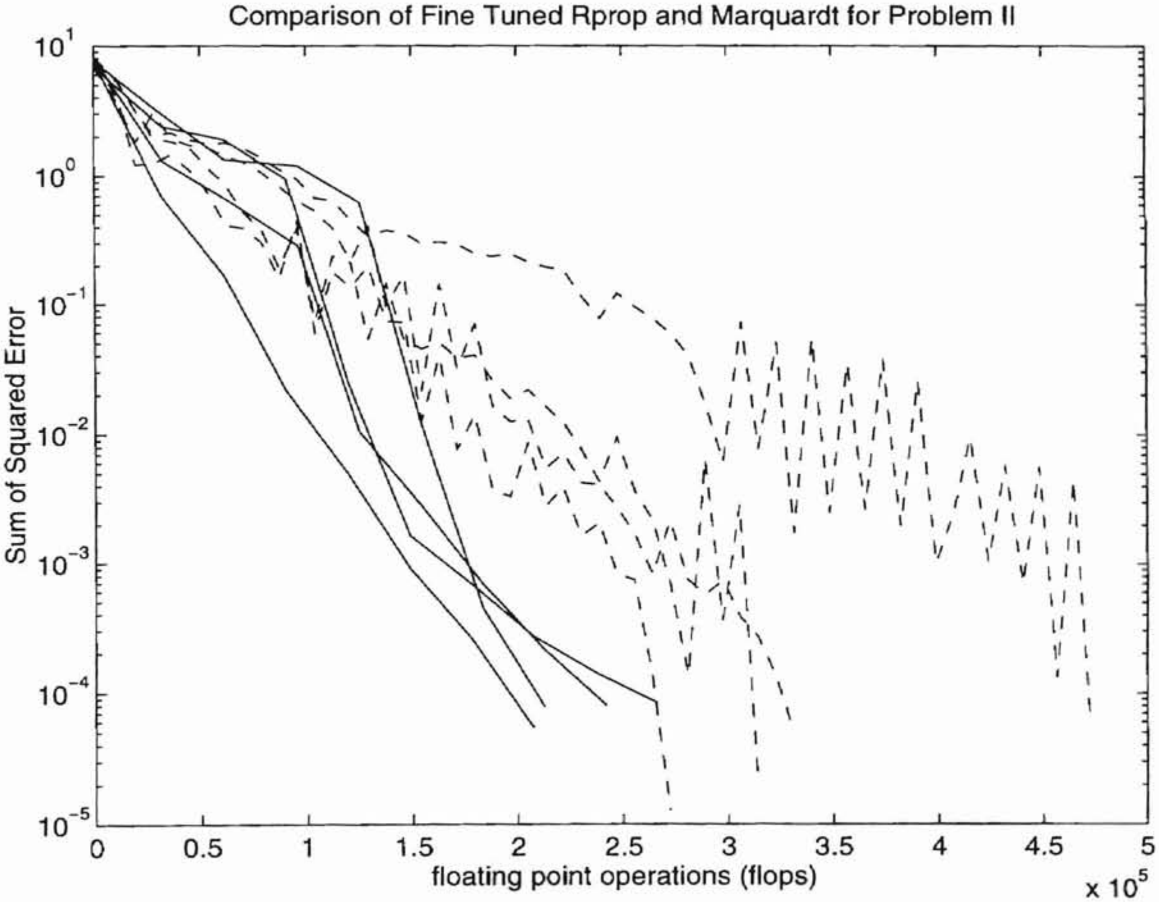


Figure 6 - 2. Comparison of ML and Rprop Algorithms for Problem II

Problem #3, Increasing the Size of the Expert Network

In this test, we added a tan-sigmoidal hidden layer, 1-2-1, to each expert network, while retaining the 1-2 architecture of the gating network. The stopping criterion is set at a sum of squared error of 10^{-3}. Due to the long training time of the **SA** method, it is not used in this test problem. As shown in Table 6 - 5, the **Rprop** algorithm takes about 2.3 times as many flops as **ML** to converge in the first stage testing. Also, **ML** converges 9 times and **Rprop** converges 7 times. This seems to suggest that **ML** converge more often than the **Rprop**. For a much accurate comparison, the second stage results are shown on Table 6 - 6. As shown, the **Rprop** takes about 1.58 times as many flops to converge than **ML** which is less than the first stage. Again, this phenomenon has suggested that the first stage result has many fluctuations.

A plot of the sum of squared error versus the number of floating point operations on the four selected sets of initial weights and biases, Figure 6 - 4, seems to suggest that the error surface on this network architecture is rough. This is especially true as **Rprop** shows many spikes during the course of convergence (implies that the weight space is very sensitive, a small change will result in a big error) and **ML** shows a very flat surface during the course of convergence (implies that **ML** takes very small steps in this region). Also, the same phenomenon shows up again where **Rprop** leads in the beginning of the training while **ML** takes over later.

	Epochs	***Floating Point Operation***	***# of Successes***	***Relative Speed***
ML	58	2521575.142	9/10	1
Rprop	521.5	5914616.4	7/10	2.346

Table 6 - 5. First Stage Algorithms Comparison for Problem #3

	Epochs	*Floating Point Operation*	*Relative Speed*
ML	27	1079905	1
Rprop	196.75	1660871	1.54

Table 6 - 6. Second Stage Algorithms Comparison for Problem #3

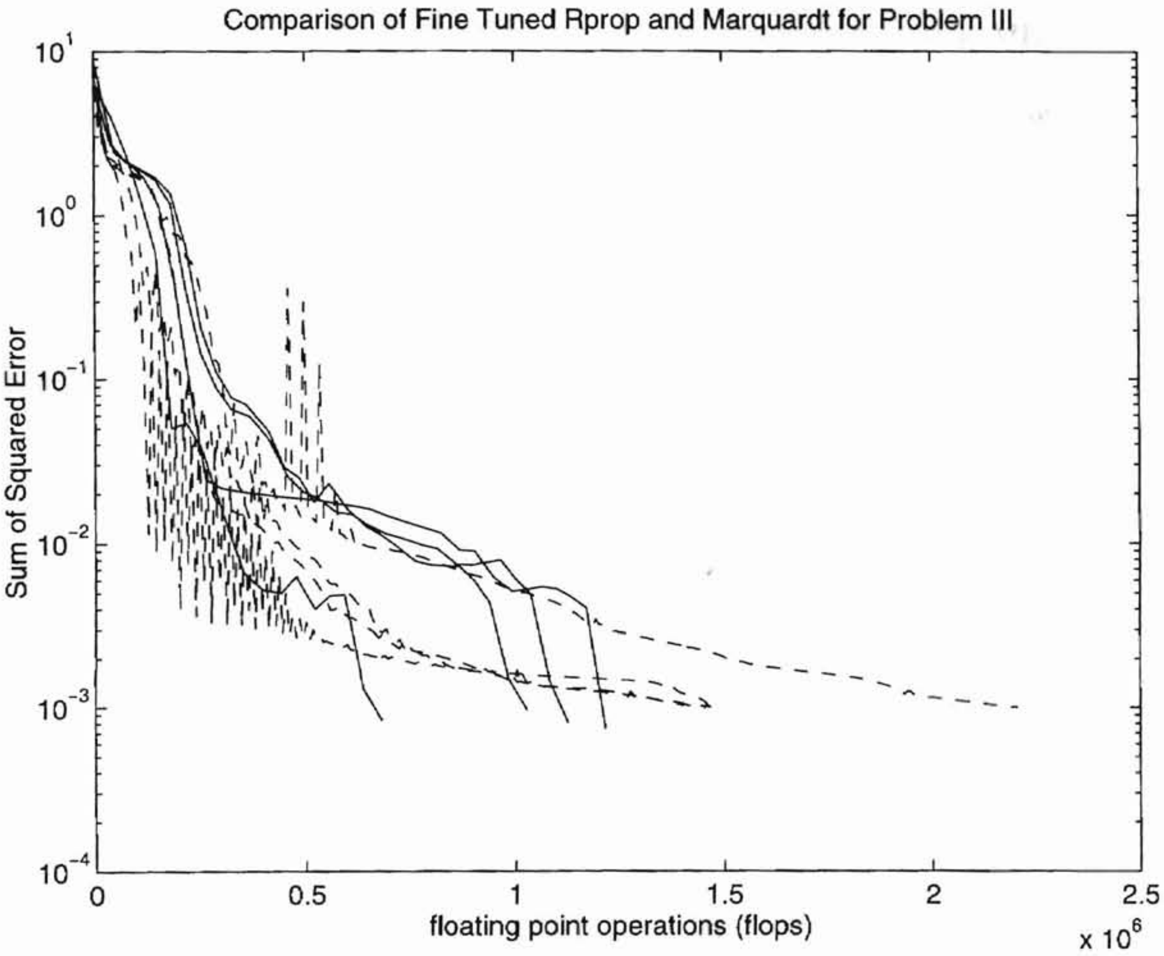


Figure 6 - 3 Comparison of ML and Rprop Algorithms for Problem III

Problem #4, Increasing the Size of both Expert and Gating Networks

The fourth test problem adds a tan-sigmoidal hidden layer to both the expert and the gating networks. Each expert network has a 1-2-1 architecture, and the gating network has a 1-2-2 architecture. The stopping criterion is set at a sum of squared error of 10^{-3}. As shown in Table 6 - 7, the **Rprop** algorithm requires 5 times more flops than **ML** to converge in the first stage testing. Meanwhile, as shown in Table 6 - 8, the **Rprop** algorithm requires about 2.2 times more flops in the second stage testing. This large variation seems to be caused by the increasing complexity of the network architecture. Nevertheless, this indicates that as network complexity increases, the network training is very sensitive to the parameters of the algorithms. Table 6 - 8 also shows that as the network complexity increases, **ML** can converge much faster than **Rprop**. A plot of the second stage results, Figure 6 - 4, have indicated that **Rprop** seems to encounter shallow error surfaces and has a hard time to converge. This shows that the convergence speed difference will become more pronounced when a lower stopping criterion is set.

	Epochs	*Floating Point Operation*	*# of Successes*	*Relative Speed*
ML	25.4	1510953	9/10	1
Rprop	890.9	7467667.4	6/10	4.94

Table 6 - 7. First Stage Algorithms Comparison for Problem #4

	Epochs	*Floating Point Operation*	*Relative Speed*
ML	9.75	642584	1
Rprop	120.5	1425887.25	2.22

Table 6 - 8. Second Stage Algorithms Comparison for Problem #4

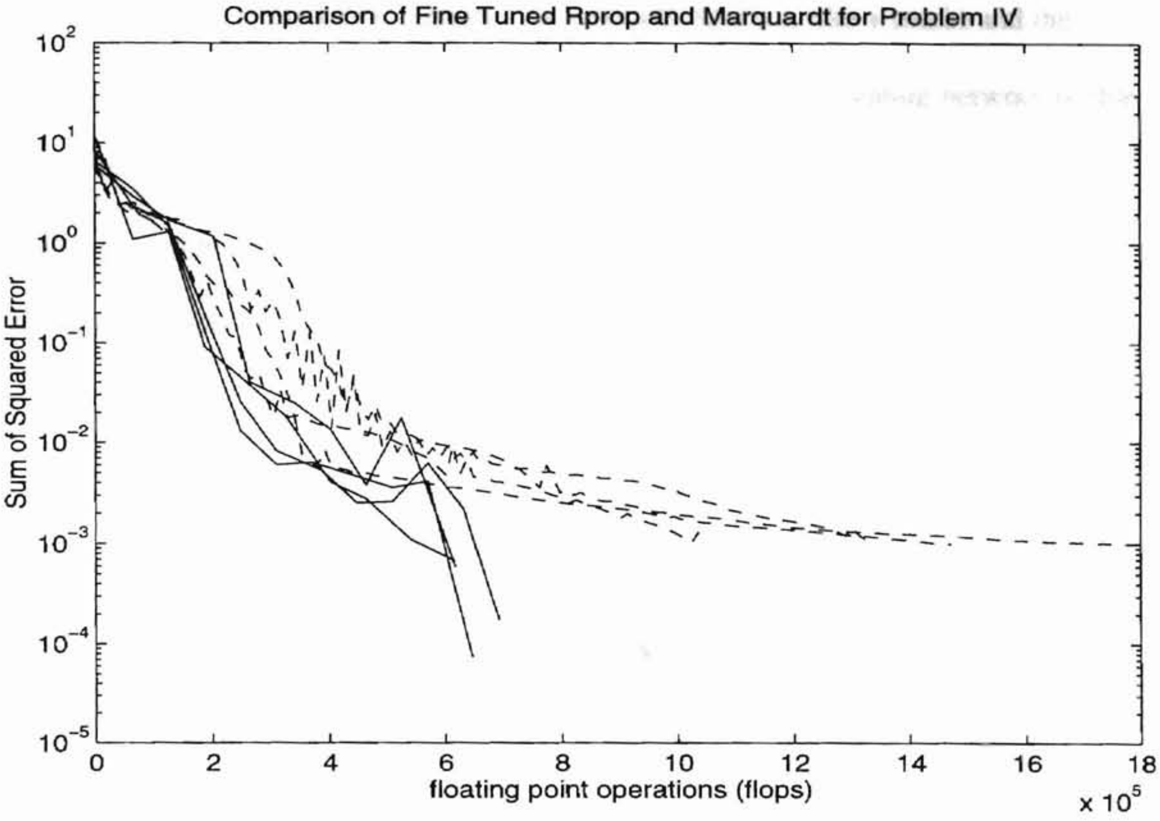


Figure 6 - 4. Comparison of ML and Rprop Algorithms for Problem IV

Problem #5

In this test we will approximate the sawtooth function which is shown in the top left corner of Figure 6 - 5 (with the + mark). We use a modular network that contains 4 expert networks and a gating network. Each expert network has a one layer 1-1 architecture with a linear transfer function, and the gating network has a 1-4 architecture. All three algorithms are used in approximating this sawtooth function, but as shown in Table 6 - 9, only **ML** successfully trains in the first stage testing with a low successful rate, two out of ten. We tried to tune the parameters for **Rprop** so that it would approximate the sawtooth function but of no avail. Figure 6 - 5 shows a case where a modular network successfully approximated the sawtooth function with the **ML** algorithm. In the top left plot, it shows the sawtooth function (the + mark) and the modular network output response (the solid line). As shown, the gating network is able to divide the region into four sections (top right plot) and assign each expert network to learn one region (lower left plot). The lower right plot is the learning curve of the network. The dash-dot line is the sum of squared error and solid line is the negative performance index.

This sawtooth approximation has provided us some insights into how the modular network works. However, the two out of ten success rates has suggested the difficulty of this test problem. Nevertheless, it indicates that **ML** can converge on some difficult problems where **Rprop** and **SA** cannot converge.

