國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

碩 士 論 文

三層類神經網路的改良式動態最佳學習

Revised Dynamic Optimal Training of Three-Layer Neural Network

研究生：林書帆

指導教授：王啟旭　教授

中華民國九十五年十月

三層類神經網路的改良式動態最佳學習

Revised Dynamic Optimal Training of Three-Layer Neural Network

研究生： 林書帆　　Student: Shu-Fan Lin

指導教授：王啟旭　教授　　Advisor: Chi-Hsu Wang

國 立 交 通 大 學

電機與控制工程學系

碩士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

October 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十月

三層類神經網路的改良式動態最佳學習

研究生：林書帆　　　　　　　　　指導教授：王啟旭 教授

國立交通大學電機與控制工程研究所

摘要

本篇論文是針對三層類神經網路提出一個改良式動態最佳訓練法則，其中類神經網路的隱藏層經過一個 S 型激發函數，輸出層經過一個線性的激發函數。這種三層的網路可以被運用於處理分類的問題，像是蝴蝶花的品種分類。我們將對這種三層神經網路的動態最佳訓練方法提出一個完整的証明，用來說明這種動態最佳訓練方法保證神經網路能在最短的迭代次數下達到收斂的輸出結果。這種改良式動態最佳訓練方法，是在每一次的迭代過程中不斷的尋找，來取得下一次迭代過程所需要的最佳學習速率以及穩定學習速率的上限值，以保證最佳的收斂的訓練結果。經由調整初始加權值矩陣，改變激發函數，改良式動態最佳學習法則比原先的動態最佳學習法則更加省時以及更加穩定。我們可以由 XOR 和蝴蝶花的測試例子得到良好的結論。

Revised Dynamic Optimal Training of Three-Layer Neural Network

Student: Shu-Fan Lin **Advisor: Chi-Hsu Wang**

Department of Electrical and Control Engineering
National Chiao Tung University

ABSTRACT

This thesis proposes a revised dynamic optimal training algorithm for a three layer neural network with sigmoid activation function in the hidden layer and linear activation function in the output layer. This three layer neural network can be used for classification problems, such as the classification of Iris data. Rigorous proof has been presented for the revised dynamical optimal training process for this three layer neural network, which guarantees the convergence of the training in a minimum number of epochs. This revised dynamic optimal training finds optimal learning rate with its upper-bound for next iteration to guarantee optimal convergence of training result. With modification of initial weighting factors and activation functions, revised dynamic optimal training algorithm is more stable and faster than dynamic optimal training algorithm. Excellent improvements of computing time and robustness have been obtained for XOR and Iris data set.

ACKNOWLEDGEMENT

I feel external gratitude to my advisor, Chi-Hsu Wang for teaching me many things. He taught me how to do the research and write a thesis. And the most important is that he taught me how to get along with people. And I am grateful to everyone in ECL. I am very happy to get along with you. Finally, I appreciate my parent’s support and concern; therefore I can finish my master degree smoothly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Artificial neural network (ANN) is the science of investigating and analyzing the algorithms of the human brain, and using similar algorithm to build up a powerful computational system to do the tasks like pattern recognition [1], [2], identification [3], [4] and control of dynamical systems [5], [6], system modeling [7], [8] and nonlinear prediction of time series [9]. The first model of artificial neural network was proposed for simulating human brain by *F. Rosenblatt* in 1958, 1962 [10], [11]. So the most attractive character of artificial neural network is that it can be taught to achieve the complex tasks we had experienced before by using some learning algorithms and training examples. Among most popular training algorithms of artificial neural network, *Error Back-Propagation Algorithm* [12], [13] is widely used for classification problems. The well-known error back-propagation algorithm or simply the back-propagation algorithm (BPA), for training multi-layer perceptrons was proposed by *Rumelhart et al.* in 1986 [14]. Although the popular back-propagation algorithm is easier to understand and to implement for most applications, it has the problems of converging to local minimum and a slow convergence rate. It is well known that performance of BPA is significantly affected by several parameters, i.e., learning rate, initial weighting factors and activation functions. In this thesis, the effect of each adjustable parameter will be analyzed, in order to find a new training algorithm with better performance compared with BPA.

There are many research works on the learning of multilayer NN with saturated nonlinear activation functions, e.g., sigmoid function. Some researches work on the learning process of multilayer NN with linear activation functions [15]. However, the authors in [16] state that if activation functions of each layer of multilayer NN are all sigmoid, the back-propagated error signal may be seriously discounted by the saturation region of sigmoid function. Hence, convergence rate of learning will slow down. Although, multilayer linear neural network does not have the problem of saturation, its learning capability is not as good as multilayer sigmoid neural network. To retain the advantages of both kinds of activation function, and to easy the bad effect of they, the architecture with combination of linear and nonlinear activation functions will be proposed in this investigation. We will also propose an adequate analysis and excellent experimental results to support that the learning of NN will have better performance with an unlimited linear activation in output layer and a sigmoid function in hidden layer.

It is well known that initial weighting factors have a significant impact on the performance of multilayer NN. In almost all applications of back-propagation, the initial weighting factors are selected randomly from a problem-dependent range. In the investigation of *Hosein et al.* in 1989 [17], high initial weighting factors can accelerate convergence rate of learning, but may suffer the problem of local minimum. Low initial weighting factors can be used over a wide range of applications for reliable learning, but convergence rate may slow down. In this thesis, we will propose a clear and definite way to find proper initial weighting factors which can reduce the probability of falling into local minimum on the error surface during the back propagation process.

Learning rate is always the most important adjustable parameter in BPA and other algorithms which are generalized from BPA. Learning rate controls the convergence of the algorithm to an optimal local solution, and it also dominates the convergence rate. Hence, the choice of learning rate is a critical decision for convergence of BPA.

For smaller learning rate, we may have a convergent result. But the speed of the output convergence is very slow and need more number of epochs to train the network. For larger learning rate, the speed of training can be accelerated, but it will cause the training result to fluctuate and even leads to divergent result. However, *Rumelhart et al.* did not show how to get a good value for the learning rate in [14]. Since, several researches have used empirical rule, heuristic methods, and adaptive algorithm to set a good value for leaning rate. The dynamical optimal learning rate was proposed in [18] for a simple two layer neural network (without hidden layers) without any activation functions in the output layer. To break the limited learning capability for the two-layer NN, authors in [19] presented the dynamical optimal learning rate for the three-layer NN with sigmoid activation functions in the hidden and output layers. The basic theme in [18] and [19] is to find a stable and optimal learning for the next iteration in BPA. *Cerqueira et al.* [20] developed a local convergence analysis made through Lyapunov's second method for the BPA and supplied an upper bound for the learning rate. *Cerqueira et al.* also present an adaptive algorithm with upper bound of learning rate in [21]. In this thesis we combine the basic theme in [18] and [19] with upper bound of learning rate presented in [20] and [21] to present a revised dynamic optimal learning rate with dynamic upper bound in this thesis.