	Epochs	***Floating Point Operation***	***# of Successes***
ML	95.5	18268004	2/10
Rprop	-	-	0/10
SA	-	-	0/10

Table 6 - 9. Algorithms Comparison for Problem V

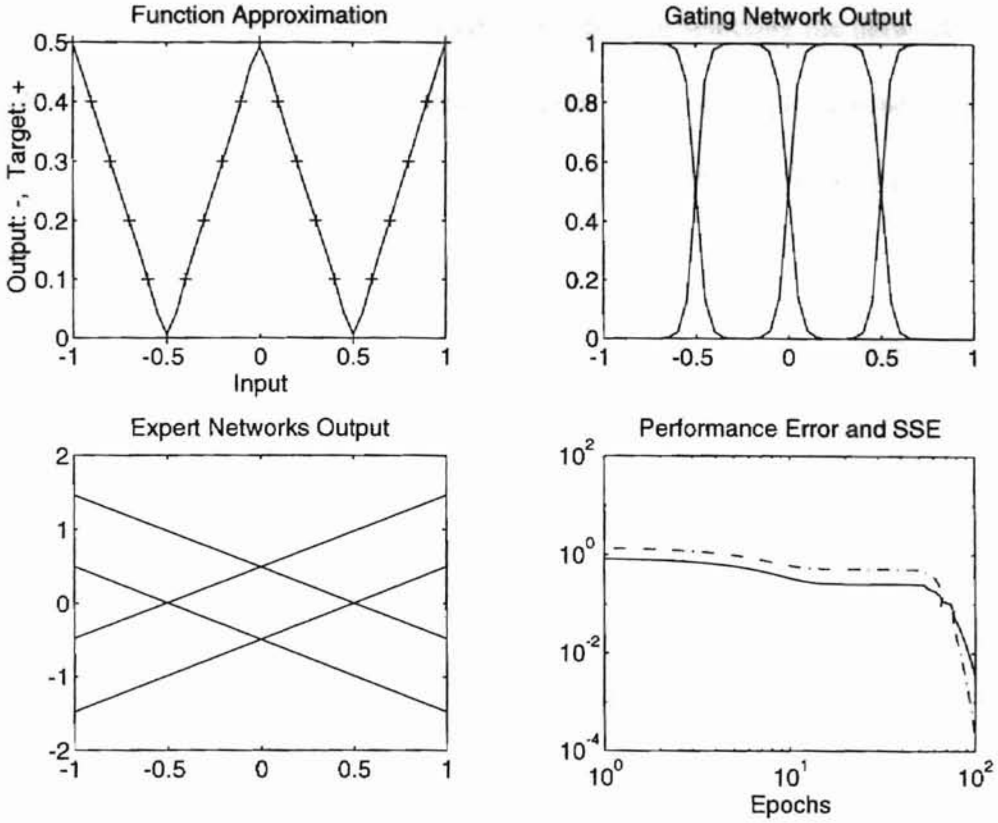


Figure 6 - 5. Approximating the Sawtooth Function

Summary

In this chapter, several tests were performed to test the Marquardt-Levenberg algorithm for the modular network. As shown in the five tests above, the **ML** algorithm converges much faster than the **Rprop** and **SA** algorithms. These effects become more pronounced when a lower stopping criterion is set. Also, as the complexity of the network increases, we can see that **Rprop** converges less often than **ML**. The author also finds that the parameters for each algorithm are very sensitive; a slight variation in the parameters may result in longer training. Throughout the tests, adding a hidden layer in the gating network seems to speed up the converging rate. The success rates in problem #5 have suggested that more research needs to be done in selecting the network architecture. We will leave this to the next Chapter, where we will examine some network architectures and a gating weight initialization method to speed up the training of the modular network.

CHAPTER VII

MODULAR NETWORK PERFORMANCE

In this Chapter, several preliminary simulation results are discussed. In test 1, we will see a comparison between a modular network and a two-layered feedforward network with varying hidden layer. This test demonstrates the superiority of the modular network over the multilayer network for certain problems. In test 2, a two-cycle sinusoid is used to test the modular network. This test demonstrates the strengths of modular networks. In test 3, a four-cycled sinusoidal test pattern is used to test a modular network that has 5 expert networks. This simulation shows a case where the modular network fails to converge. Test 4 and test 5 offer some ways to improve the performances of the modular network. One way of improving the modular network is to use the what and where modular network, and this is demonstrated in test 4. Another way of improving the modular network is shown in test 5. This improvement is to develop a weights initialization method to preset the gating network's weights. In test 6 and test 7, we will demonstrate the capability of modular network in approximating the coulumb friction model and the classical friction model. Lastly, in test 8, we will use the modular network to model a single linked pendulum that contains a coulomb friction. This test demonstrates an important application of the modular network in system identification.

Test #1 Comparison of Multilayer Networks and Modular Network

Test Function: Absolute Value Function

In test 1, we compare the capability of the modular network versus the multilayer network in implementing a function with discontinuous derivative. The test pattern is an absolute value function over the interval [-1,1]. The modular network has two expert networks. Each expert network has a 1-2-1 architecture with a hyperbolic tangent function in the first layer and a linear function in the second layer. The gating network has a 1-2 architecture. Meanwhile, the multilayer network has a 1-N-1 architecture with hyperbolic tangent functions in the first layer and a linear function in the second layer. N is the number of hidden neurons and is selected to be N=2, 10, 20. Hence, we will compare three two-layer networks with one modular network. The intention of this test is to see how many hidden neurons, N, are needed to train the absolute value function. Both networks are trained using the Marquardt-Levenberg algorithm. The algorithm stops if it reaches 1000 epochs or if the magnitude of the gradient is less than 10^{-5}.

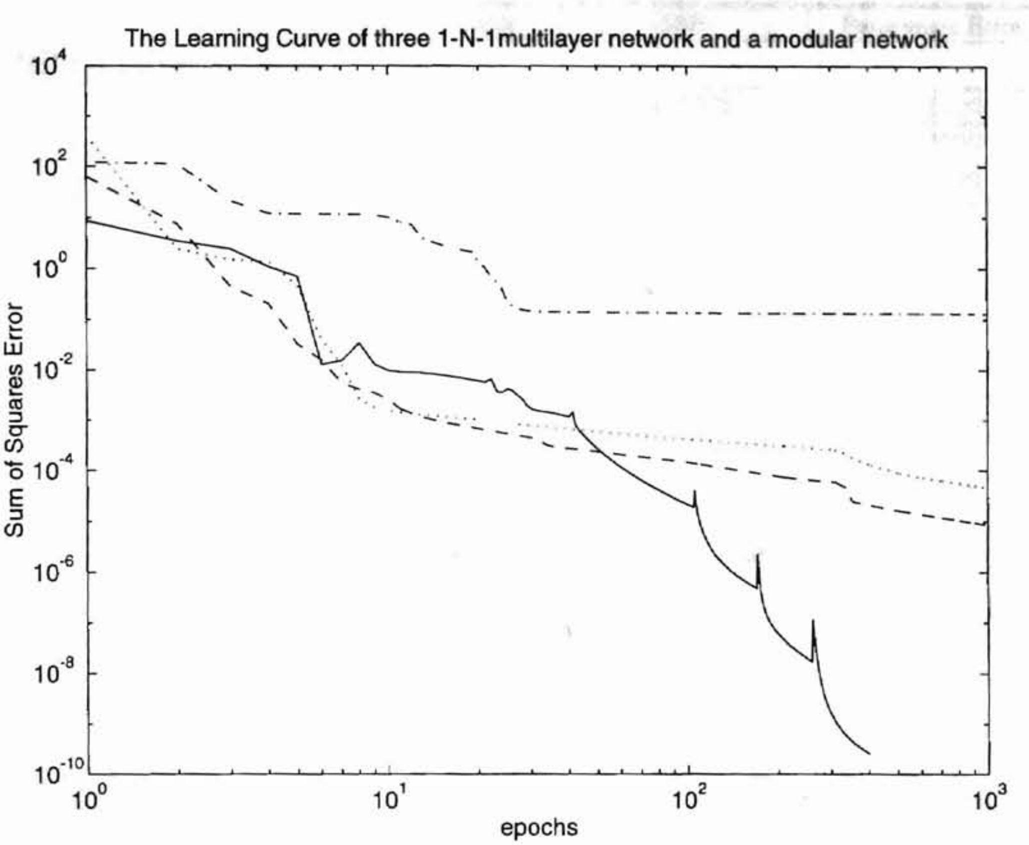


Figure 7 - 1. Learning Curve of 3 Two-layered Network and a Modular Network

Figure 7 - 1 shows the learning curves of these 4 networks. The 1-2-1 network is indicated by the dash-dot line and it has the highest sum of squared error. As shown, the network stops learning at 22 epochs. Meanwhile, the 1-10-1 and 1-20-1 networks, indicated by dotted line and dashed line, have a lower sum of squared error than the 1-2-1 network. However, when we compare them to the modular network, the modular network has a lower sum of squared error than any of the two-layer networks.

	# of Parameters	SSE	Percentage Error
Multilayer Networks			
1-2-1 network	7	0.131558	36.8212%
1-10-1 network	31	6.345857×10^{-5}	0.7966%
1-20-1 network	61	8.761381×10^{-6}	0.2960%
Modular Network			
each expert 1-2-1, gating 1-2	18	5.364634×10^{-10}	0.0023%

Table 7 - 1. Comparison of Multilayer and Modular Networks

Table 7 - 1 compares the number of parameters, the minimum sum of squared error and the percentage error in each network. The modular network uses only 18 parameters and has a lower percentage error than all three two-layer networks. Also, it takes less training time than any of the multilayer networks.

Considering the above test, the modular network seems to be a very good candidate for discontinuous functions.

Test #2 Sine Wave Testing I

Test Function: Two-Cycled Sine Wave

The modular network used in this two-cycle sine wave test (+ mark in Figure 7 - 2) is a modular network with two experts and a gating network. Each expert network has a 1-2-1 architecture, with hyperbolic tangent functions in the first layer and a linear function in the output layer. The gating network used in this test is a single layer with 1-2 architecture. As shown in Figure 7 - 2, the gating network is able to assign each expert to one cycle of the sine wave and approximate the two-cycle sine wave function to a sum squared error of less than 10^{-5}. Interestingly, if a 1-2-1 multilayer network is used to approximate this two-cycle sine wave function, the minimum sum of squared error it can reach is 10^{-3}.

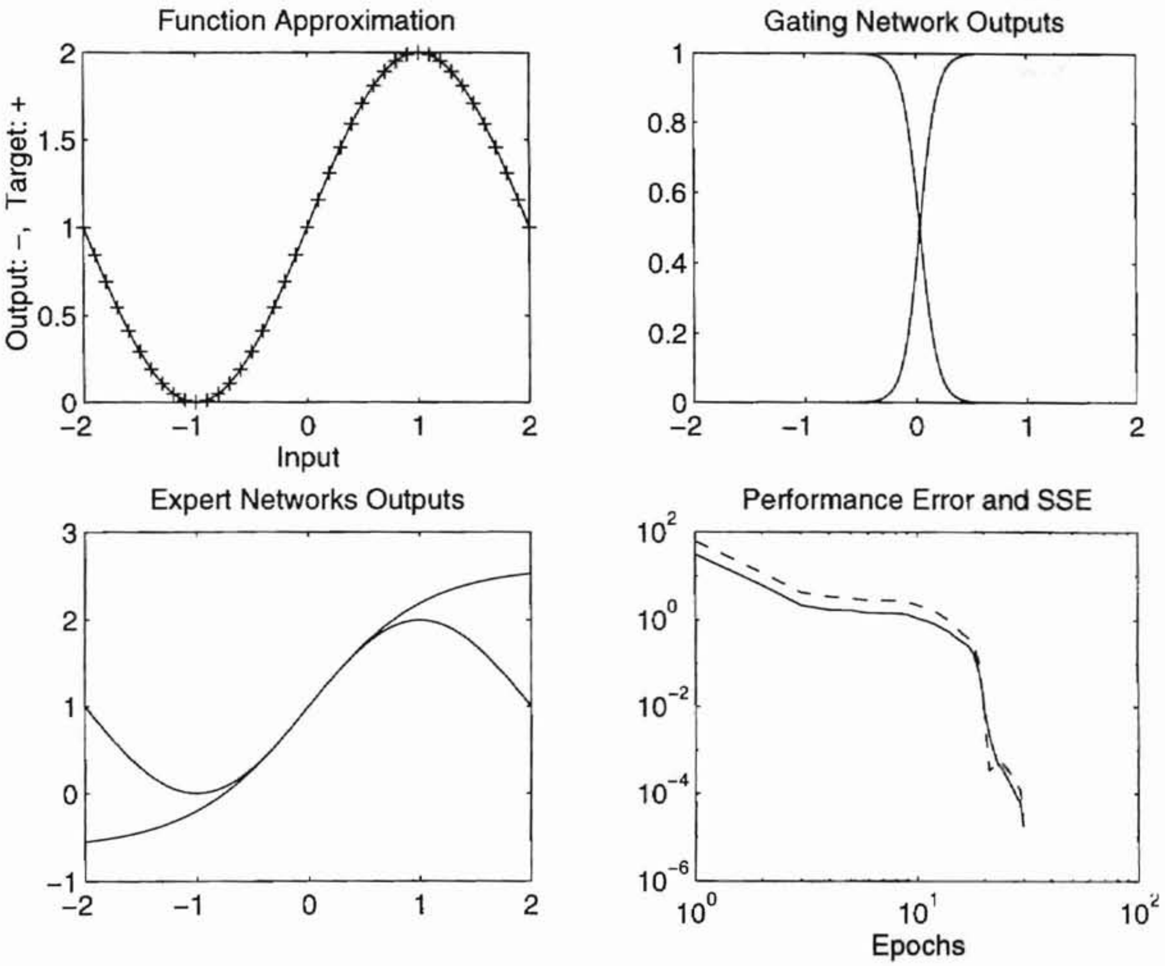


Figure 7 - 2. Two-cycle Sine Wave Testing

Test #3 Failure in Training

Test Function: Four-Cycled Sine Wave

In this test, a modular network with 5 expert networks is used to approximate a four-cycle sinusoidal function. Each expert network has a 1-2-1 architecture with tan-sigmoidal and linear transfer functions. It is shown in the plots in Figure 7 - 3 that the gating network uses only two expert networks and turns off the other three expert networks. The sum of squared error dashed line as shown in the lower right of Figure 7 - 3, reaches 10^{-3}. As we have noted from test #2, a 1-2-1 multilayer network can only train to a sum of squared error of 10^{-3}. Hence, it is not surprising that this network reaches the same minimum point. This shows that as the complexity of the network architecture increases, it becomes harder and harder to train the modular network because of many local minimum and local maximum points which exist during the course of training. To avoid these cases, we will look into several ways to improve the training particularly: the what and where modular network architectures and the gating weights initialization method.

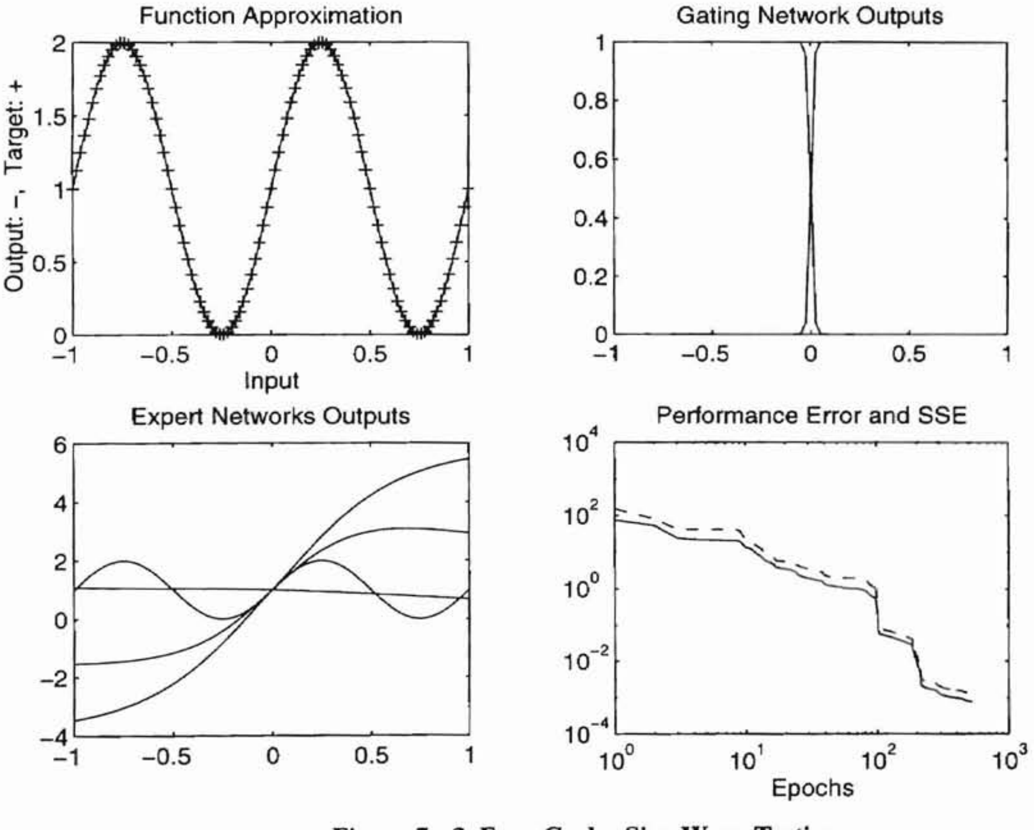


Figure 7 - 3. Four Cycles Sine Wave Testing

Test #4 The What and Where Modular Network

Test Function: Absolute Value

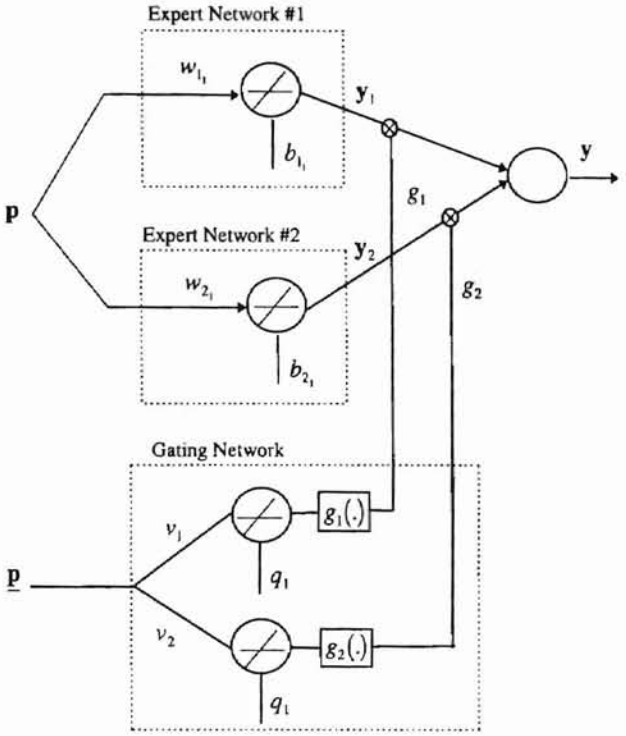


Figure 7 - 4. The What and Where Modular Network Architecture

To compare the performance of the modular network with the what and where modular network, we use the architecture shown in Figure 7 - 4 to approximate the absolute value function over the interval [-1,1]. In the first architecture, MA#1, we feed the same input pattern into both the expert network and the gating network, $\underline{\mathbf{p}} = \mathbf{p}$. Hence, this network has no pre-information of how the data should be divided. In the second architecture, MA#2, we feed the gating network with the sign of the test pattern; that is if $\mathbf{p} \geq 0$ then $\underline{\mathbf{p}} = 1$ and if $\mathbf{p} < 0$ then $\underline{\mathbf{p}} = -1$. In the third architecture, MA#3, the gating network is fed with binary information, using 2 inputs. The gating network receives $\underline{\mathbf{p}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ if the input $\mathbf{p} \geq 0$ and receives $\underline{\mathbf{p}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ if the input $\mathbf{p} < 0$. Each modular network is trained using the Marquardt-Levenberg method until the sum of squared errors reach 10^{-4}. In all ten trials, the weights and biases in the modular network are randomly initialized. The initial μ is set to 100 and the increasing and decreasing parameter, β, is set to 5.

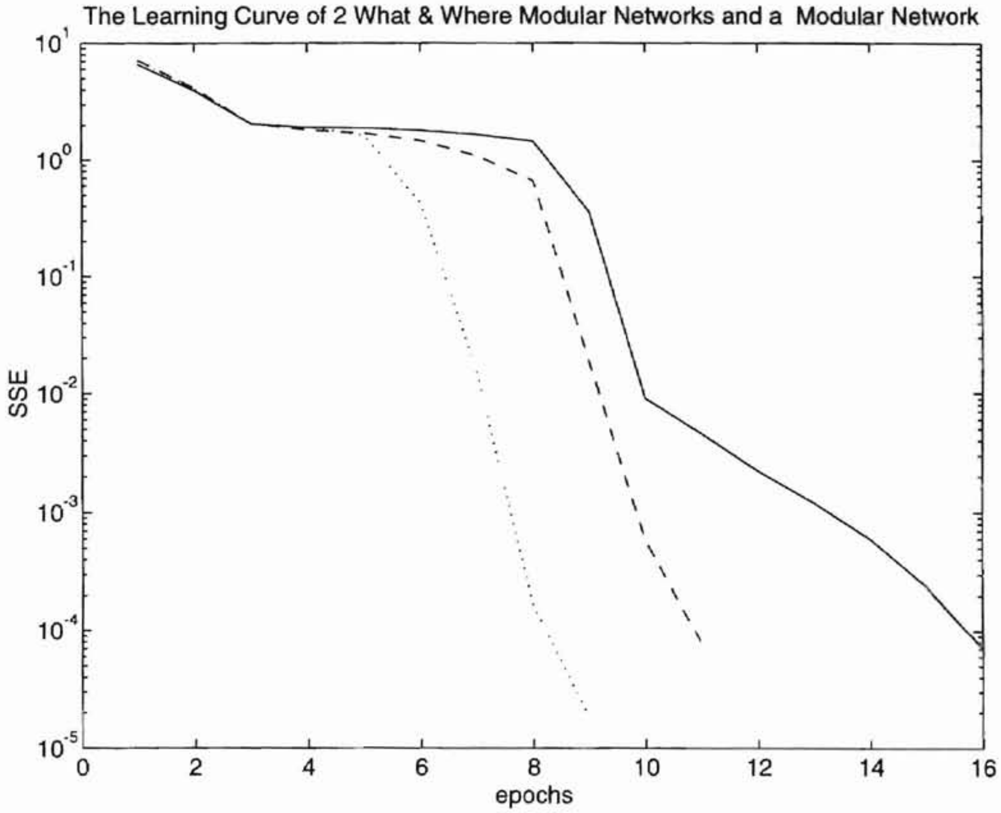


Figure 7 - 5. Learning Curve Comparison of 3 modular networks

Figure 7 - 5 shows the learning curves of the three different modular networks described above. The MA#1, MA#2 and MA#3 are represented by the solid line, dashed line and the dotted line, respectively.

Network Architecture	**Number of Success in 10 trails**	**Average # of Epochs**
MA # 1	10/10	16.375 epochs
MA #2	10/10	11.5 epochs
MA #3	10/10	7.5 epochs

Table 7 - 2. Comparison Table of 3 Different Modular Architectures

Advantages of Using What and Where Modular Network

Table 7 - 2 summarizes the number of successes and the average # of epochs in 10 attempts on the 3 architectures described above. As shown in Figure 7 - 5 and Table 7 - 2, MA#1 takes longer to train than MA#2 and MA#3. This comparison tells us that we can achieve faster convergence in training by providing the network with more precise information on how the data should be divided. From Table 7 - 2, we can see that all three network trained successes. Meanwhile, when we compare MA#2 and MA#3, the MA#3 trains slightly faster than the MA#2. This implies that the stronger the condition given to separate the classes, the faster it trains.

Disadvantage of Using What and Where Modular Network

There are two disadvantages of the what and where modular network. First, when there is no classification information given, then we cannot use the what and where modular network, because we do not know which information belongs to which class. Another disadvantage is that we cannot compute the derivative of the outputs of the network with respect to the network inputs. This is especially important when we use a modular network to model a plant and we want to backpropagate from the modular network plant to the controller. For example, the what and where modular network cannot be used for modeling the plant in Model Reference Adaptive Control.

Test #5: Weight Initialization Method

Test Function: Absolute Value Function

In this test we will develop a weight initialization method for a single layer gating network with 2 expert networks. In Chapter 3, we discussed how the weights and biases affected the decision region. Here, we will discuss this effect in more detail.

The basic idea behind the weight initialization in the modular network is to set the weights and biases in the gating network based on the input data, so that each expert network gets to learn a region. Take a case where we have 2 expert networks, then it is desired to have 2 regions classified based on the input to the gating network. If we have an input *p* into the gating network, then the output of the gating network will be

$$g_1 = \frac{\exp(v_1 p + q_1)}{\exp(v_1 p + q_1) + \exp(v_2 p + q_2)} \qquad (7 - 1)$$

and

$$g_2 = \frac{\exp(v_2 p + q_2)}{\exp(v_1 p + q_1) + \exp(v_2 p + q_2)}. \qquad (7 - 2)$$

g_1 and g_2 can be rewritten as

$$g_1 = \frac{1}{1 + \dfrac{\exp(v_2 p + q_2)}{\exp(v_1 p + q_1)}} \qquad (7 - 3)$$

and

$$g_2 = \frac{1}{1 + \dfrac{\exp(v_1 p + q_1)}{\exp(v_2 p + q_2)}} \tag{7 - 4}$$

Then, the decision boundary occurs at $\exp(v_2 p + q_2) = \exp(v_1 p + q_1)$ or

$$p = \frac{q_2 - q_1}{v_2 - v_1}. \tag{7 - 5}$$

This is consistent with our discussion in Chapter 3 where we said that the decision boundary depends on the distance between the weights and biases. Let

$$p_{mid} = \frac{p_{\max} - p_{\min}}{2}. \tag{7 - 6}$$

Then, our desired decision boundary will be at

$$\frac{q_2 - q_1}{v_2 - v_1} = p_{mid}. \tag{7 - 7}$$

If we randomly select v_1 , v_2 and q_1, we can get q_2 by

$$q_2 = p_{mid}(v_2 - v_1) + q_1. \tag{7 - 8}$$

We randomly selected v_1 , v_2 and q_1 in the interval [-10,+10] and retrained the MA#1 architecture in test #3 with and without this new weight initialization procedure. Figure 7 - 6 shows the learning curves of the modular network trained with and without weight initialization using the Marquardt-Levenberg algorithm. With weight initialization, the network trains a lot faster than without weight initialization. In the average over ten trials, the network takes 11.2 epochs to converge to SSE=10^{-4}. In fact, the speed is comparable with the MA#2 in test #4.

The weight initialization for the case of more than 2 experts and more than 1 layer in the gating network is still under development.

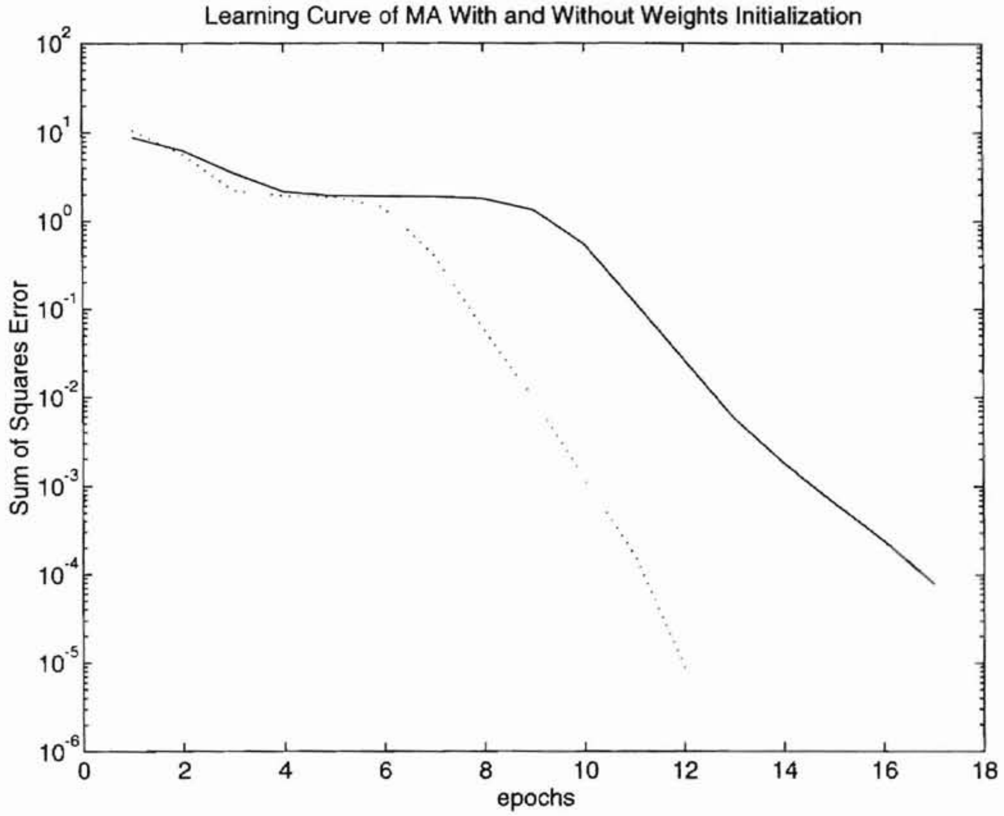


Figure 7 - 6. Learning Curve of a Modular Network with and without weights initialization on gating network

Test #6 Friction Model Approximation I

Test Function: Coulomb Friction Model

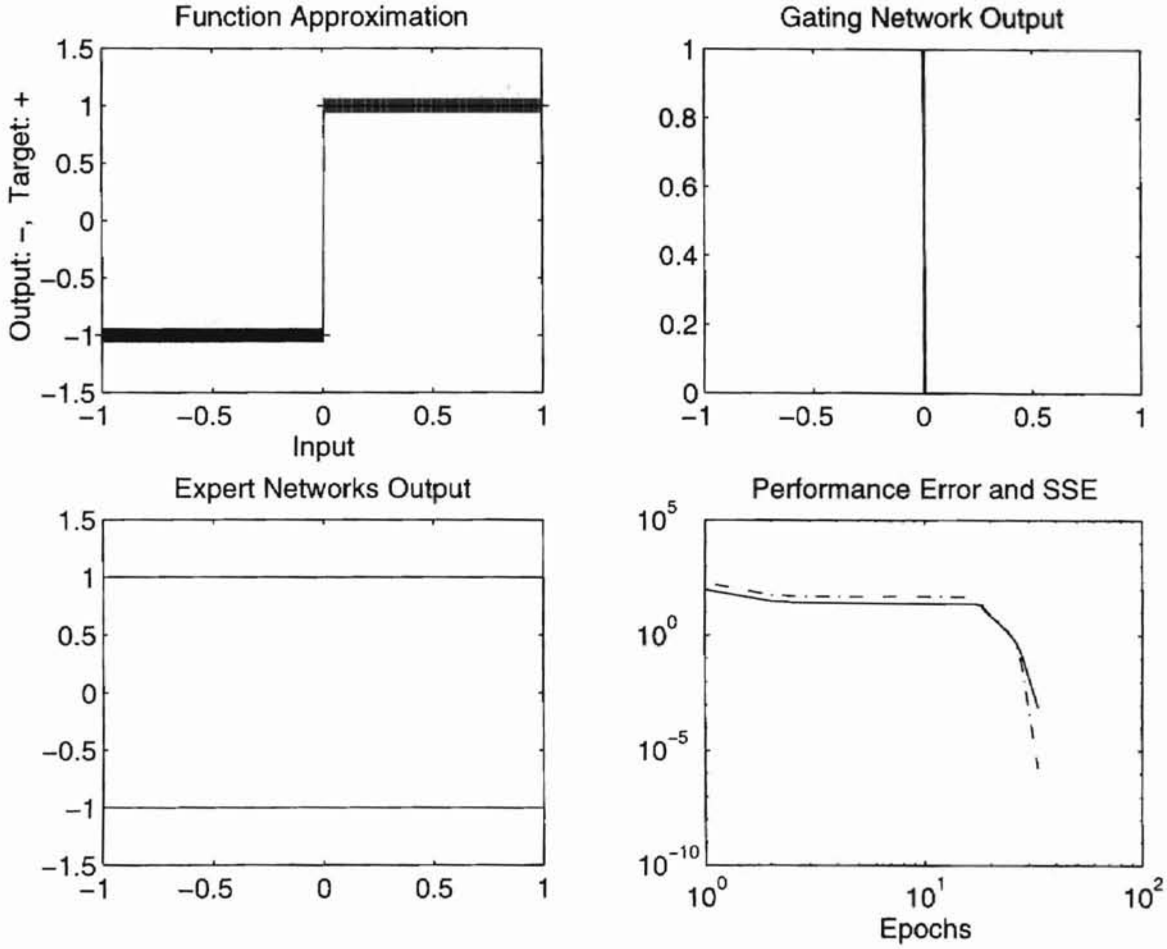


Figure 7 - 7. Modeling Coulumb Friction Model

To use the modular network to approximate a discontinuous function, in particular the friction function, our first attempt is to model a simple coulumb friction. We can think of a coulumb friction as a sign function. We have modeled the sign function over the interval of [-1,1]. The modular network architecture used in modeling this function has a 1-1, linear architecture for each expert and a 1-2 architecture for the gating network. As shown in Figure 7 - 7, the modular network is able to approximate the function almost exactly. When looking closely at the gating outputs, we see that the gating is able to split the region into two sections and allows each expert network to approximate one region. In this case, the sum of squared error is trained to less than the 10^{-6} in 34 epochs.