The goal of this thesis is to propose a revised dynamic optimal training algorithm based on modifications made for three key parameters, i.e., learning rate, initial weighting factor and activation function. Rigorous proof will be proposed and the popular XOR [22] and Iris data [23], [24] classification benchmarks will be fully illustrated. Excellent results have been obtained by comparing our optimal training results with previous results using fixed small learning rates.

CHAPTER 2

Neural Network and Back-Propagation Algorithm

In this chapter, the multi-layer feed-forward perceptron (MLP) structure of artificial neural networks (ANN) and the training algorithm will be introduced. First the model of a neuron will be introduced. Then, the single layer perceptron will be explained and it will lead to the multi-layer perceptrons in later sections. The back-propagation algorithm for multi-layer feed forward perceptrons will be explained in Section 2.4. Finally, the dynamic optimal training algorithm will be introduced in Section 2.5.

2.1 Models of A Neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network. It is the basic unit in a neural network. It forms the basis for designing artificial neural network. The block diagram of the model of a neuron is shown in Figure 2-1. The neural model can be divided into three basic elements.

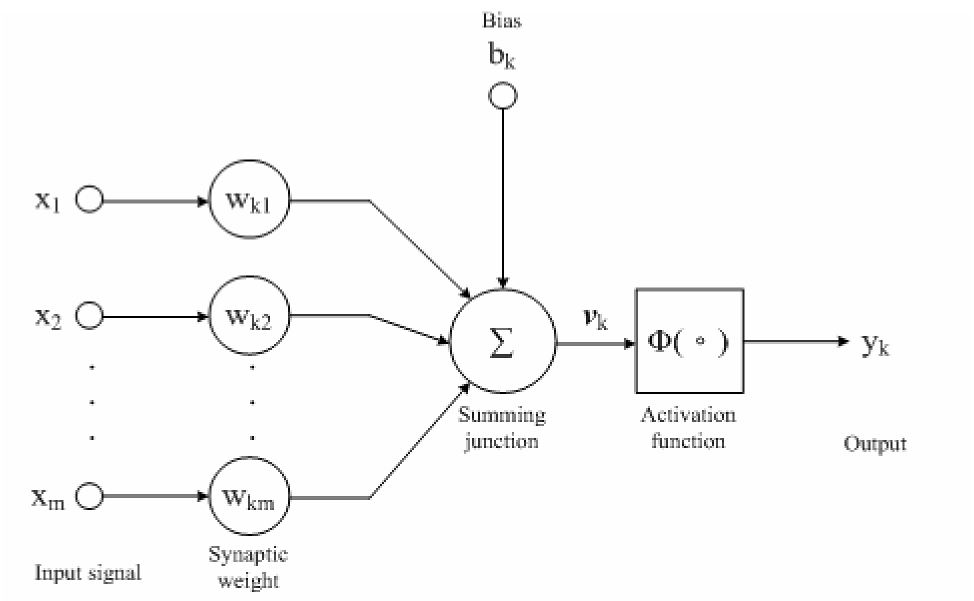


Figure 2-1. Model of A Neuron

1. A set of synaptic links, each of which is characterized by a weight factor. Symbol x_j denotes the input signal at the input of synapse j. The input signal x_j connected to neuron k is multiplied by the synaptic weight w_{kj}. Symbol w_{kj} denotes the weight factor from j^{th} neuron to k^{th} neuron.
2. An adder for summing the input signals which are weighted by the respective synapses of the neuron. These operations constitute a linear combiner.
3. An activation function for limiting the amplitude of the output of a neuron. The activation function squashes or limits the amplitude range of the output signal to some finite value. It can also set a saturation region for the output signal.

The neuron model of Figure 2-1 also includes an externally applied bias, denoted by b_k. The bias b_k has the effect of increasing or decreasing the net input of the activation function, depending on whether it is positive or negative, respectively. In mathematical terms, we can describe the model of a neuron k by using the following pair of equations:

$$v_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

and

$$y_k = \phi\left(v_k + b_k\right) \tag{2.2}$$

where $x_1, x_2, \ldots., x_m$ are the input signals; $w_{k1}, w_{k2}, \ldots., w_{km}$ are the synaptic weights of neuron k; V_k is the linear combiner output; b_k is the bias; $\phi(\cdot)$ is the activation function; and y_k is the output signal of the neuron. The bias b_k is an external parameter of artificial neuron k. It can be expressed in another form. In Figure 2-1, we add a new synapse, it input is $x_0 = 1$ and its weight is $w_{k0} = b_k$. We may therefore redefine a new model of neuron k as in Figure 2-2.

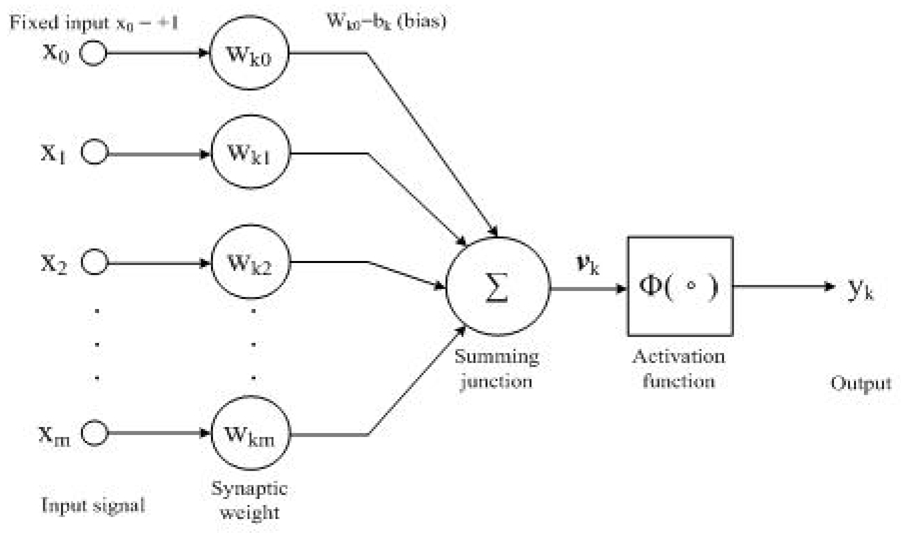


Figure 2-2. Another Model of A Neuron

Activation function $\phi(\square)$ defines the output of a neuron $\boldsymbol{k}$ in terms of the induced local field $\boldsymbol{V_k}$. There are three basic types of activation functions.

1. *Threshold Function. (Figure 2-3(a))*

$$\phi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ 0 & if \quad v < 0 \end{cases} \tag{2.3}$$

In this type, the output of a neuron has the value 1 if the induced local field of the neuron is nonnegative and 0 otherwise. The output of neuron $\boldsymbol{k}$ applies into such a threshold function can be expressed as

$$y_k = \begin{cases} 1 & if \quad v_k \geq 0 \\ 0 & if \quad v_k < 0 \end{cases} \tag{2.4}$$

2. *Piecewise-Linear Function. (Figure 2-3(b))*

$$\phi(v) = \begin{cases} 1 & v \geq +\frac{1}{2} \\ v + 0.5 & +\frac{1}{2} > v > -\frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases} \tag{2.5}$$

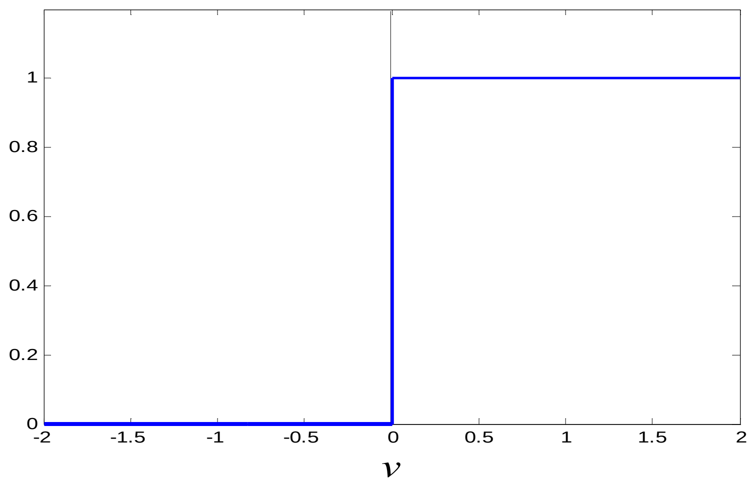
This form of an activation function can be viewed as an approximation to a nonlinear amplifier. A linear function can be viewed as the special form of piecewise-linear function when piecewise-linear function maintains in the linear region of operation without running into saturation.

1. *Sigmoid Function. (Figure 2-3(c))*

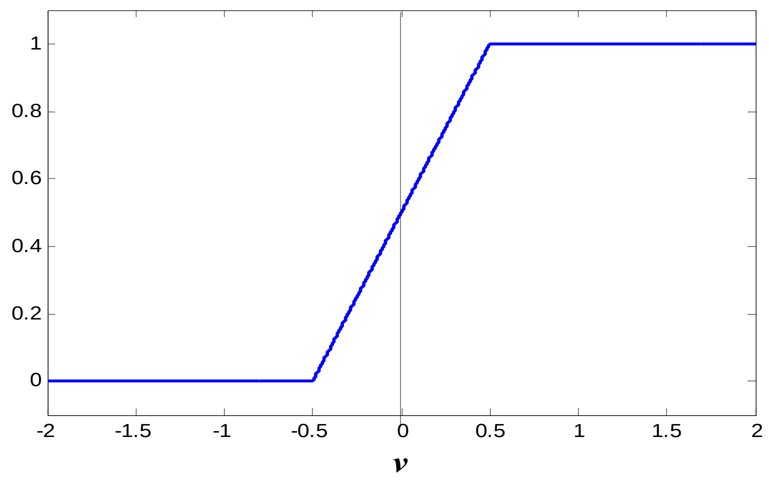
Sigmoid function is the most common form of activation function used in construction of artificial neural network. The graph of sigmoid function is s-shaped. It is defined as a strictly increasing function and it exhibits a graceful balance between linear and nonlinear behavior. A common form of sigmoid function is the *logistics function* [25] which is defined by

$$\phi(v) = \frac{1}{1+\exp(-av)} \tag{2.6}$$

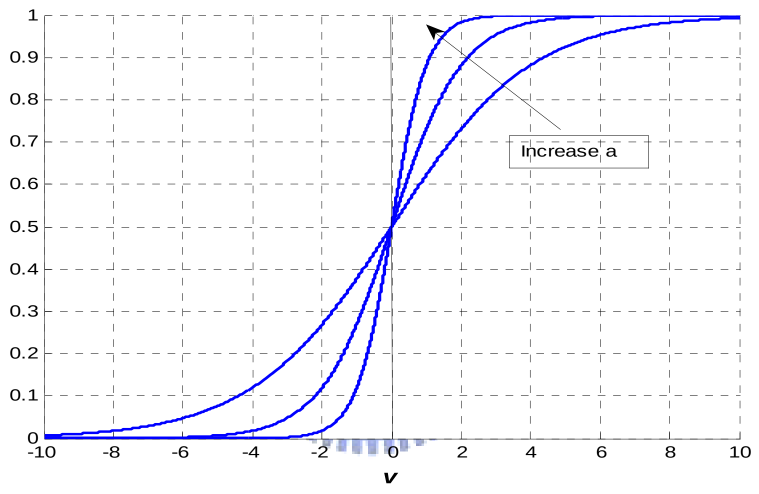
where a is the slope parameter of the sigmoid function. By varying the slope parameter a, we have different types of sigmoid function. It is noted that the slope at original point equals $a/4$. A sigmoid function gives a continuous range of values from 0 to 1. Sigmoid function is differentiable over the all range of values, whereas threshold function and piecewise-linear function are not. It is an important fact of neural network theory discussed in the later chapter.



(a)



(b)



(c)

Figure 2-3. (a) Threshold function (b) Piecewise-linear function

(c) Sigmoid function for varying slope parameter *a*

2.2 Single Layer Perceptron

The first model of the feed-forward network, ***perceptron***, was proposed by *F. Rosenblatt* in 1958 [10]. He hoped to find a suitable model to simulate the animal's brain and the visual system so that he proposed the "perceptron" model, which is a *supervised learning* model. The supervised learning is also referred to as the learning with a teacher, and the teacher here means the input-output data sets for training. It also means that the perceptron can be trained by the given input-output data. The perceptron is the simplest form of a neural network used for classification of patterns which are said to be *linear separable*. Basically, the structure of the perceptron consists of a neuron with adjustable synaptic weights, bias, and desired output. The structure of the perceptron is depicted in Figure 2-4. The limiter is using a threshold function; its graph is shown in Figure 2-3(a).