Test #7 Friction Model Approximation II

Test Function: Classical Friction Model

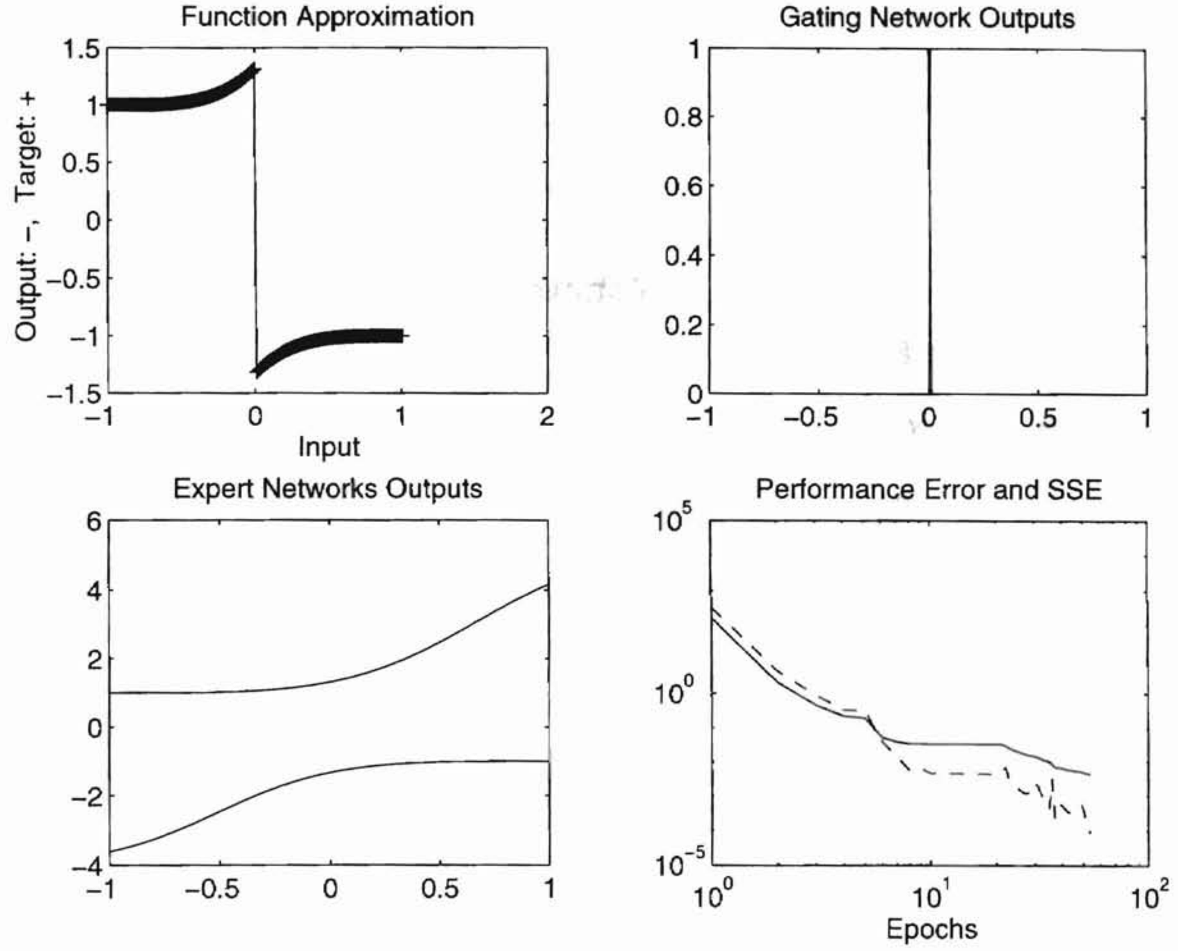


Figure 7 - 8. Modeling Classical Friction Model

In this test, the modular network is used to train the test pattern (+ marks) as shown in the top left of the Figure 7 - 8. This function closely mimics the classical friction model. We can view this function as a force versus velocity of a friction model. Due to the sharp changes at the discontinuity at zero, two-layer networks are used for the expert networks. The Marquardt-Levenberg algorithm is used to train the network until the sum of squared errors is less than 10^{-4}. Again the modular network is able to train this friction function accurately. When we look closely into how the gating and expert networks perform, the gating network divides the region into two sections and assigns an expert network to learn each half of the curve.

From these preliminary tests, it appears that the modular network will perform better than the multilayer network in modeling the friction.

Test #8 Identification: Plant Modeling

In this test, we will use the modular network to identify a single-link pendulum that has a coulomb friction non-linearity. This single link pendulum with coulomb friction is shown in Figure 7 - 9. It is driven by a dc motor with one of its ends attached to the motor shaft. The mathematical model for this pendulum system can be shown as:

$$\frac{\partial^2\theta}{\partial t^2} + 2\frac{\partial\theta}{\partial t} + 10.\sin(\theta) + sign\left(\frac{\partial\theta}{\partial t}\right) = u \qquad (7-9)$$

where

θ is the angle of the pendulum and

u is the current applied to the DC motor.

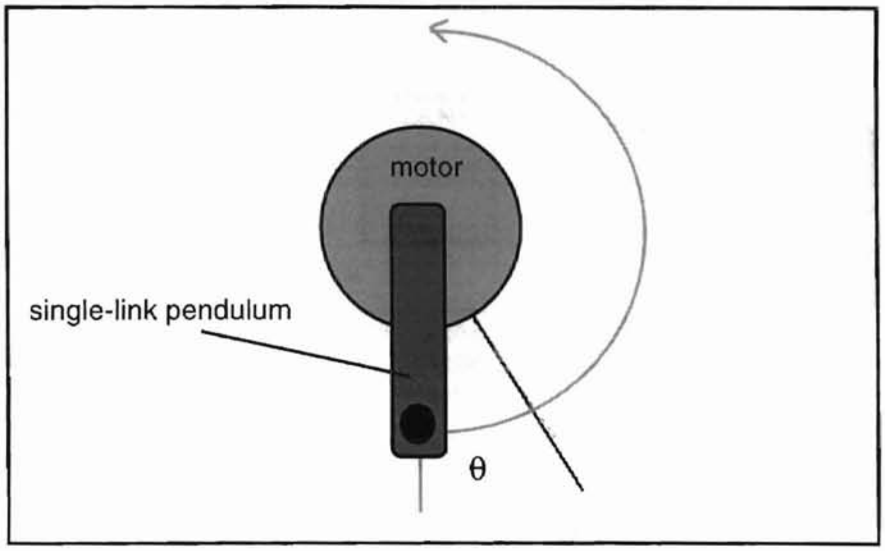


Figure 7 - 9. The Single Linked Pendulum with Coulomb Friction

To identify a model, we mean to train a neural network to follow the real plant as close as possible. The plant can be modeled by using the configuration shown in

Figure 7 - 10. According to system identification theory, a nonlinear plant can be identified by using the current and delayed inputs and outputs relation [14] [15]. Hence, the inputs of the network consist of the current and previous motor control voltages $u(k)$ and $u(k\text{-}1)$ and the current and previous motor positions $y(k)$ and $y(k\text{-}1)$. The output of the network is the next motor position $y(k+1)$. To see how both multilayer feedforward networks and modular networks perform in plant modeling, we will identify the plant using both feedforward networks and modular networks.

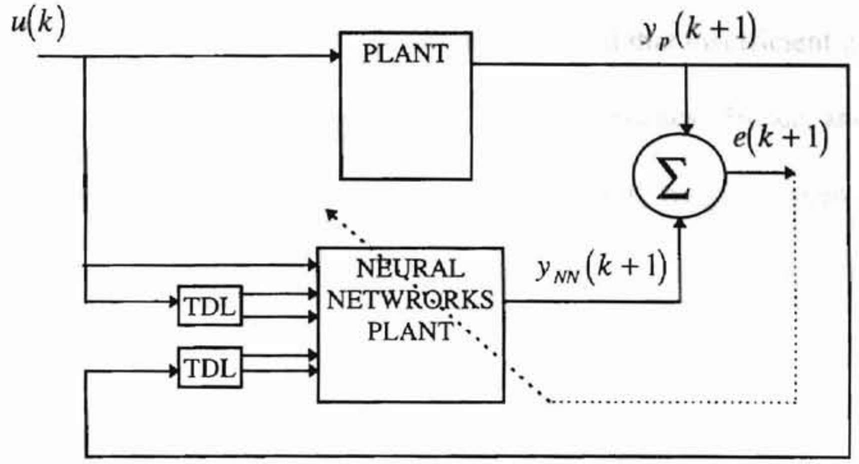


Figure 7 - 10. System Identification of a Plant

Data Sets

In plant modeling, the most important aspect is to sample enough data for the neural network model. This data must be able to cover the entire operating range of the system that you are trying to model. One way to obtain enough data is to use random initial conditions for position and input voltage. Then, allow the system to run for one time step and obtain the future, current and previous time step positions and also the current and previous time steps of input voltage. The single-link pendulum has a full range motion from the straight downward position 0^o to the straight upward position 180^o. Hence, the position is randomly sampled within this range, and the velocity is randomly sampled between -12 and 12. Meanwhile, the input voltage is randomly sampled from -15 to 15. The sampling rate is chosen at 20 samples a second. We will use the alternative training methods (ATS), as described in [15], to train both the modular network and the multilayer network. Initially, two data sets, each consisting of 400 data points, were collected and trained. The results were not satisfactory as it did not pass the parallel test described in the next section. Further investigation revealed that insufficient data were collected at low velocity, where the friction effects are the greatest. Hence, another 100 data points were collected for each data set in the low velocity range between -1 and 1. A total of 500 data points for each set of data was used.

Evaluation Methods

After the neural networks are trained to model the plant, two evaluation methods are performed on the trained model. These methods are the series-parallel test and the parallel test. The series-parallel test, shown in Figure 7 - 11, has the delayed actual plant outputs as the network inputs. On the other hand, the parallel test, shown in Figure 7 - 12, has the delayed network outputs fed back to its input.

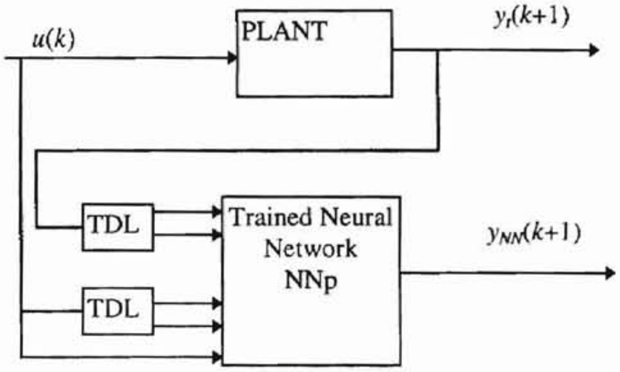


Figure 7 - 11. Series-Parallel Test

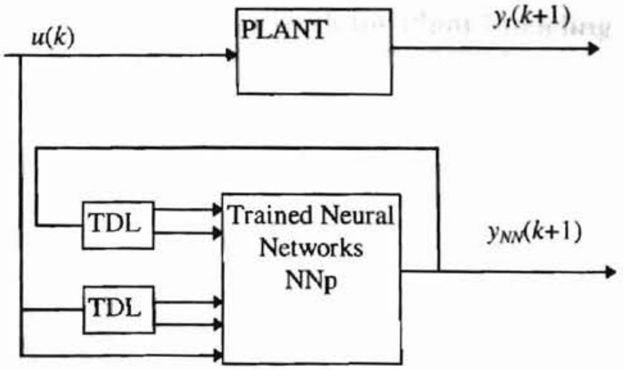


Figure 7 - 12. Parallel Test

It is obvious that the parallel test is a more difficult test, because the trained model does not receive the actual plant input. If the plant model does not exactly match the real plant, then a small error in each time step will accumulate and become a large error in the end. On the other hand, the series parallel test receives the actual time-delayed plant input and therefore has no accumulation error and is much easier to pass. For a trained neural network model to perform well, the trained model must pass both series-parallel and parallel tests.

Modular Network and Multilayer Network for Plant Modeling

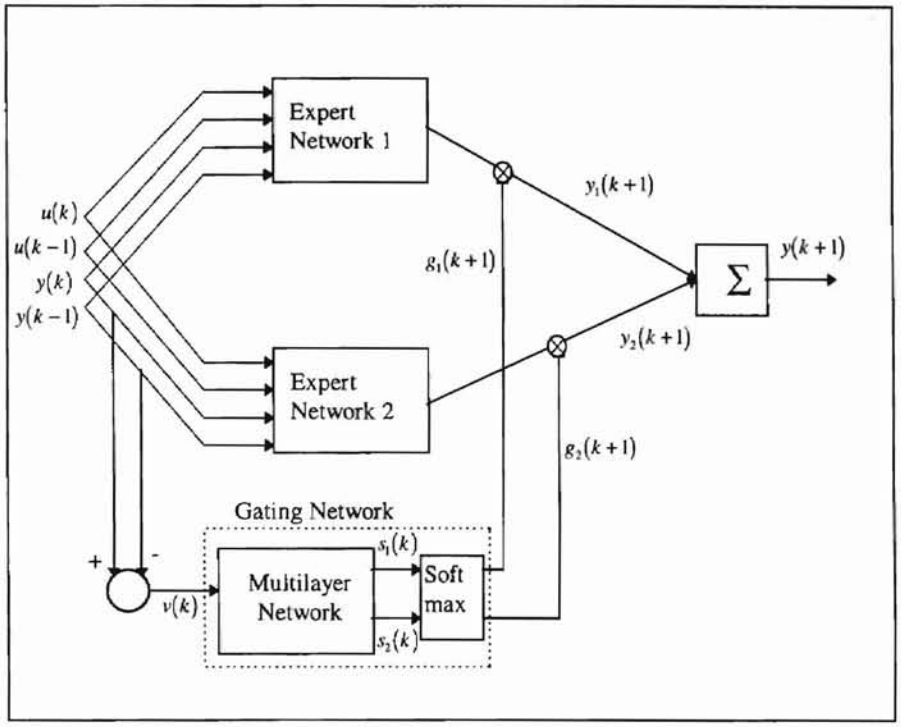


Figure 7 - 13. Modular Network Architecture

The modular network used in plant modeling has 2 expert networks, each with a 4-8-1 architecture, hyperbolic tangent function in the first layer and linear layer in the last layer. The gating network is a 1-2 single layer. Since we know that the friction changes sign when the velocity changes sign, the gating network receives the difference between the current position and the previous position of the plant. Also, the gating network's weights are preset to allow one expert to train for positive velocity and the other expert to train for negative velocity. The network architecture is shown in Figure 7 - 13. Using the newly developed Marquardt-Levenberg algorithm for the modular network and the ATS method, the model is trained. As shown in Figure 7 - 14, the sum of square error (solid line) and the negative performance index (dashed line) both went below 10^{-4}.

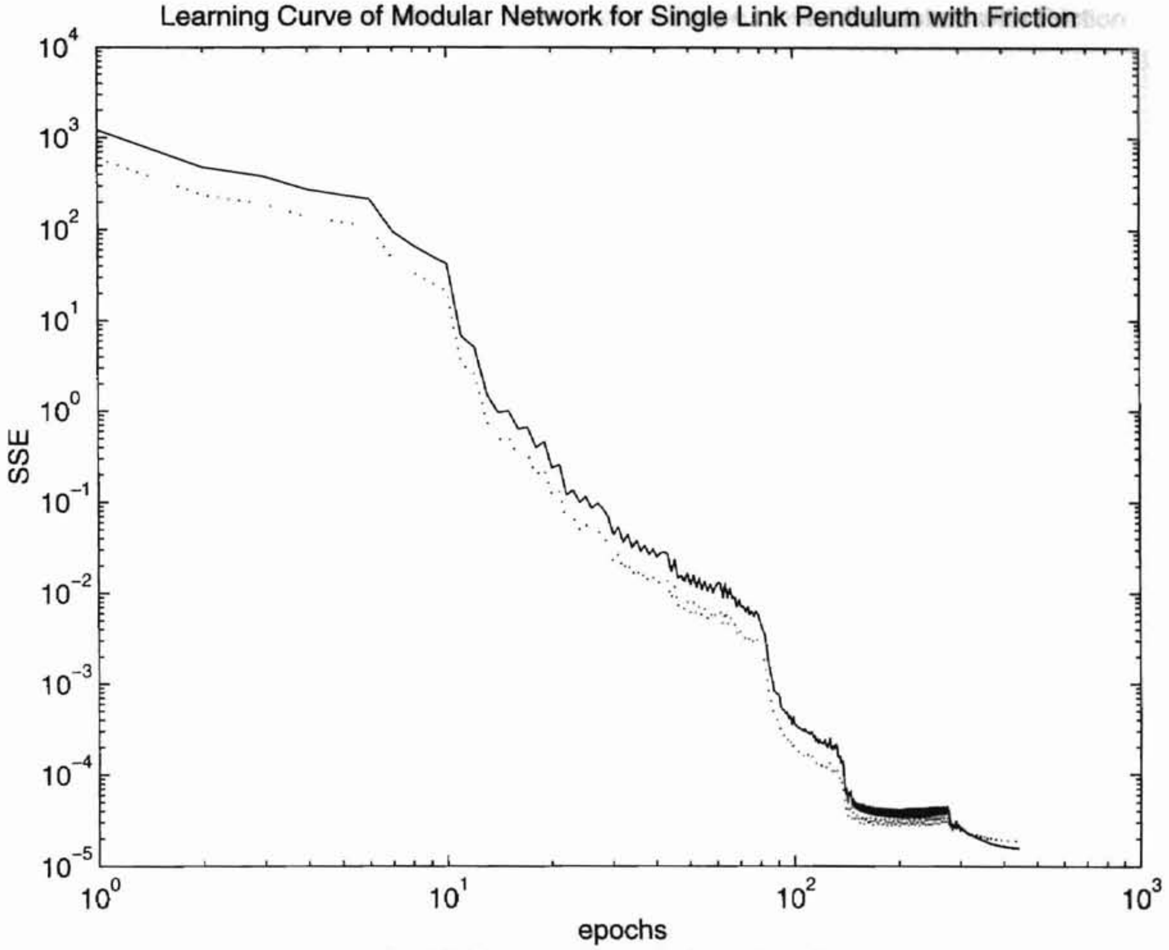


Figure 7 - 14. Modular Networks Learning Curve

Simultaneously, a 4-15-1 multilayer network is also used to model the plant with the same data set that was given to the modular network. Using the Marquardt-Levenberg algorithm for the multilayer network and the ATS method, the 4-15-1 network was trained. Figure 7 - 15 shows the learning curve of the 4-15-1 network. As shown, the learning curve is saturated at about a sum of squared error equal to 10^{-3} which is not as low as the modular network. This could mean a local minimum was reached. However, after retrained the network several times, the same results were achieved. Hence, this could be the global minimum.