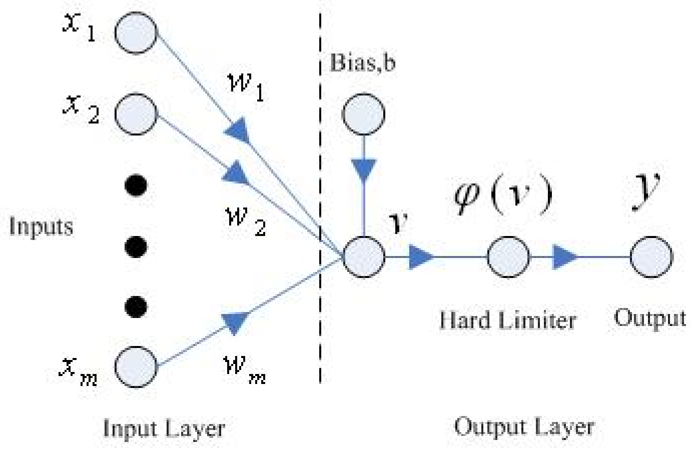


Figure 2-4. Single layer perceptron

In Figure 2-4, the input data set of the perceptron is denoted by ***{x1, x2,…, xm}*** and the corresponding synaptic weights of the perceptron are denoted by ***{w1, w2,…, wm}***. The external bias is denoted by ***b***. The first part of the percpetron computes the linear combination of the products of input data and synaptic weight with an externally applied bias. So the result of the first part of the perceptron, ***v***, can be expressed as

$$v = b + \sum_{i=1}^{m} w_i x_i \tag{2.7}$$

Then in the second part, the resulted sum *v* is applied to a hard limiter. Therefore, the output of the perceptron equals +1 or 0. The output of perceptron equals to +1 if the resulted sum *v* is positive or zero; and the output of perceptron equals to 0 if *v* is negative. This can be simply expressed by (2.8).

$$y = \begin{cases} +1 \text{ , if } & b + \sum_{i=1}^{m} x_i w_i \geq 0 \\ 0 \text{ , if } & b + \sum_{i=1}^{m} x_i w_i < 0 \end{cases} \tag{2.8}$$

The goal of the perceptron is to classify the input data represents by the set $\{x_1, x_2, \ldots, x_m\}$ into one of two classes, C_1 and C_2. If the output of the perceptron equals to +1, the input data represented by the set $\{x_1, x_2, \ldots, x_m\}$ will be assigned to class C_1. Otherwise, the input data will be assigned to class C_2. The single layer perceptron with only a single neuron is limited to performing pattern classification with only two classes. By expending the output layer of the perceptron to include more than one neuron, we can perform classification with more than two classes. However, these classes must be linearly separable for the perceptron to work properly. If we need to deal with classes which are not linearly separable, we need multi-layer feed-forward perceptron to handle such classes.

2.3 Multi-layer Feed-forward Perceptron

In this section we will introduce the multi-layer feed-forward network, an important class of neural network. The difference between single layer perceptron and multi-layer feed-forward perceptron is the "*hidden layer*". The multi-layer network consists of a set of input nodes (input layer), one or more hidden layers, and a set of output nodes that constitute the output layer. The input signals will propagate through the network in the forward direction. A multi-layer feed-forward fully-connected network is shown in Figure 2-5 with only one hidden layer.

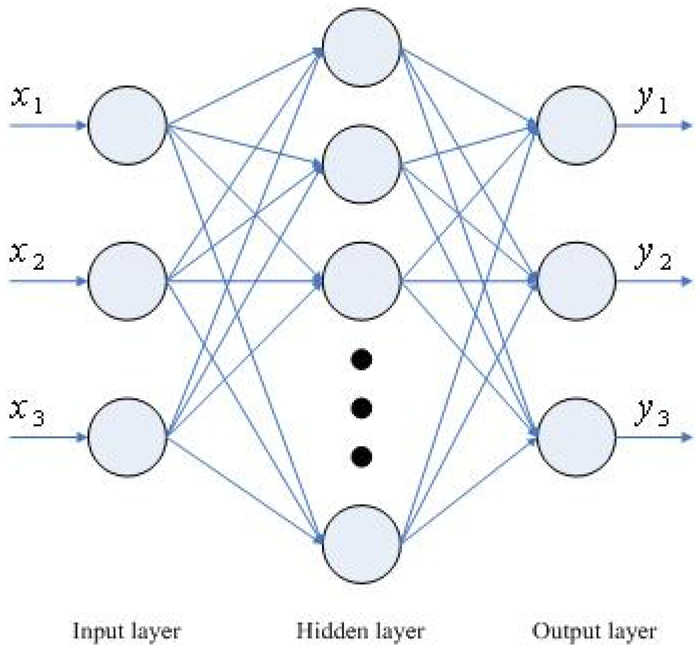


Figure 2-5. A three layer feed-forward network

A multi-layer perceptron has three major characteristics:

1. The model of each neuron in the network includes a nonlinear activation function. Note that the activation function used in the multilayer perceptron is smooth, means the activation function is differentiable everywhere. It is different from the threshold function (Figure 2-3(a)) used in the model of single-layer perceptron. A commonly used form of nonlinear activation function is a sigmoid function.

$$y = \frac{1}{1 + \exp(-ax)}$$

   where ***x*** is the induced local field, the weighted sum of all synaptic inputs plus the bias. ***y*** is the output of the neuron. And a is the slope parameter of the sigmoid function. Usually, we set the value of a to be 1.

2. The neural network contains one or more layers of hidden neurons between input layer and output layer. The hidden neurons enable the multi-layer networks to deal with more complex problems which can not be solved by single layer perceptron. The number of hidden layers for different problems is also an open question. From

previous researches, one hidden layer is enough to handle many problems properly. Two or more hidden layers can do more complex tasks, but take a huge computation time.

3. The network exhibits a high degree of connectivity. Any change in the connectivity of the network requires a change in the population of synaptic weights. This character makes network more like the human brain.

2.4 The Back-Propagation Algorithm (BPA)

The most serious problem of the single-layer perceptron is that we don't have any proper learning algorithm to adjust the synaptic weights of the perceptron. But the multi-layer feed-forward perceptron doesn't have this problem. The synaptic weights of the multi-layer perceptrons can be trained by using the highly popular algorithm known as the *error back-propagation algorithm (EBP algorithm)* [14]. We can view the error back-propagation algorithm as the general form of the *least-mean square algorithm* (LMS) [26]. The error back-propagation algorithm (or the back-propagation algorithm) consists of two parts, one is the forward pass and the other is the backward pass. In the forward pass, the effect of the input data propagates through the network layer by layer. During this process, the synaptic weights of the networks are all *fixed*. On the other hand, the synaptic weights of the multi-layer perceptron are all *adjusted* in accordance with the error-correction rule during the backward pass. Before using error-correction rule, we have to define the *error signal* of the learning process. Because the goal of the learning algorithm is to find one set of weights which makes the actual outputs of the network equal to the desired outputs, so we can define the error signal as the difference between the actual output and the desired output. Specifically, the desired response is subtracted from the actual response of the network to produce the error signal. In the backward pass, the error signal propagates backward through the network from output layer. During the backward pass, the synaptic weights of the multi-layer perceptron are adjusted to make the actual outputs of the network move closer to the desired outputs. This is why we call this algorithm as "error back-propagation algorithm".