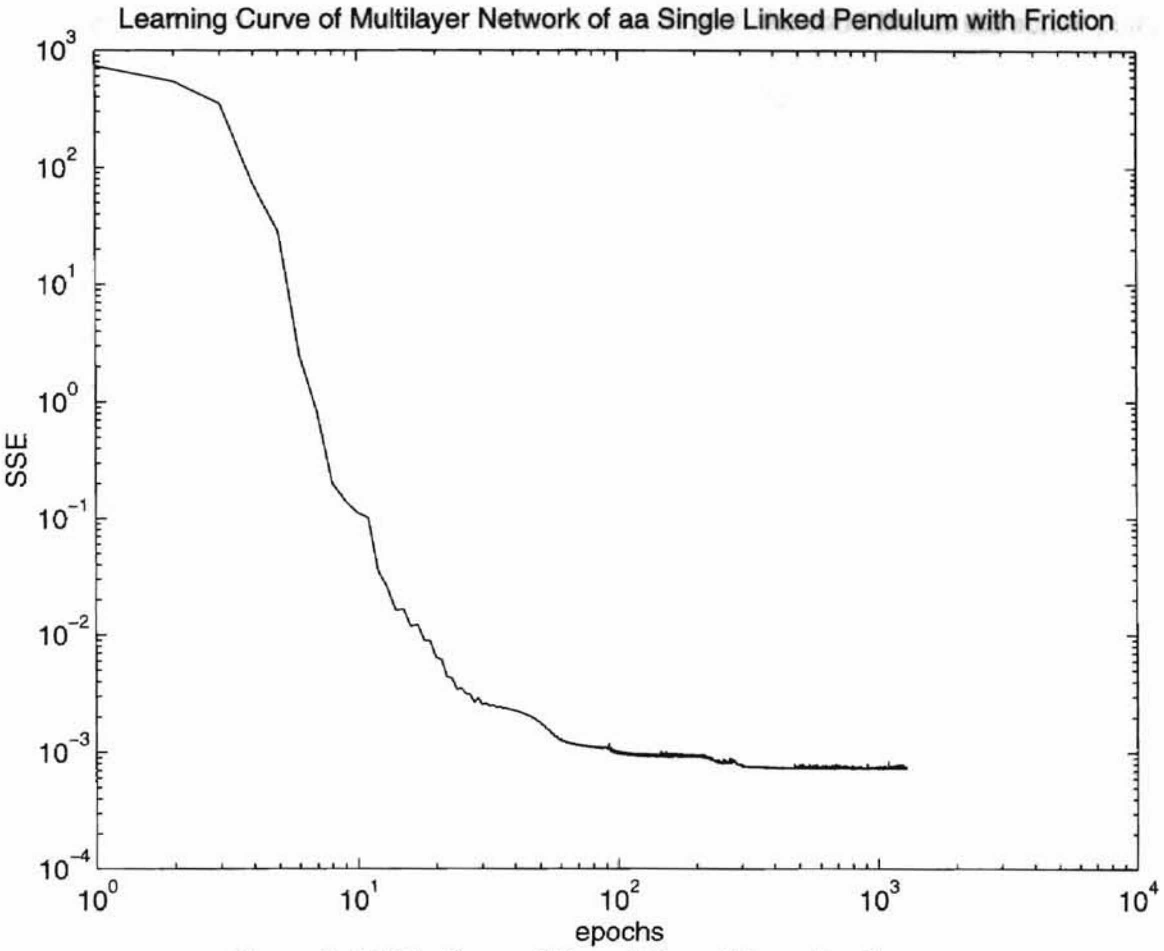


Figure 7 - 15. Feedforward Neural Network Learning Curve

Model Testing

To evaluate the neural network models, both the trained modular network plant model and the trained feedforward network plant model are tested using the series-parallel and parallel tests. As mention above, to accurately model a plant, the trained neural network model must pass both the series-parallel and parallel tests. On both the series-parallel and parallel tests, four test cases were selected: 90° free fall (the top left plot), 10 volt step input with zero initial condition (the top right plot), random initial condition where x1(0)=2.089 and x2(0)=-0.2778 (bottom left plot) and -10 volt pulse response with x1(0)=pi (bottom right plot). On each plot, the solid line is the actual plant response and the dashed line is the trained network response.

Series-Parallel Test

Figure 7 - 16 and Figure 7 - 17 show the series-parallel tests of a trained feedforward network and a trained modular network. As shown in the plot, both trained networks show excellent response on each test; that is why you only see the solid line (actual plant response) but not the dashed line (neural network response). Several other tests were also performed and the results are the same.

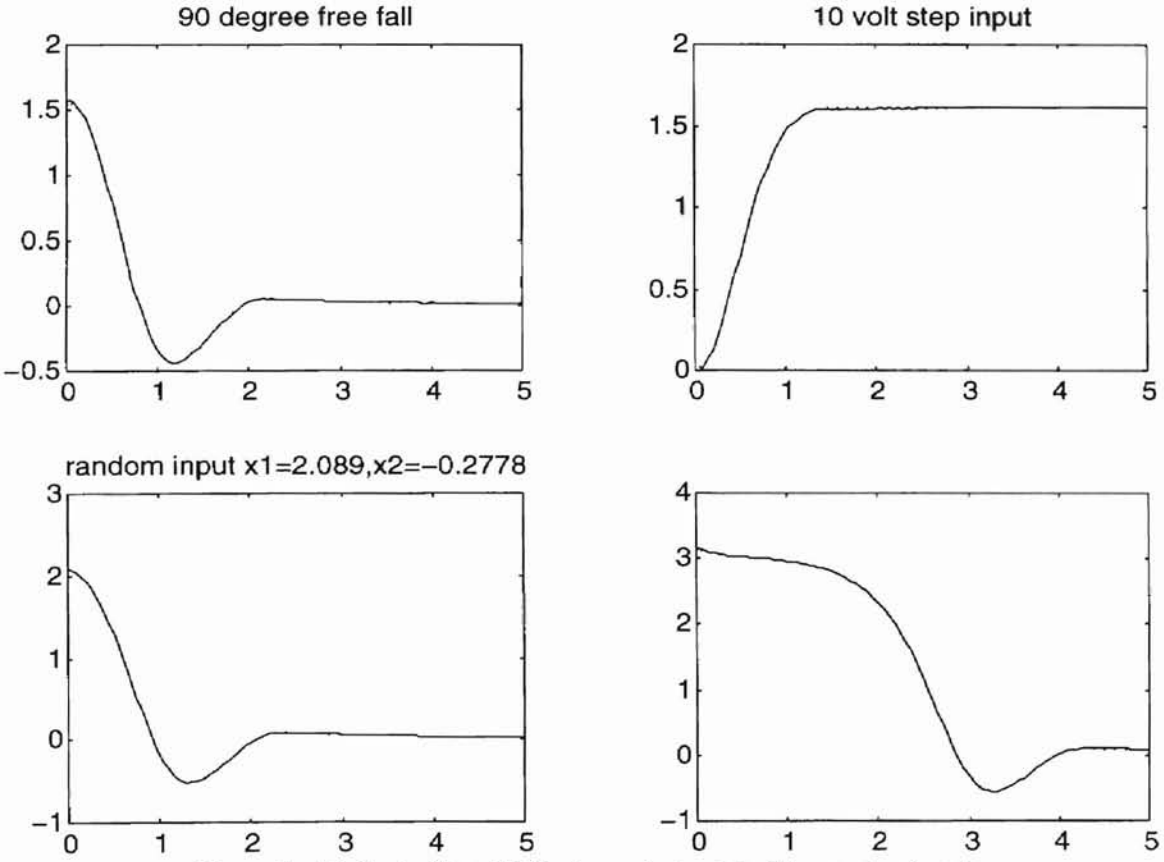


Figure 7 - 16. Series Parallel Test on a trained feedforward network

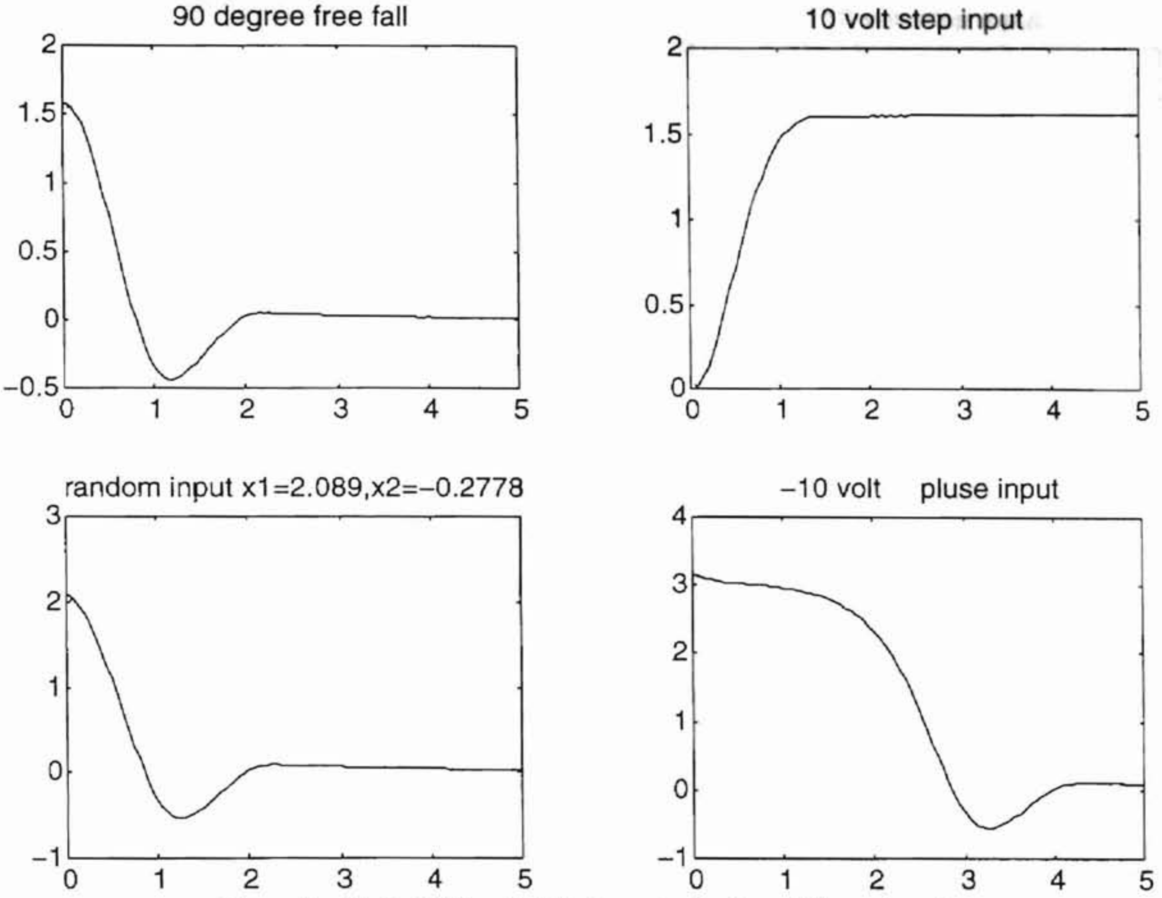


Figure 7 - 17. Serial Parallel Test on a trained modular network

Parallel Test

In the parallel test, the trained feedforward network did not perform very well. As shown in Figure 7 - 18, the trained feedforward network seems to have difficulty in capturing the friction effects. In most of the cases, it did not stick as the real plant does. I retrained the feedforward network plant several times, but the result was always the same. While the trained feedforward network has difficulty in capturing the friction effects, the trained modular network seems to perform fairly well. Figure 7 - 19 shows the parallel tests of the trained modular network. As shown, the trained modular network response closely follows the real plant response. Several other test cases were aiso performed and the results were the same.

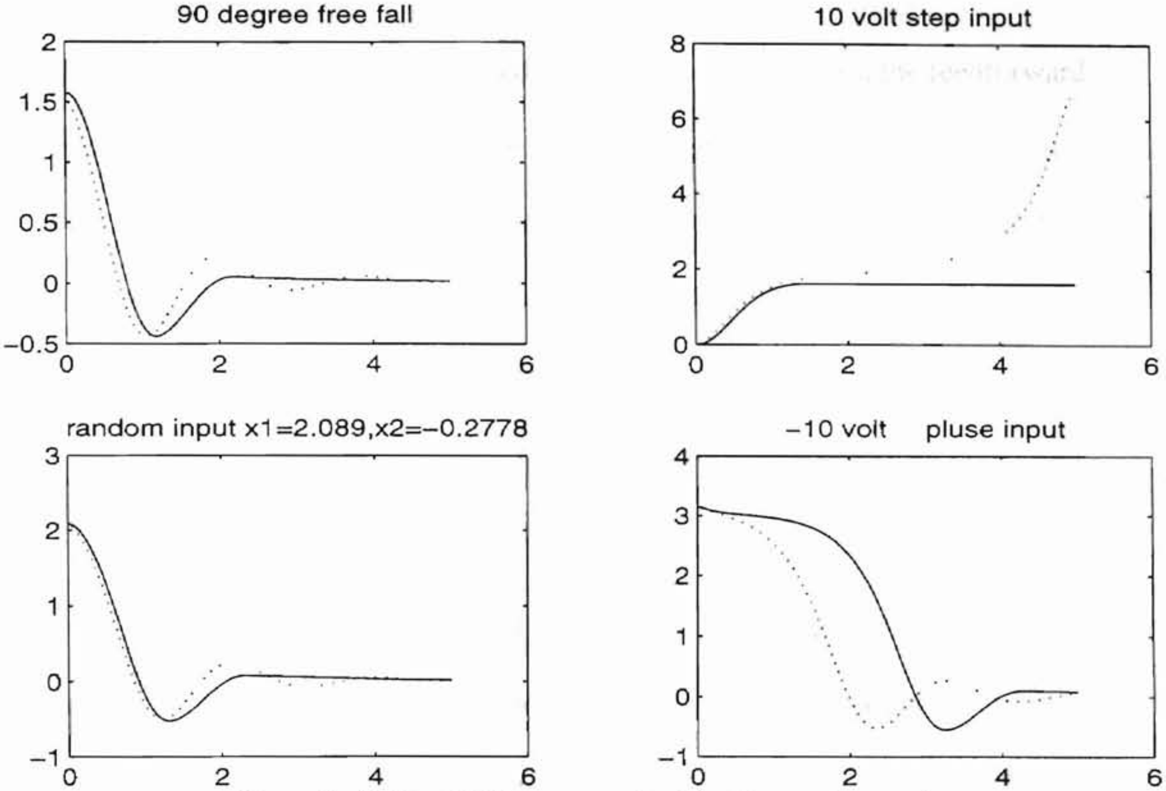


Figure 7 - 18. Parallel Test on a trained feedforward network

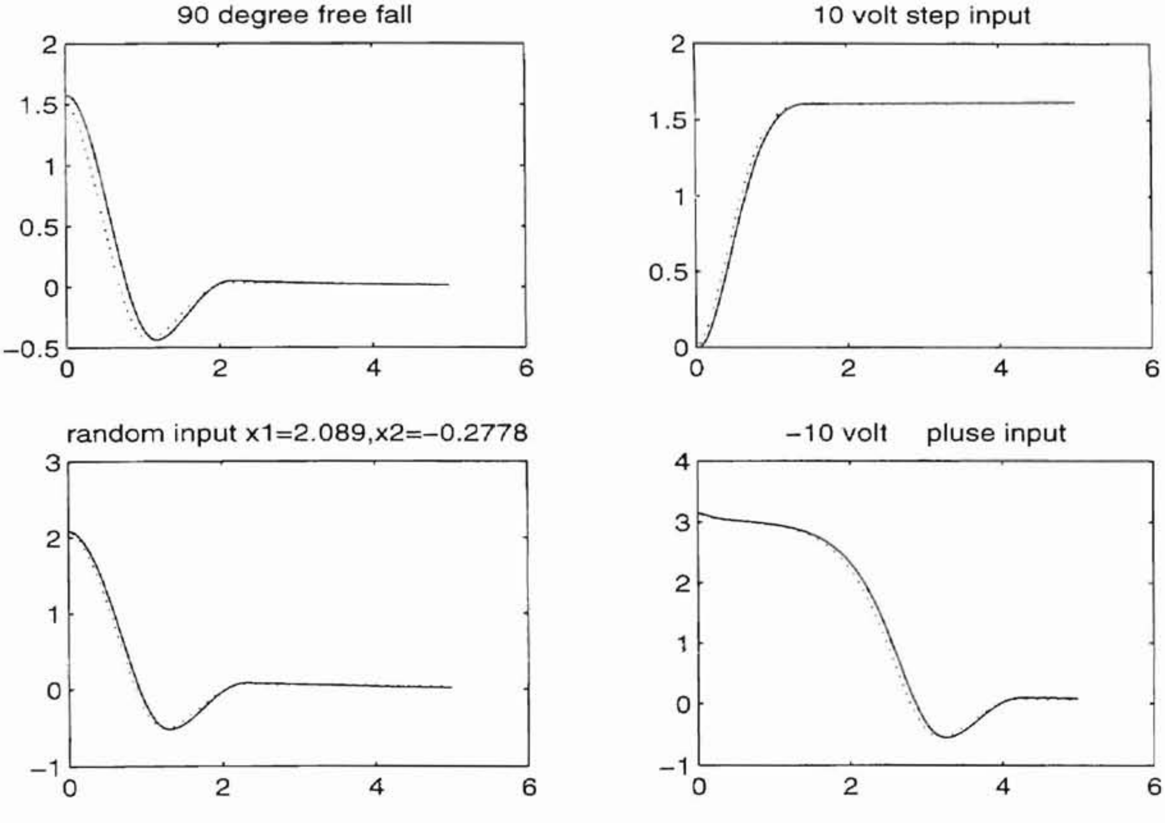


Figure 7 - 19. Parallel Test on a Trained Modular Network

Summary

In this chapter, we have compared the modular network with the feedforward network and have found that the modular network is superior for many applications. However, several tests have indicated that the modular networks can be very hard to train. Hence, two methods of improving the convergence were tested. These methods are the what and where modular network architecture and the weight initialization method. As shown in the tests, these methods do speed up the training of the modular network.

In addition, we also showed some applications of the modular network for implementing discontinuous functions. Several tests have indicated that the modular network can be used for modeling friction. In one particular test, test #8, we have demonstrated an important application of the modular network. We have shown the possibility of using modular networks to capture a friction non-linearity that is embedded in the dynamics of a single-link pendulum. This clearly implies a possibility of using modular network in reducing or eliminating friction.

CHAPTER VIII

CONCLUSION

Starting from the basic building blocks in Chapter II, this research has developed a faster training algorithm for the modular network, described in Chapter V, known as the Marquardt-Levenberg (**ML**) algorithm. It is shown in Chapter VI that the Marquardt-Levenberg algorithm is more than 100 times faster than the Steepest Ascent method (**SA**) and about two times faster than **Rprop**. Hence, it is by far the most promising and the fastest training method for the modular network which has been reported. Another algorithm which has evolved in recent years, the Expected Maximization (**EM**) algorithm, has been incorporated into the training of the hierarchical modular network [16] [17]. This algorithm has been shown to be one to two orders of magnitude faster than the Steepest Ascent (**SA**) method for the hierarchical modular network. Hence, it would be interesting to incorporate the Marquardt-Levenberg algorithm into the hierarchical modular network and compare it with the **EM** algorithm. However, this comparison is beyond the scope of this research.

A test in Chapter VII has revealed that the modular network is far superior than the multilayer network in implementing discontinuous functions. However, the fast convergence of the **ML** training algorithm is sometimes hindered by the modular network architecture, since the performance surface contains many local maximums and local minimums. Hence, two methods of improving the training, the what and where network and the gating weight initialization, are discussed in Chapter VII. The what and where modular network shows significant improvement in the training process, but it can only be used efficiently if one know how the tasks should be divided. Meanwhile, the gating weight initialization also shows significant improvement in the training process, but the weight initialization for the case of more than 2 experts and more than 1 layer in the gating network is still under development. Nevertheless, both methods show promising results.

Since the modular networks are capable of implementing discontinuous functions, several friction functions have been tested. The results have indicated that the modular networks are capable of implementing friction functions. Moreover, it is capable of capturing the friction model that is embedded in another system dynamic, as shown in test 8 in Chapter VII. With these capabilities, the modular network can be used in modeling or possibly controlling a system where the friction is the primary concern. Nonetheless, the use of modular networks in friction compensation will be a promising method in the future.

The use of the ML training algorithm for the modular network has opened up several possible future research areas. First, it would be interesting to incorporate the ML algorithm into the hierarchical modular network and compare it with the EM algorithm. Second, we should further investigate methods for improving the training process, such as setting the initial weights or using the what-and-where modular network. As described above, the modular network training process is hindered by the network architecture.

Specially adapted training algorithms could overcome the problems caused by this unique architecture. Lastly, we can investigate a number of application areas. As noted above, modular networks, when trained with the **ML** algorithm, are capable of implementing discontinuous functions, such as friction models. Hence, it may be possible to use the modular network, trained with the **ML** algorithm, to implement an adaptive friction compensation control system.

REFERENCES

1 Haykin S. (1994). *Neural Networks, A Comprehensive Foundation*. Macmillan: NY.

2 Martin T. Hagan, Howard Demuth & Mark Beale. (1996*). Neural Network Design*. PWS Publishing :MA.

3 Hornik, K., Stinchcombe, M. & White, H. (1989*). Multilayer Feedforward Networks are Universal Approximators.* Neural Networks, 2, pg. 183-192.

4 Jacobs, R.A., and M.I. Jordan, (1993). Learning Piecewise Control Strategies in a Modular Neural Network Architecture. *IEEE Transaction of System, Man, and Cybernetics.* vol. 23, 337-345.

5 Bridle, J. (1989). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In F. Fogelman-Soulie & J, Herault (eds.), *Neuro Computing: Algorithms, Architectures, and Applications.* NY: Springer-Verlag.

6 Bridle, J.S. (1990). Training Stochastic Model Recognition Algorithms as Networks can lead to Maximum Mutual Information Estimation of Parameters. *Advances in Neural Information Processing System* 2 (D.D. Touretzky, ed.), pp.211-217. San Mateo, CA: Morgan Kaufmann.

7 Sutton, R.S. (1986). Two Problems with Backpropagation and Other Steepest Descent Learning Procedures for Networks. *Pro. Eighth Annual Conference Cognitive Science Society.* 823-831.

8 Jacobs, R.A. and Jordan M.I. (1991). Task Decomposition through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *Neural Computation*, vol. 3, 79-87.

9 Jacobs, R.A. and M.I. Jordan, (1992). "Recent Development in Supervised Learning" *Tutorial Notes. International Joint Conference on Neural Networks,* Baltimore, MD.

10 Wilks, S.S., (1962). *Mathematical Statistics*. New York: Wiley.

11 Martin T. Hagan & Menhaj, Nov. (1994) " Training Feedforward Neural Networks with Marquardt-Levenberg Algorithm" *IEEE Transaction on Neural Networks,* vol. 5 No.6.

12 L.E. Scales (1985), *Introduction to Non-linear Optimization*. New York: Springer-Verlag.

13 Martin Riedmiller, (1994). "Advanced Supervised Learning in Multilyer Perceptrons - From Backpropagation to Adaptive Learning Algorithms.", Denmark

14 Narendra, Kumpati S. (1989). *Stable Adaptive Systems*. Prentice Hall. N.J.

15 Yang, Wei Chung. (1994) *Neurocontrol using Dynamic Learning*. OSU Theses .

16 M. I. Jordan and Robert A. Jacobs. (1994). Hierarchical Mixtures of Experts and the EM Algorithm", *Neural Computation* 6: 181-214.

17 Jordan M.I. & Xu L. (1993). Convergence Results for the EM Approach to Mixtures of Experts Architectures. *Neural Networks*.

APPENDIXES:

A) Pre-steps

Before we go through the computation of gradient matrix and Hessian matrix, we will compute several preliminary steps that we will need in the computing the Gradient and Hessian matrix.

A-I) Calculate $\frac{\partial J}{\partial \mathbf{y}_i}$

From equation (3-1), we define the performance index as

$$J = \ln\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)} \right). \tag{A - I - 1}$$

Then, differentiating the performance index with respect to the outputs of the i^{th} expert network yields,

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{y}_i} &= \frac{\frac{\partial}{\partial \mathbf{y}_i}\left(g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)} \right)}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)}} \\ &= \frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)}} \left(\mathbf{y}^* - \mathbf{y}_i \right) \qquad \text{if } i = j \\ &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } i \neq j \end{aligned} \tag{A - I - 2}$$

$$\boxed{\frac{\partial J}{\partial \mathbf{y}_i} = h_i\left(\mathbf{y}^* - \mathbf{y}_i \right)} \tag{A - I - 3}$$

A-II) Calculate $\frac{\partial J}{\partial u_i}$

Again, using the log-likelihood performance index, (A- I - 1), we differentiate it with respect to the i^{th} output of the gating network.

$$\begin{aligned}\frac{\partial J}{\partial u_i} &= \frac{\frac{\partial}{\partial u_i}\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}\right)}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}} \\ &= \frac{\sum_{j=1}^{N} \frac{\partial g_j}{\partial u_i} \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}}\end{aligned} \qquad (\text{A - II - 1})$$

From Appendix A-IV, the differentiation of softmax function with respect to the i^{th} gating outputs is calculated as:

$$\begin{aligned}\frac{\partial g_j}{\partial u_i} &= g_j - g_j^{\ 2} \qquad \text{if } j = i \\ &= -g_j g_i \qquad \text{if } j \neq i\end{aligned} \qquad (\text{A - II - 2})$$

Substituting equation (A-II-2) into (A-II-1), we have

$$\begin{aligned}\frac{\partial J}{\partial u_i} &= \frac{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)} - \sum_{j=1}^{N} g_j g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}} \qquad \text{if } \ i = j \\ &= \frac{-\sum_{j=1}^{N} g_j g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}} \qquad \text{if } \ i \neq j\end{aligned} \qquad .(\text{A - II - 3})$$

Combining the above equations, we have

$$\frac{\partial J}{\partial u_i} = \frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_i)} - g_i \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T(\mathbf{y}^* - \mathbf{y}_j)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T(\mathbf{y}^* - \mathbf{y}_j)}} \qquad \text{(A - II - 4)}$$

and finally

$$\boxed{\frac{\partial J}{\partial u_i} = h_i - g_i}. \qquad \text{(A - II - 5)}$$

A-III) Calculate the $\frac{\partial h_i}{\partial \mathbf{y}_k}$

We have defined the posterior probability as:

$$h_i = \frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_i)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T(\mathbf{y}^* - \mathbf{y}_j)}}. \qquad \text{(A - III - 1)}$$

Now, we would like to differentiate them with respect to the output of the k^{th} expert network. Let

$$h_i = \frac{u(\mathbf{y}_i)}{v(\mathbf{y}_j)} \qquad \text{(A - III - 2)}$$

where

$$u(\mathbf{y}_i) = g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T(\mathbf{y}^* - \mathbf{y}_1)} \quad \text{,and} \qquad v(\mathbf{y}_j) = \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)(\mathbf{y}^* - \mathbf{y}_j)}.$$

Then, using the chain rule, we differentiate the numerator, $u(\mathbf{y}_i)$, and denominator, $v(\mathbf{y}_j)$:

$$\frac{\partial h_i}{\partial \mathbf{y}_k} = \frac{1}{v(\mathbf{y}_i)^2}\left[v(\mathbf{y}_i)\frac{\partial u(\mathbf{y}_i)}{\partial \mathbf{y}_k} - u(\mathbf{y}_i)\frac{\partial v(\mathbf{y}_i)}{\partial \mathbf{y}_k}\right]. \qquad \text{(A - III - 3)}$$

Calculate the derivative inside:

$$\begin{aligned}\frac{\partial u(\mathbf{y}_i)}{\partial \mathbf{y}_k} &= \frac{\partial}{\partial \mathbf{y}_k}\left(g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\right)\\ &= g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\left[\mathbf{y}^* - \mathbf{y}_i\right] \qquad \text{if} \quad i = k \qquad\qquad \text{(A - III - 4)}\\ &= 0 \qquad \text{if } i \neq k\end{aligned}$$

and

$$\begin{aligned}\frac{\partial v(\mathbf{y}_j)}{\partial \mathbf{y}_k} &= \frac{\partial}{\partial \mathbf{y}_k}\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}\\ &= g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\left[\mathbf{y}^* - \mathbf{y}_i\right] \qquad \text{if} \quad i = k \quad . \qquad\qquad \text{(A - III - 5)}\\ &= g_k \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_k)^T(\mathbf{y}^*-\mathbf{y}_k)}\left[\mathbf{y}^* - \mathbf{y}_k\right] \qquad \text{if } i \neq k\end{aligned}$$

Substituting these back into equation (A-III-3), $\frac{\partial h_i}{\partial \mathbf{y}_k}$ becomes

$$\begin{aligned}\frac{\partial h_i}{\partial \mathbf{y}_k} &= \frac{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)(\mathbf{y}^*-\mathbf{y}_j)}\, g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\left[\mathbf{y}^* - \mathbf{y}_i\right] - \left(g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\right)^2\left[\mathbf{y}^* - \mathbf{y}_i\right]}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)(\mathbf{y}^*-\mathbf{y}_j)}\right)^2} \quad \text{if } i = k\\ &= \frac{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)(\mathbf{y}^*-\mathbf{y}_j)}.0 - g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\, g_k \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_k)^T(\mathbf{y}^*-\mathbf{y}_k)}\left[\mathbf{y}^* - \mathbf{y}_k\right]}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)(\mathbf{y}^*-\mathbf{y}_j)}\right)^2} \quad \text{if } i \neq k\end{aligned}$$

$$\boxed{\begin{aligned} \frac{\partial h_i}{\partial \mathbf{y}_k} &= \left(h_i - h_i^{\,2}\right)\left(\mathbf{y}^{*} - \mathbf{y}_i\right) && \text{if } i = k \\ &= -h_i h_k \left(\mathbf{y}^{*} - \mathbf{y}_k\right) && \text{if } i \neq k \end{aligned}} \qquad (\text{A - III - 6a,b})$$

Also, from the above equation, we can shown that

$$\boxed{\frac{\partial}{\partial u_i}\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^{*}-\mathbf{y}_j)^T(\mathbf{y}^{*}-\mathbf{y}_j)}\right) = g_i \exp^{-\frac{1}{2}(\mathbf{y}^{*}-\mathbf{y}_i)^T(\mathbf{y}^{*}-\mathbf{y}_i)} - g_i \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^{*}-\mathbf{y}_j)^T(\mathbf{y}^{*}-\mathbf{y}_j)}} \qquad (\text{A - III - 7})$$