Back-propagation algorithm:

First, we define the error signal at the output of the neuron j at iteration t

$$e_j(t) = y_j(t) - d_j(t) \tag{2.9}$$

where $\boldsymbol{d_j}$ is the desired output of output node j and $\boldsymbol{y_j}$ is the actual output of output node j. We define the square error of output node j.

$$\zeta_j = \frac{1}{2} e_j^2(t) \tag{2.10}$$

By the same way, we can define the total square error $\boldsymbol{J}$ of the network as:

$$J = \sum_j \zeta_j = \frac{1}{2} \sum_j e_j^2 \tag{2.11}$$

The goal of the back-propagation algorithm is to find one set of weights so that the actual outputs can be as close as possible to the desired outputs. In other words, the purpose of the back-propagation algorithm is to reduce the total square error J, as described in (2.11). In the method of steepest descent, the successive adjustments applied to the weight matrix $\boldsymbol{W}$ are in the direction opposite to the gradient matrix $\partial J / \partial W$. The adjustment of weight matrix $\boldsymbol{W}$ can be expressed as:

$$W_{(t+1)} = W_{(t)} - \Delta W_{(t)} = W_{(t)} - \eta \left. \frac{\partial J}{\partial W} \right|_t$$

$$\Delta W(t) = -\eta \left. \frac{\partial J}{\partial W} \right|_t \tag{2.12}$$

where $\boldsymbol{\eta}$ is the *learning rate parameter* of the back-propagation algorithm. It decides the step-size of correction of the method of steepest descent. Note that the learning rate is a positive constant. Fig. 2-6 shows the signal-flow graph of multi-layer perceptron with the ith neuron in the $(k\text{-}1)$th layer and jth neuron in the kth layer. The activation functions in each layer are assumed to be the same sigmoid function in Fig. 2-6. By using the chain rule, the element of matrix $\partial J/\partial W$, i.e., $\partial J/\partial W_k^{(j,i)}$, can be represented as:

$$\frac{\partial J}{\partial W_k^{(j,i)}} = \frac{\partial J}{\partial V_k^{(j,:)}} \frac{\partial V_k^{(j,:)}}{\partial S_k^{(j,:)}} \frac{\partial S_k^{(j,:)}}{\partial W_k^{(j,i)}} \tag{2.13}$$

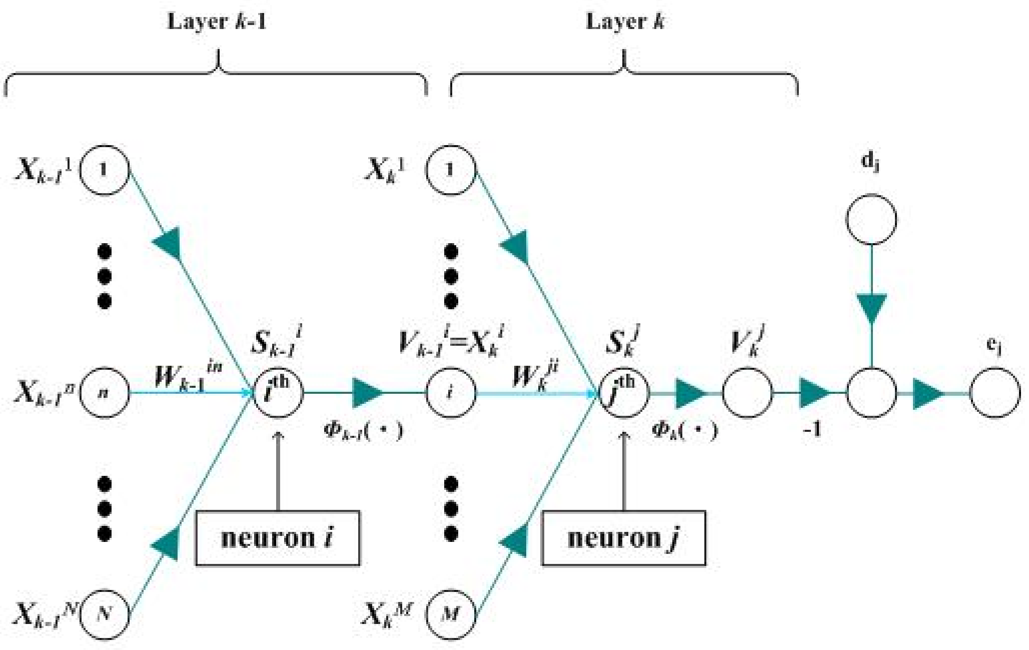


Figure 2-6 The signal-flow graph of multi-layer perceptron

where $W_k^{(j,i)}$ is the weight value between the ith neuron in the (k-1)th layer and jth neuron in the kth layer, as shown in Figure 2-6. $\varphi_k\left(\square\right)$ denotes the activation function used in the kth layer. $V_k^{(j,:)}$ denotes the activation function output of jth neuron in the kth layer. If kth layer is output layer, $V_k^{(j,:)}$ is equal to y_j. $S_k^{(j,:)}$ denotes the linear combination output of jth neuron in the kth layer. It can be shown as:

$$S_k^{(j,:)} = \sum_i \left(W_k^{(j,i)} X_k^{(i,:)} \right) \tag{2.14}$$

where X_k^i is the input of neuron in the kth layer. The superscript i means this input comes from ith neuron in the (k-1)th layer. Note that X_k^i is also the activation function output of ith neuron in the (k-1)th layer. So it has another form as:

$$X_k^{(i,:)} = V_{k-1}^{(i,:)} \tag{2.15}$$

$S_k^{(j,:)}$can also be viewed as the input of activation function of jth neuron in the kth layer. We use sigmoid function as activation function, so the relation between $V_k^{(j,:)}$ and $S_k^{(j,:)}$ can be expressed as:

$$V_k^{(j,:)} = \varphi_k\left(S_k^{(j,:)}\right) = \frac{1}{1+\exp(-S_k^{(j,:)})} \tag{2.16}$$

The following computation can be discussed in two different situations:

1. *If k*

1. *If k^{th} layer is an output layer.*

In this section, we change symbol k to symbol y for representing the output-layer.

$$W_{y(t+1)}^{(j,i)} = W_{y(t)}^{(j,i)} - \eta \frac{\partial J}{\partial W_{y(t)}^{(j,i)}} \tag{2.17}$$

We can substitute (2.13) into (2.17) to have

$$W_{y(t+1)}^{(j,i)} = W_{y(t)}^{(j,i)} - \eta \frac{\partial J}{\partial V_y^{(j,:)}} \frac{\partial V_y^{(j,:)}}{\partial S_y^{(j,:)}} \frac{\partial S_y^{(j,:)}}{\partial W_Y^{(j,i)}} \tag{2.18}$$

where

$$\frac{\partial J}{\partial V_y^{(j,:)}} = \frac{\partial}{\partial V_y^{(j,:)}}\left(\frac{1}{2}\sum_j \left(V_y^{(j,:)} - D^{(j,:)}\right)^2\right) = \left(V_y^{(j,:)} - D^{(j,:)}\right) \tag{2.19}$$

$$\frac{\partial V_y^{(j,:)}}{\partial S_y^{(j,:)}} = \frac{\partial}{\partial S_y^{(j,:)}}\left(\frac{1}{1+\exp(-S_y^{(j,:)})}\right) = \frac{\exp(-S_y^{(j,:)})}{\left(1+\exp(-S_y^{(j,:)})\right)^2} = V_y^{(j,:)}\left(1-V_y^{(j,:)}\right) \tag{2.20}$$

$$\frac{\partial S_y^{(j,:)}}{\partial W_y^{(j,i)}} = \frac{\partial}{\partial W_y^{(j,i)}}\left(\sum_i W_y^{(j,i)} X_y^{(i,:)}\right) = X_y^{(i,:)} = V_{y-1}^{(i,:)} \tag{2.21}$$

Substitute (2.19), (2.20), and (2.21) into (2.18), (2.18) can be rewritten as:

$$W_{y(t+1)}^{(j,i)} = W_{y(t)}^{(j,i)} - \eta(V_y^{(j,:)} - D^{(j,:)})V_y^{(j,:)}(1-V_y^{(j,:)})V_{y-1}^{(i,:)} \tag{2.22}$$

2. *If k^{th} layer is a hidden layer*

In this section, we change symbol k to symbol h for representing hidden-layer.

$$W_{h(t+1)}^{(j,i)} = W_{h(t)}^{(j,i)} - \eta \frac{\partial J}{\partial W_{h(t)}^{(j,i)}} \tag{2.23}$$

We can substitute (2.13) into (2.23) to have

$$W_{h(t+1)}^{(j,i)} = W_{h(t)}^{(j,i)} - \eta \frac{\partial J}{\partial V_h^{(j,:)}} \frac{\partial V_h^{(j,:)}}{\partial S_h^{(j,:)}} \frac{\partial S_h^{(j,:)}}{\partial W_h^{(j,i)}} \tag{2.24}$$

where

$$\frac{\partial J}{\partial V_h^{(j,:)}} = \sum_l \frac{\partial J}{\partial S_{h+1}^{(l,:)}} \frac{\partial S_{h+1}^{(l,:)}}{\partial V_h^{(j,:)}} \tag{2.25}$$

and

$$\frac{\partial S_{h+1}^{(l,:)}}{\partial V_h^{(j,:)}} = \frac{\partial}{\partial V_h^{(j,:)}}\left(\sum_j W_{h+1}^{(l,j)} X_{h+1}^{(j,:)}\right) = \frac{\partial}{\partial V_h^{(j,:)}}\left(\sum_j W_{h+1}^{(l,j)} V_h^{(j,:)}\right) = W_{h+1}^{(l,j)} \tag{2.26}$$

where $W_{h+1}^{(l,j)}$ is the synaptic weight between the lth neuron in the $(h+1)$th layer and the jth neuron in the kth layer. For simplicity, we assume the item $\partial J / \partial S_{h+1}^{(l,:)}$ in (2.25) to have the following form:

$$\frac{\partial J}{\partial S_{h+1}^{(l,:)}} = \delta_{h+1}^l \tag{2.27}$$

Substitute (2.26) and (2.27) into (2.25), we have

$$\frac{\partial J}{\partial V_k^{(j,:)}} = \sum_l \left(\delta_{h+1}^l W_{h+1}^{(l,j)}\right) \tag{2.28}$$

Then, $\partial V_h^{(j,:)} / \partial S_h^{(j,:)}$ and $\partial S_h^{(j,:)} / \partial W_h^{(j,i)}$ can be represented as

$$\frac{\partial V_h^{(j,:)}}{\partial S_h^{(j,:)}} = \frac{\partial}{\partial S_h^{(j,:)}}\left(\frac{1}{1+\exp(-S_h^{(j,:)})}\right) = \frac{\exp(-S_h^{(j,:)})}{\left(1+\exp(-S_h^{(j,:)})\right)^2} = V_h^{(j,:)}\left(1-V_h^{(j,:)}\right) \tag{2.29}$$

and

$$\frac{\partial S_h^{(j,:)}}{\partial W_h^{(j,i)}} = \frac{\partial}{\partial W_h^{(j,i)}}\left(\sum_i W_h^{(j,i)} X_h{}^{(i,:)}\right) = X_h{}^{(i,:)} = V_{h-1}^{(i,:)} \tag{2.30}$$

So we can substitute (2.28), (2.29), and (2.30) into (2.24), then rewrite (2.24) as

$$W_{h(t+1)}^{(j,i)} = W_{h(t)}^{(j,i)} - \eta\left(\sum_l \delta_{h+1}^l W_{h+1}^{(l,j)}\right) V_h^{(j,:)}\left(1 - V_h^{(j,:)}\right) V_{h-1}{}^{(i,:)} \tag{2.31}$$

From (2.17) to (2.31), we have the following observations:

1. If the synaptic weights are between the hidden layer and output layer, then

$$\frac{\partial J}{\partial W_y^{(j,i)}} = (V_y^{(j,:)} - D^{(j,:)}) V_y^{(j,:)} (1 - V_y^{(j,:)}) V_{y-1}^{(i,:)} \tag{2.32}$$

$$\Rightarrow \quad \Delta W_y^{(j,i)} = \eta (V_y^{(j,:)} - D^{(j,:)}) V_y^{(j,:)} (1 - V_y^{(j,:)}) V_{y-1}^{(i,:)} \tag{2.33}$$

2. If the synaptic weights are between the hidden layers or between the input layer and the hidden layer, then

$$\frac{\partial J}{\partial W_h^{(j,i)}} = \left(\sum_l \delta_{h+1}^l W_{h+1}^{(l,j)}\right) V_h^{(j,:)}\left(1 - V_h^{(j,:)}\right) V_{h-1}{}^{(i,:)} \tag{2.34}$$

$$\Rightarrow \quad \Delta W_h^{(j,i)} = \eta\left(\sum_l \delta_{h+1}^l W_{h+1}^{(l,j)}\right) V_h^{(j,:)}\left(1 - V_h^{(j,:)}\right) V_{h-1}{}^{(i,:)} \tag{2.35}$$

Equation (2.33) and (2.35) are the most important formulas of the back-propagation algorithm. The synaptic weights can be adjusted by substituting (2.33) and (2.35) into the following (2.36):

$$W^{(j,i)}{}_{k(t+1)} = W^{(j,i)}{}_{k(t)} - \Delta W^{(j,i)}{}_{k(t)} \tag{2.36}$$

The learning process of back-propagation learning algorithm can be expressed by the following steps:

Back-Propagation learning algorithm:

Step 1: Decide the structure of the network for the problem.