A - IV) Calculate the $\frac{\partial g_i}{\partial u_k}$

The softmax function, equation (3-9), is defined as:

$$g_i = \frac{\exp(u_i)}{\sum_{m=1}^{N} \exp(u_m)} . \qquad (\text{A - IV - 1})$$

By differentiating the softmax function with respect to the k^{th} output of the gating network, we obtain

$$\begin{aligned} \frac{\partial g_i}{\partial u_k} &= \frac{\partial}{\partial u_k}\left(\frac{\exp^{u_i}}{\sum_{m=1}^{N} \exp^{u_m}}\right) \\ &= \frac{\exp^{u_i} \sum_{m=1}^{N} \exp^{u_m} - \exp^{u_i} \exp^{u_i}}{\left(\sum_{m=1}^{N} \exp^{u_m}\right)^2} && \text{if } i = k . \qquad (\text{A - IV - 2}) \\ &= \frac{0. \sum_{m=1}^{N} \exp^{u_m} - \exp^{u_i} \exp^{u_k}}{\left(\sum_{m=1}^{N} \exp^{u_m}\right)^2} && \text{if } i \neq k \end{aligned}$$

Then, combining them together, we have

$$\boxed{\begin{aligned}\frac{\partial g_i}{\partial u_k} &= g_i - g_i^{\,2} && \text{if } i = k \\ &= -g_i g_k && \text{if } i \neq k\end{aligned}} \qquad (\text{A - IV - 3})$$

A-V) Calculate the $\frac{\partial h_i}{\partial u_j}$ term

The posterior probability is defined as:

$$h_i = \frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)}}. \qquad (\text{A - V - 1})$$

By differentiating the a posteriori probability with respect to the j^{th} output of the gating we get:

$$\frac{\partial h_i}{\partial u_j} = \frac{\partial}{\partial u_j}\left(\frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)}}{\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)}} \right) \qquad (\text{A - V - 2})$$

$$= \frac{\frac{\partial g_i}{\partial u_j} \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)} \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)} - g_i \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_i)^T (\mathbf{y}^* - \mathbf{y}_i)} \frac{\partial}{\partial u_j}\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)} \right)}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^* - \mathbf{y}_j)^T (\mathbf{y}^* - \mathbf{y}_j)} \right)^2}.$$

From equation (A-IV-3) and (A-III-7), we have

$$\begin{aligned}\frac{\partial g_j}{\partial u_i} &= g_j - g_j^{\,2} && \text{if } j = i \\ &= -g_j g_i && \text{if } j \neq i\end{aligned} \qquad (\text{A - V - 3})$$

and

$$\frac{\partial}{\partial u_i}\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(y^* - y_j)^T(y^* - y_j)}\right) = g_i \exp^{-\frac{1}{2}(y^* - y_i)^T(y^* - y_i)} - g_i \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(y^* - y_j)^T(y^* - y_j)} . \quad (\text{A - V - 4})$$

We can show that

if $j = i$, then

$$\frac{\partial h_i}{\partial u_j} = \frac{\left[g_i - g_i^{\,2}\right]\exp^{-\frac{1}{2}(y^* - y_i)^T(y^* - y_i)} \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(y^* - y_j)^T(y^* - y_j)}}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(y^* - y_j)^T(y^* - y_j)}\right)^2} - \frac{g_i \exp^{-\frac{1}{2}(y^* - y_i)^T(y^* - y_i)}\left(g_i \exp^{-\frac{1}{2}(y^* - y_i)^T(y^* - y_i)} - g_i \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(y^* - y_j)^T(y^* - y_j)}\right)}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(y^* - y_j)^T(y^* - y_j)}\right)^2}, \quad (\text{A V - 5})$$

and can be reduced to

$$\begin{aligned}\frac{\partial h_i}{\partial u_j} &= h_i - g_i h_i - h_i(h_i - g_i)\\ &= h_i - g_i h_i - h_i^2 + h_i g_i\end{aligned}$$

$$\frac{\partial h_i}{\partial u_j} = h_i - h_i^2 \qquad \text{if } i = j. \qquad (\text{A - V - 6})$$

If $i \neq j$, then

$$\begin{aligned}\frac{\partial h_i}{\partial u_j} &= \frac{-g_i g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)} \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}\right)^2} \\ &\quad - \frac{g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)}\left(g_i \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_i)^T(\mathbf{y}^*-\mathbf{y}_i)} - g_i \sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}\right)}{\left(\sum_{j=1}^{N} g_j \exp^{-\frac{1}{2}(\mathbf{y}^*-\mathbf{y}_j)^T(\mathbf{y}^*-\mathbf{y}_j)}\right)^2}\end{aligned} \quad ,(\text{A - V - 7})$$

and it can be reduced to

$$\begin{aligned}\frac{\partial h_i}{\partial u_j} &= -g_j h_i - h_i\left(h_j - g_j\right) \\ &= -h_i h_j \qquad \text{if } i \neq j\end{aligned}. \qquad (\text{A - V - 8})$$

Therefore,

$$\boxed{\begin{aligned}\frac{\partial h_i}{\partial u_j} &= h_i - h_i^{\,2} \qquad \text{if } i = j \\ &= -h_i h_j \qquad \text{if } i \neq j\end{aligned}}. \qquad (\text{A - V - 9})$$

B) Gradient Calculation in Modular Network

To calculate the gradient of the modular network, we have to compute the gradient in the expert and gating networks simultaneously. Define **x** as the total weights and biases in the expert networks, $\mathbf{w}_1 \ldots \mathbf{w}_N$, and gating network, **v**, such that

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_N \\ \mathbf{z} \end{bmatrix}, \qquad (\text{B - 1})$$

where

$$\mathbf{r}_i = \begin{bmatrix} w^1_{i_{1,1}} & w^1_{i_{1,2}} & \cdots & w^1_{i_{S^1,R}} & b^1_{i_1} & \cdots & b^1_{i_{S^1}} & w^2_{i_{1,1}} & \cdots & b^M_{i_{S^M}} \end{bmatrix}^T \qquad (\text{B - 2})$$

contains the weights and biases in i^{th} expert network and

$$\mathbf{z} = \begin{bmatrix} v^1_{1,1} & v^1_{1,2} & \cdots & v^1_{S^1,R} & q^1_1 & \cdots & q^1_{S^1} & v^2_{1,1} & \cdots & q^P_{S^P} \end{bmatrix}^T . \qquad (\text{B - 3})$$

Therefore, the total gradient in the modular network is

$$\frac{\partial J}{\partial \mathbf{x}} = \begin{bmatrix} \dfrac{\partial J}{\partial \mathbf{r}_1} \\ \dfrac{\partial J}{\partial \mathbf{r}_2} \\ \vdots \\ \dfrac{\partial J}{\partial \mathbf{r}_N} \\ \dfrac{\partial J}{\partial \mathbf{z}} \end{bmatrix}. \qquad (\text{B - 4})$$

In the following, we will show the gradient calculations in both the expert and the gating networks.

B-I) Gradient Calculation in Expert Networks

The weights and biases in the i^{th} expert network are defined as $\mathbf{r}_i$. Take note that $\mathbf{r}_i$ consists of all the weights and biases in i^{th} expert stack together in one long vector. Therefore, the gradient calculation for the expert networks involves computing the derivative of the performance index with respect to all the weights and biases. Since the performance index is an indirect function of the weights and biases in the expert networks, we have to use the chain rule to relate them. The equation below shows the chain rule:

$$\frac{\partial J}{\partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_k}{\partial \mathbf{r}_i}\right)^T \frac{\partial J}{\partial \mathbf{y}_k} \qquad \text{(B - I - 1)}$$

Note that $i \neq k$ gives

$$\frac{\partial \mathbf{y}_k}{\partial \mathbf{r}_i} = 0, \qquad \text{(B - I - 2)}$$

because each expert is independent of one another. Therefore, the only case which has a nonzero value is when $i = k$. When $i = k$ this gives

$$\frac{\partial J}{\partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \frac{\partial J}{\partial \mathbf{y}_i}. \qquad \text{(B - I - 3)}$$

Since we use a multilayer network for the expert network, the $\frac{d\mathbf{y}_i}{d\mathbf{r}_i}$ term can be obtained using backpropagation from Chapter III. From equation (A-I-3), we have shown that

$$\frac{\partial J}{\partial \mathbf{y}_i} = h_i\left(\mathbf{y}^* - \mathbf{y}_i\right). \qquad \text{(B - I - 4)}$$

This gives

$$\boxed{\frac{\partial J}{\partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T h_i\left(\mathbf{y}^* - \mathbf{y}_i\right)} \qquad \text{(B - I - 5)}$$

B-II) Gradient Calculation in the Gating Network

The weights and biases in the gating network are denoted as $\mathbf{z}$. Like the expert networks, the $\mathbf{z}$ consists of all the weights and biases in the gating networks stacked together in one long vector. Therefore, the gradient calculation in the gating network involves computing the derivative of the performance index with respect to all the weights in gating network. As with the expert network, the performance index is an indirect function of the $\mathbf{z}$. Hence, we need to use the chain rule to relate them:

$$\frac{\partial J}{\partial \mathbf{z}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}} \right)^T \frac{\partial J}{\partial \mathbf{u}} \qquad \text{(B - II - 1)}$$

where

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix} \qquad \text{(B - II - 2)}$$

is the output of the gating network.

Since we also use a multilayer network as the gating network, the $\frac{\partial \mathbf{u}}{\partial \mathbf{z}}$ term can be obtained using backpropagation. From equation (A-II-5), we have shown that

$$\frac{\partial J}{\partial u_i} = h_i - g_i . \qquad \text{(B - II - 3)}$$

We can lump these equations into one column vector as

$$\frac{\partial J}{\partial \mathbf{u}} = \mathbf{h} - \mathbf{g} \qquad (\mathrm{B-II-4})$$

where

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_N \end{bmatrix} \text{ and } \mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_N \end{bmatrix}. \qquad (\mathrm{B-II-5})$$

This gives

$$\boxed{\frac{\partial J}{\partial \mathbf{z}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T (\mathbf{h} - \mathbf{g})} \qquad (\mathrm{B-II-6})$$

B-III) Total Gradient in Modular Network

With the above derivation, the total gradient in the modular network is given as

$$\boxed{\frac{\partial J}{\partial \mathbf{x}} = \left[\left(\frac{\partial \mathbf{y}_1}{\partial \mathbf{r}_1}\right)^T h_1(\mathbf{y}^* - \mathbf{y}_1) \;\; \left(\frac{\partial \mathbf{y}_2}{\partial \mathbf{r}_2}\right)^T h_2(\mathbf{y}^* - \mathbf{y}_2) \;\; \cdots \;\; \left(\frac{\partial \mathbf{y}_N}{\partial \mathbf{r}_N}\right)^T h_N(\mathbf{y}^* - \mathbf{y}_N) \;\; \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T (\mathbf{h} - \mathbf{g})\right]^T} \qquad (\mathrm{B-II-7})$$

C) Hessian Matrix Calculation

Recall the total weights, **x**, in the modular network in equation (B-1). By computing the second derivative of the performance index with respect to the total weights in modular network, **x**, the Hessian matrix gives

$$\frac{\partial^2 J}{\partial \mathbf{x}^2}=\begin{bmatrix} \frac{\partial^2 J}{\partial \mathbf{r}_1{}^2} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{r}_1} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{r}_1} & \frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_1} \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{r}_2} & \frac{\partial^2 J}{\partial \mathbf{r}_2{}^2} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{r}_2} & \frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_2} \\ \vdots & \vdots & \ddots & & \vdots \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{r}_N} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{r}_N} & & \frac{\partial^2 J}{\partial \mathbf{r}_N{}^2} & \frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_N} \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{z}} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{z}} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{z}} & \frac{\partial^2 J}{\partial \mathbf{z}^2} \end{bmatrix}. \qquad (C-1)$$

In the following, we will show step by step calculations of each term in the Hessian matrix for the modular network.

C-I) The $\frac{\partial^2 J}{\partial \mathbf{r}_i \partial \mathbf{r}_j}$ term

By $\frac{\partial^2 J}{\partial \mathbf{r}_i \partial \mathbf{r}_j}$, we mean the upper left terms in the Hessian matrix (C-1):

$$\begin{bmatrix} \frac{\partial^2 J}{\partial \mathbf{r}_1{}^2} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{r}_1} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{r}_1} \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{r}_2} & \frac{\partial^2 J}{\partial \mathbf{r}_2{}^2} & & \frac{\partial^2 J}{\partial \mathbf{r}_N \partial \mathbf{r}_2} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \mathbf{r}_1 \partial \mathbf{r}_N} & \frac{\partial^2 J}{\partial \mathbf{r}_2 \partial \mathbf{r}_N} & \cdots & \frac{\partial^2 J}{\partial \mathbf{r}_N{}^2} \end{bmatrix}. \qquad (C-I-1)$$

Recall equation (B-I-3), which gives

$$\frac{\partial J}{\partial \mathbf{r}_i}=\left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \frac{\partial J}{\partial \mathbf{y}_i}. \qquad (C-I-2)$$

We can express each element of $\frac{\partial J}{\partial \mathbf{r}_i}$ as the following summation:

$$\frac{\partial J}{\partial r_{k_i}} = \sum_j \frac{\partial y_{j_i}}{\partial r_{k_i}} \frac{\partial J}{\partial y_{j_i}} . \qquad \text{(C - I - 3)}$$

Taking the derivative with respect to the experts' weights using the chain rule again, we will have the following equation:

$$\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial r_{k_i}}\right] = \sum_j \left(\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]\frac{\partial J}{\partial y_{j_i}} + \frac{\partial y_{j_i}}{\partial r_{k_i}} \frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial y_{j_i}}\right]\right). \qquad \text{(C - I - 4)}$$

We expand the second term on the right hand side as:

$$\begin{aligned} \frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial y_{j_i}}\right] &= \sum_q \sum_p \frac{\partial\left[\frac{\partial J}{\partial y_{j_i}}\right]}{\partial y_{p_q}} \frac{\partial y_{p_q}}{\partial r_{m_l}} \\ &= \sum_q \sum_p \frac{\partial}{\partial y_{p_q}}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial y_{p_q}}{\partial r_{m_l}} \end{aligned} . \qquad \text{(C - I - 5)}$$

If $q \neq l$,

$$\frac{\partial y_{p_q}}{\partial r_{m_l}} = 0 \qquad \text{(C - I - 6)}$$

because the weights of the l^{th} expert have no direct relation with q^{th} expert's output.

Equation (C - I - 5) then simplifies to

$$\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial y_{j_i}}\right] = \sum_p \frac{\partial}{\partial y_{p_l}}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial y_{p_l}}{\partial r_{m_l}} . \qquad \text{(C - I - 7)}$$

Then, equation (C-I-4) becomes

$$\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j \frac{\partial}{\partial r_{m_l}}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]\frac{\partial J}{\partial y_{j_i}}+\sum_j \frac{\partial y_{j_i}}{\partial r_{k_i}}\sum_p \frac{\partial}{\partial y_{p_l}}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial y_{p_l}}{\partial r_{m_l}} \qquad (\text{C - I - 8})$$

For the first term in the equation (C-I-8),

if $l \neq i$,

$$\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]=0 \qquad (\text{C - I - 9})$$

because the weights of the l^{th} expert are not related to the weights of the i^{th} expert.

Equation (C - I - 8) thus reduces to

$$\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j \cancelto{0}{\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]}\frac{\partial J}{\partial y_{j_i}}+\sum_j \frac{\partial y_{j_i}}{\partial r_{k_i}}\sum_p \frac{\partial}{\partial y_{p_l}}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial y_{p_l}}{\partial r_{m_l}} \quad . \qquad (\text{C - I - 10})$$

Now, if $l = i$, the first term in equation (C - I - 8) becomes

$$\sum_j \frac{\partial}{\partial r_{m_i}}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]\frac{\partial J}{\partial y_{j_i}}=\sum_j \frac{\partial^2 y_{j_i}}{\partial r_{m_i}\partial r_{k_i}}\frac{\partial J}{\partial y_{j_i}}=\mathbf{H}(\mathbf{x}), \qquad (\text{C - I - 11})$$

If we assume that **H(x)** is small, then we can approximate the Hessian matrix as

$$\frac{\partial}{\partial r_{m_l}}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j \frac{\partial y_{j_i}}{\partial r_{k_i}}\sum_p \frac{\partial^2 J}{\partial y_{p_l}\partial y_{j_i}}\frac{\partial y_{p_l}}{\partial r_{m_l}}. \qquad (\text{C - I - 12})$$

In fact, we can express the Hessian in vector form:

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{r}_l \partial \mathbf{r}_i}=\left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T\left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{y}_l}\right)\left(\frac{\partial \mathbf{y}_l}{\partial \mathbf{r}_l}\right)}. \qquad (\text{C - I - 13})$$

C-II) The $\frac{\partial^2 J}{\partial \mathbf{z}^2}$ term

Recall the Hessian matrix of equation (C-1). Now, we would like to calculate the lower right corner term: $\frac{\partial^2 J}{\partial \mathbf{z}^2}$. Recall equation (B-II-1), which gives

$$\frac{\partial J}{\partial \mathbf{z}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \frac{\partial J}{\partial \mathbf{u}}. \qquad (\text{C - II - 1})$$

We can express each element of $\frac{\partial J}{\partial \mathbf{z}}$ as

$$\frac{\partial J}{\partial z_k} = \sum_j \frac{\partial u_j}{\partial z_k}\frac{\partial J}{\partial u_j}. \qquad (\text{C - II - 2})$$

By taking the derivative with respect to gating network's weights, $\mathbf{z}_k$, using the chain rule, we obtain

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial z_k}\right] = \sum_j \frac{\partial}{\partial z_m}\left[\frac{\partial u_j}{\partial z_k}\right]\frac{\partial J}{\partial u_j} + \sum_j \frac{\partial u_j}{\partial z_k}\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial u_j}\right]. \qquad (\text{C - II - 3})$$

We can expand the second term using the total derivative as

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial u_j}\right] = \sum_n \frac{\partial}{\partial u_n}\left[\frac{\partial J}{\partial u_j}\right]\frac{\partial u_n}{\partial z_m}. \qquad (\text{C - II - 4})$$

Then, equation (C-II-3) becomes

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial z_k}\right] = \sum_j \frac{\partial}{\partial z_m}\left[\frac{\partial u_j}{\partial z_k}\right]\frac{\partial J}{\partial u_j} + \sum_j \frac{\partial u_j}{\partial z_k}\sum_n \frac{\partial}{\partial u_n}\left[\frac{\partial J}{\partial u_j}\right]\frac{\partial u_n}{\partial z_m}. \qquad (\text{C - II - 5})$$

The first term in the summation can be rewritten

$$\sum_j \frac{\partial}{\partial z_m}\left[\frac{\partial u_j}{\partial z_k}\right]\frac{\partial J}{\partial u_j} = \sum_j \frac{\partial^2 u_j}{\partial z_m \partial z_k}\frac{\partial J}{\partial u_j} = \mathbf{B}(\mathbf{x}) \qquad \text{(C - II - 6)}$$

where we assume $\mathbf{B}(\mathbf{x})$ is small, then we can approximate equation (C - II - 5) as

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial z_k}\right] = \sum_j \frac{\partial u_j}{\partial z_k}\sum_n \frac{\partial}{\partial u_n}\left[\frac{\partial J}{\partial u_j}\right]\frac{\partial u_n}{\partial z_m}. \qquad \text{(C - II - 7)}$$

Notice the similarity between this computation and the $\frac{\partial^2 J}{\partial \mathbf{r}_l \partial \mathbf{r}_i}$ term computation of equation (C - I - 12). In fact, we can also express equation (C-II-7) in vector form as :

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{z}^2} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{u}^2}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)}. \qquad \text{(C - II - 8)}$$

C-III) The $\frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_i}$ term

This term resembles the upper right hand section of the Hessian matrix. To calculate this term, we use equation (B-I-3), which gives

$$\frac{\partial J}{\partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \frac{\partial J}{\partial \mathbf{y}_i}. \qquad \text{(C - III - 1)}$$

Again, we can express the elements in summation form as:

$$\frac{\partial J}{\partial r_{i_k}} = \sum_{j=1}^{Q} \frac{\partial y_{i_j}}{\partial r_{i_k}} \frac{\partial J}{\partial y_{i_j}}. \qquad \text{(C - III - 2)}$$

By taking the derivative with respect to gating network's weights, we will have the second derivative of the performance index with respect to the cross term weights and biases between the i^{th} expert network and the gating network:

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j\left(\frac{\partial}{\partial z_m}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]\frac{\partial J}{\partial y_{j_i}}+\frac{\partial y_{j_i}}{\partial r_{k_i}}\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial y_{j_i}}\right]\right). \qquad \text{(C - III - 3)}$$

Expanding the second term on the summation, yields

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial y_{j_i}}\right]=\sum_q\frac{\partial}{\partial u_q}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial u_q}{\partial z_m}. \qquad \text{(C - III - 4)}$$

Substituting equation (C-III-4) back into equation (C-III-3), we obtain

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j\left(\frac{\partial}{\partial z_m}\left[\frac{\partial y_{j_i}}{\partial r_{k_i}}\right]\frac{\partial J}{\partial y_{j_i}}+\frac{\partial y_{j_i}}{\partial r_{k_i}}\sum_q\frac{\partial}{\partial u_q}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial u_q}{\partial z_m}\right). \qquad \text{(C - III - 5)}$$

Notice that the first term of the summation goes to zero because the outputs of the i^{th} expert network and the weights of the gating network are unrelated. Therefore, equation (C-III-5) reduces to

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j\frac{\partial y_{j_i}}{\partial r_{k_i}}\sum_q\frac{\partial}{\partial u_q}\left[\frac{\partial J}{\partial y_{j_i}}\right]\frac{\partial u_q}{\partial z_m} \qquad \text{(C - III - 6)}$$

or

$$\frac{\partial}{\partial z_m}\left[\frac{\partial J}{\partial r_{k_i}}\right]=\sum_j\sum_q\frac{\partial y_{j_i}}{\partial r_{k_i}}\frac{\partial^2 J}{\partial u_q\partial y_{j_i}}\frac{\partial u_q}{\partial z_m}. \qquad \text{(C - III - 7)}$$

We can also express the above equation in the following vector form:

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{z}\partial \mathbf{r}_i}=\left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T\left(\frac{\partial^2 J}{\partial \mathbf{y}_i\partial \mathbf{u}}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)}. \qquad (\text{C - III - 8})$$

The $\frac{\partial^2 J}{\partial \mathbf{r}_k\partial \mathbf{z}}$ term

Since $\frac{\partial^2 J}{\partial \mathbf{r}_k\partial \mathbf{z}}$ is the transpose of $\frac{\partial^2 J}{\partial \mathbf{z}\partial \mathbf{r}_i}$, we have

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{r}_k\partial \mathbf{z}}=\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T\left(\frac{\partial^2 J}{\partial \mathbf{u}\partial \mathbf{y}_k}\right)\left(\frac{\partial \mathbf{y}_k}{\partial \mathbf{r}_k}\right)}. \qquad (\text{C - IV - 1})$$

C-V) Compute the $\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right)$ term

Recall equation (A-I-3), which gives

$$\frac{\partial J}{\partial \mathbf{y}_i}=h_i\left(\mathbf{y}^*-\mathbf{y}_i\right). \qquad (\text{C - V - 1})$$

By using the chain rule, we take the derivative of this equation with respect to the k^{th} expert network's output, which gives

$$\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right)=h_i\frac{\partial}{\partial \mathbf{y}_i}\left(\mathbf{y}^*-\mathbf{y}_i\right)+\left(\mathbf{y}^*-\mathbf{y}_i\right)\left(\frac{\partial h_i}{\partial \mathbf{y}_i}\right)^T. \qquad (\text{C - V - 2})$$

If $i=k$, then

$$\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right)=h_i\frac{\partial}{\partial \mathbf{y}_k}\left(\mathbf{y}^*-\mathbf{y}_i\right)+\left(\mathbf{y}^*-\mathbf{y}_i\right)\left(\frac{\partial h_i}{\partial \mathbf{y}_k}\right)^T. \qquad (\text{C - V - 3})$$

From equation (A-III-6a), we have

$$\frac{\partial h_i}{\partial \mathbf{y}_k} = \left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right) \qquad \text{if } i = k . \tag{C - V - 4}$$

Substitute equation (C-V-4) into (C-V-3) yields

$$\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) = h_i(-\mathbf{I}) + \left(\mathbf{y}^* - \mathbf{y}_i\right)\left[\underbrace{\left(h_i - h_i^2\right)}_{\text{scalar}}\left(\mathbf{y}^* - \mathbf{y}_i\right)\right]^T . \tag{C - V - 5}$$

and we have

$$\boxed{\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) = h_i(-\mathbf{I}) + \left(\mathbf{y}^* - \mathbf{y}_i\right)\left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right)^T \quad \text{if } i = k}. \tag{C - V - 6}$$

In equation (C-V-2), if $i \neq k$, then from equation (A-III-6b), we have

$$\frac{\partial h_i}{\partial \mathbf{y}_k} = -h_i h_k\left(\mathbf{y}^* - \mathbf{y}_k\right) \qquad \text{if } i \neq k . \tag{C - V - 7}$$

Substitute equation (C-V-7) into (C-V-6), yields,

$$\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) = h_i .0 + \left(\mathbf{y}^* - \mathbf{y}_i\right)\left[\underbrace{\left(-h_i h_k\right)}_{\text{scalar}}\left(\mathbf{y}^* - \mathbf{y}_k\right)\right]^T \tag{C - V - 8}$$

and we have

$$\boxed{\frac{\partial}{\partial \mathbf{y}_k}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) = -\left(\mathbf{y}^* - \mathbf{y}_i\right)\left(h_i h_k\right)\left(\mathbf{y}^* - \mathbf{y}_k\right)^T \quad \text{if } i \neq k} \tag{C - V - 9}$$

C-VI) Compute the $\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial u_i}\right)$ term

From equation (A-II-5), we have

$$\frac{\partial J}{\partial u_i} = h_i - g_i . \qquad (\text{C - VI - 1})$$

By taking the derivative of this equation with respect to the output of the gating network, we have:

$$\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial u_i}\right) = \frac{\partial}{\partial u_j}\left(h_i - g_i\right) = \frac{\partial h_i}{\partial u_j} - \frac{\partial g_i}{\partial u_j} . \qquad (\text{C - VI - 2})$$

Notice that we obtain two terms in equation (C-VI-2), and both terms have been computed in Appendix A-V and Appendix A-IV. Hence, substituting equation (A-V-9) and (A-IV-3) into equation (C-VI-2) yields:

$$\boxed{\begin{aligned} \frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial u_i}\right) &= \left(h_i - h_i^2\right) - \left(g_i - g_i^2\right) && \text{if } \; i = j \\ &= h_i h_j + g_i g_j && \text{if } \; i \neq j \end{aligned}} . \qquad (\text{C - VI - 3})$$

C-VII) Compute the $\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right)$ term

Recall equation (A-II-5), which yields

$$\frac{\partial J}{\partial u_i} = h_i - g_i . \qquad (\text{C - VII - 1})$$

Differentiating it with respect to the j^{th} expert network output gives

$$\begin{aligned}\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right) &= \frac{\partial}{\partial \mathbf{y}_j}(h_i - g_i) \\ &= \frac{\partial h_i}{\partial \mathbf{y}_j} - \frac{\partial g_i}{\partial \mathbf{y}_j}\end{aligned} . \qquad \text{(C - VII - 2)}$$

Since g_i and $\mathbf{y}_j$ are not related,

$$\frac{\partial g_i}{\partial \mathbf{y}_j} = 0 . \qquad \text{(C - VII - 3)}$$

From Appendix A-III, $\frac{\partial h_i}{\partial \mathbf{y}_j}$ is given as:

$$\begin{aligned}\frac{\partial h_i}{\partial \mathbf{y}_j} &= \left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right) && \text{if } i = j \\ &= -h_i h_j\left(\mathbf{y}^* - \mathbf{y}_j\right) && \text{if } i \neq j\end{aligned} . \qquad \text{(C - VII - 4)}$$

Substituting equation (C-VII-3) and (C-VII-4) into equation (C-VII-2) gives

$$\boxed{\begin{aligned}\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right) &= \left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right) && \text{if } i = j \\ &= -h_i h_j\left(\mathbf{y}^* - \mathbf{y}_j\right) && \text{if } i \neq j\end{aligned}} . \qquad \text{(C - VII - 5)}$$

C-VIII) Compute the $\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right)$ term

Since $\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right)$ is the transpose of $\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right)$,

$$\frac{\partial}{\partial \mathbf{u}}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) = \left[\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial \mathbf{u}}\right)\right]^T , \qquad \text{(C - VIII - 1)}$$

we have

$$\boxed{\begin{aligned}\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) &= \left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right)^{\mathrm{T}} \quad \text{if } \ i = j \\ &= -h_i h_j\left(\mathbf{y}^* - \mathbf{y}_i\right)^{\mathrm{T}} \qquad \text{if } \ i \neq j\end{aligned}} \qquad (\mathrm{C - VIII - 2})$$

C-IX) Summary

The approximated Hessian matrices are calculated using the following equations:

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{r}_j \partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{y}_j}\right)\left(\frac{\partial \mathbf{y}_j}{\partial \mathbf{r}_j}\right)} \qquad (\mathrm{C - IX - 1})$$

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{z}^2} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{u}^2}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)} \qquad (\mathrm{C - IX - 2})$$

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{z} \partial \mathbf{r}_i} = \left(\frac{\partial \mathbf{y}_i}{\partial \mathbf{r}_i}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{y}_i \partial \mathbf{u}}\right)\left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)} \qquad (\mathrm{C - IX - 3})$$

$$\boxed{\frac{\partial^2 J}{\partial \mathbf{r}_j \partial \mathbf{z}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{z}}\right)^T \left(\frac{\partial^2 J}{\partial \mathbf{u} \partial \mathbf{y}_j}\right)\left(\frac{\partial \mathbf{y}_j}{\partial \mathbf{r}_j}\right)}. \qquad (\mathrm{C - IX - 4})$$

Each middle term in the above equations are calculated as

$$\boxed{\begin{aligned}\frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) &= h_i(-\mathbf{I}) + \left(\mathbf{y}^* - \mathbf{y}_i\right)\left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right)^T \quad \text{if } \ i = j \\ &= -\left(\mathbf{y}^* - \mathbf{y}_i\right)\left(h_i h_j\right)\left(\mathbf{y}^* - \mathbf{y}_j\right)^T \qquad \text{if } \ i \neq j\end{aligned}} \qquad (\mathrm{C - IX - 5})$$

$$\boxed{\begin{aligned}\frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial u_i}\right) &= \left(h_i - h_i^2\right) - \left(g_i - g_i^2\right) \qquad \text{if } \ i = j \\ &= h_i h_j + g_i g_j \qquad \text{if } \ i \neq j\end{aligned}} \qquad (\mathrm{C - IX - 6})$$

$$\boxed{\begin{aligned} \frac{\partial}{\partial \mathbf{y}_j}\left(\frac{\partial J}{\partial u_i}\right) &= \left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right) && \text{if } i = j \\ &= -h_i h_j\left(\mathbf{y}^* - \mathbf{y}_j\right) && \text{if } i \neq j \end{aligned}} \qquad (\mathrm{C-IX-7})$$

$$\boxed{\begin{aligned} \frac{\partial}{\partial u_j}\left(\frac{\partial J}{\partial \mathbf{y}_i}\right) &= \left(h_i - h_i^2\right)\left(\mathbf{y}^* - \mathbf{y}_i\right)^{\mathrm{T}} && \text{if } i = j \\ &= -h_i h_j\left(\mathbf{y}^* - \mathbf{y}_i\right)^{\mathrm{T}} && \text{if } i \neq j \end{aligned}} \qquad (\mathrm{C-IX-8})$$

VITA

Meng Hock Fun

Candidate for the Degree of

Master of Science

Thesis: TRAINING MODULAR NEURAL NETWORKS WITH MARQUARDT-LEVENVERG ALGORITHM

Major Field: Electrical Engineering

Biographical:

Education: Graduated from Taylor's College, Kuala Lumpur, Malaysia in 1989; received Bachelor of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma in July 1993. Completed the Requirements for the Master of Science degree in Electrical Engineering at Oklahoma State University in May, 1996.

Experience: Employed by Oklahoma State University, Department of Electrical Engineering as a teaching assistant and as a graduate research assistant in a Friction Control Project from 1993 to present.

Professional Memberships: Tau Beta Phi Society, Eta Kappa Nu Society, and United States Chess Federation.