Step 2: Choosing a suitable value between 0 and 1 for the learning rate η.

Step 3: Picking the initial synaptic weights from a uniform distribution whose value is usually small, like between -1 and 1.

Step 4: Calculate the output signal of the network by using (2.16).

Step 5: Calculate the error energy function J by using (2.11).

Step 6: Using (2.36) to update the synaptic weights.

Step 7: Back to **Step 4** and repeat **Step 4** to **Step 6** until the error energy function J is small enough.

2.5 Dynamic optimal training algorithm of a Three-layer Neural Network

Although *back-propagation algorithm* (BPA) is well known for dealing with more complex problems, which can not be solved by the single-layer perceptron, it still has some problems. An important issue is the value of learning rate. The learning rate decides the step-size of learning process. Although smaller learning rate can have a better chance to have convergent results, the speed of convergence is very slow and need more number of epochs. For larger learning rate, it can speed up the learning process, but it will create more unpredictable results. So how to find a suitable learning rate is also an important problem of the training of neural network. For solving this problem, we need to find a dynamic optimal learning rate of every iteration [19]. In most cases, a three-layer structure of neural network is enough to solve classification problems, so the *dynamical optimal training algorithm* will be proposed in this Chapter for a three layer neural network with sigmoid activation functions in the hidden and output layers.

2.5.1 The Architecture of A Three-Layer Network

Figure 2-7 shows a three-layer neural network adopted in this paper.

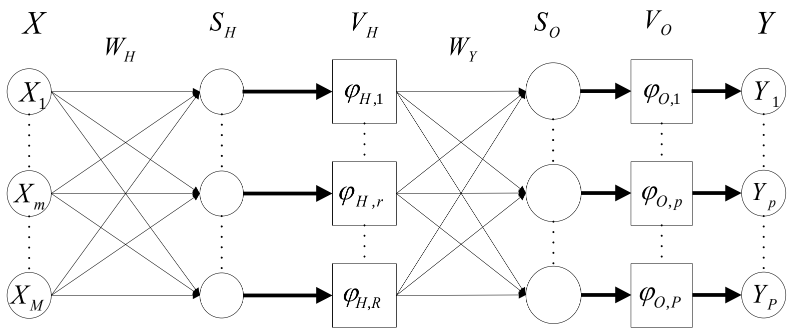


Figure 2-7. Three-layer Neural Network

Where

$\sum$

$X = \begin{bmatrix} X_1 & X_2 & \cdots & X_M \end{bmatrix}^T \in \mathbf{R}^{M\times N}$ is the input matrix,

$S_H = \begin{bmatrix} S_{H1} & S_{H2} & \cdots & S_{HR} \end{bmatrix}^T \in \mathbf{R}^{R\times N}$ is the output matrix of hidden-layer neurons,

$\sum$

$V_H = \begin{bmatrix} V_{H1} & V_{H2} & \cdots & V_{HR} \end{bmatrix}^T \in \mathbf{R}^{R\times N}$ is the output matrix of hidden-layer activation function,

$S_O = \begin{bmatrix} S_{O1} & S_{O2} & \cdots & S_{OP} \end{bmatrix}^T \in \mathbf{R}^{P\times N}$ is the output matrix of output-layer neurons,

$\sum$

$V_O = \begin{bmatrix} V_{O1} & V_{O2} & \cdots & V_{OP} \end{bmatrix}^T \in \mathbf{R}^{P\times N}$ is the output matrix of output-layer activation function,

Input layer Hidden layer

$Y = \begin{bmatrix} Y_1 & Y_2 & \cdots & Y_P \end{bmatrix}^T \in \mathbf{R}^{P\times N}$ is the actual output of neural network,

And N denotes the number of training data. M, R and P denote the number of input neurons, hidden-layer neurons and output-layer neurons, respectively. Further we define the weighting matrices:

$$W_H = \begin{bmatrix} W_{H(1,1)} & \cdots & \cdots & W_{H(1,M)} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ W_{H(R,1)} & \cdots & \cdots & W_{H(R,M)} \end{bmatrix}_{R\times M} \tag{2.37}$$

$$W_Y = \begin{bmatrix} W_{Y(1,1)} & \cdots & \cdots & W_{Y(1,R)} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ W_{Y(P,1)} & \cdots & \cdots & W_{Y(P,R)} \end{bmatrix}_{P\times R} \tag{2.38}$$

Where:

$W_{H(r,m)}$ denotes weight value between the mth neuron in the input layer and rth neuron in the hidden layer.

$W_{Y(p,r)}$ denotes weight value between the rth neuron in the hidden layer and pth neuron in the output layer.

And $\varphi_H(x)$ is the activation function of the hidden layer. $\varphi_O(x)$ is the activation function of the output layer. Usually, we both use the same sigmoid function for them. So we have

$$\varphi_H(x) = \varphi_O(x) = \frac{1}{1+e^{-x}} \tag{2.39}$$

2.5.2 Updating Process of a Three Layer Neural Network

Let $D = \begin{bmatrix} D_1 & D_2 & \cdots & D_P \end{bmatrix}^T \in \mathbf{R}^{P\times N}$ represents the desired output matrix and E denotes the error matrix and cost function J represents a normalized total square error function with η stands for the learning rate parameter in back-propagation algorithm. Based on BAP, the training process includes the following forward and backward passes:

(1) Forward pass (computing cost function):

$S_H = W_H X\big|_{R\times N}$; $V_H = \varphi_H(S_H)\big|_{R\times N}$; $S_O = W_Y V_H\big|_{P\times N}$; $Y = V_O = \varphi_O(S_O)\big|_{P\times N}$

$E = Y - D\big|_{P\times N}$;

The normalized total square error function is defined as:

$$J = \frac{1}{2PN}\sum_{p=1}^{P}\sum_{n=1}^{N}\left(Y^{(p,n)} - D^{(p,n)}\right)^2 = \frac{1}{2PN}\operatorname{trace}\left(EE'\right) \tag{2.40}$$

(2) Backward pass (update rule of synaptic weights):

$$W_{(t+1)} = W_{(t)} - \Delta W_{(t)} = W_{(t)} - \eta\frac{\partial J}{\partial W}\bigg|_t \tag{2.41}$$

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta\frac{1}{PN}(V_O^{(p,:)} - D^{(p,:)})V_O^{(p,:)}(1 - V_O^{(p,:)})V_H^{(r,:)} \tag{2.42}$$

$$W_{H(t+1)}^{(r,m)} = W_{H(t)}^{(r,m)} - \eta\left(\sum_{p=1}^{P}\frac{1}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right)V_O^{(p,:)}(1 - V_O^{(p,:)})W_Y^{(p,r)}\right)V_H^{(r,:)}\left(1 - V_H^{(r,:)}\right)X^{(m,:)} \tag{2.43}$$

2.5.3 Dynamical Optimal Training via Lyapunov's Method

We define the Lyapunov function as:

$$V = J$$

The item J is normalized total square error, defined in (2.40). Now if we consider learning rate η as a variable, then $J(\eta)$ denotes a positive function of η. We use $W_{Y(t+1)}$ and $W_{H(t+1)}$ to compute the normalized total square error of iteration $t+1$: J_{t+1}. And we can define the difference of the *Lyapunov function* as:

$$\Delta V = \Delta J_{(\eta,\, t)} = J_{(\eta,\, t+1)} - J_{(\eta,\, t)}$$

So if we can find a learning rate $\boldsymbol{\eta}$ to make both $\Delta J_{(\eta\,,t)} < 0$ and $\Delta J_{(\eta\,,t)}$ be at its minimum then we can get a stable and optimal convergence network. By using (2.40) we can get

$$J_{t+1} - J_t = \frac{1}{2PN}\sum_{p=1}^{P}\sum_{n=1}^{N}\left(Y_{t+1}^{(p,n)} - D^{(p,n)}\right)^2 - \frac{1}{2PN}\sum_{p=1}^{P}\sum_{n=1}^{N}\left(Y_t^{(p,n)} - D^{(p,n)}\right)^2 \qquad (2.41)$$

$$= \frac{1}{2PN}\sum_{p=1}^{P}\sum_{n=1}^{N}\left[\left(Y_{t+1}^{(p,n)} - D^{(p,n)}\right)^2 - \left(Y_t^{(p,n)} - D^{(p,n)}\right)^2\right]$$

$$= G\left(\eta(t)\right) \qquad (2.42)$$

From (2.42), if the parameter $\eta(t)$ satisfies $J_{t+1} - J_t = G(\eta(t)) < 0$, then $\eta(t)$ is the stable learning rate of the system at the tth iteration. For stable $\eta(t)$, if the $\boldsymbol{\eta}_{opt}(t)$ will let that $J_{t+1} - J_t$ be at its minimum, the $\boldsymbol{\eta}_{opt}(t)$ is the optimal learning rate at the tth iteration. The optimal learning rate $\boldsymbol{\eta}_{opt}(t)$ will not only guarantee the stability of the training process, but it also has the fastest speed of convergence.

How to find the optimal learning rate $\boldsymbol{\eta}_{opt}(t)$ from $G(\eta(t))$? Because $G(\eta(t))$ is a very complicated nonlinear algebraic function, it is hard to have a well designed formula for finding the optimal learning rate $\boldsymbol{\eta}_{opt}(t)$ from $G(\eta(t))$. Hence, we use the MATLAB routine *"fminbnd"* to search the optimal learning rate from (2.41).The calling sequence of "*fminbnd"* is [27]:

<u>FMINBND Scalar bounded nonlinear function minimization.</u>

X = FMINBND(FUN,x1,x2) starts at X0 and finds a local minimizer X of the function FUN in the interval x1 <= X <= x2. FUN accepts scalar input X and returns a scalar function value F evaluated at X.

The capability of the Matlab routine "***fminbnd***" is to find a local minimizer η_{opt} of the function $G(\eta)$, which has only one independent variable η, in a given interval. So we have to define an interval when we use this routine.

Algorithm 1:

Dynamic optimal training algorithm of a Three Layer Neural Network [19]

Step 0: Give input matrix X and desired output matrix D.

Step 1: Set initial weight factor W_H and W_Y, which are random values in a small random range. Set maximum acceptable final value of the cost function J_0.

Step 2: Iteration count k=1.

Step 3: Start forward pass of back-propagation training process.

Step 4: Compute error function E(k)=Y(k)-D(k) and cost function J(k). If J(k) is smaller than acceptable value J_0, the algorithm goes to Step 8. If no, it goes to Step 5.

Step 5: Using Matlab routine "fminbnd" with the search interval [0.01, 100] to solve the nonlinear function $\Delta J(k) = J(k+1)-J(k)$ and find the optimal learning rate $\eta_{opt}(t)$.

Step 6: Start backward pass of back-propagation training process. Update the synaptic weights matrix to yield $W_H(k+1)$ and $W_Y(k+1)$ by using (2.42) and (2.43) respectively.

Step 7: Iteration count k=k+1, and algorithm goes back to Step 3.

Step 8: End the algorithm.

CHAPTER 3

Dynamic Optimal Training Algorithm of A Three-Layer Neural Network with Better Performance

The structure of three layer neural network was presented in Chapter 2 with preparation for its improved dynamical optimal training to be presented in this Chapter 3. Although the dynamical optimal training of a three layer neural network was presented in [19], its back propagation algorithm with dynamical optimal learning rates can not guarantee global convergence. Although we can decrease the learning rate to have a better chance of global convergence, its convergence speed will be really slow. Also, the sigmoid function adopted for the activation functions for each layer will also slow down the training process. Therefore the following approaches will be proposed to further improve the performance of the dynamical optimal training of a three layer neural network. They are:

1. **Simplification of activation function:** Replace the sigmoid activation function in the second (output) layer with a linear activation function. (Section 3.2)
2. **Selection of proper initial weighting factors:** By confining the initial weighting matrices in a reasonable range, the training process can have a better chance to reach global convergence. (Section 3.3)
3. **Determine the upper-bound of learning rate in each iteration:** Therefore the search of optimal learning rate in each iteration can be simplified to increase the speed of convergence. (Section 3.4)

3.1 Problems in the Training of Three Layer Neural Network

There are two layers of sigmoid activation functions in **Fig. 2-6**, i.e., V_H and V_O for hidden and output layers. The purpose of activation function is to confine the output in each layer to a proper level. But the computing effort in computing the sigmoid function is much more than that in computing a linear function. Further it may be good enough to confine the output of hidden layer to a reasonable level. Once the data flow thru the output layer, the chance of the output data to lie outside proper level is small. These observations lead us to simplify the activation function in the output layer so that the speed of training process can be increased. Considering the defects of sigmoid function, whose graph is ***s***-shaped, as shown below (Figure 3-1) with

$$y = \frac{1}{1 + \exp(-ax)} \tag{3.1}$$

where y is the output signal of the neuron and x is the input signal. And the parameter a is the slope parameter of the sigmoid function. We can get different sigmoid functions by varying the slope parameter a, but we usually choose the sigmoid function with a =1.

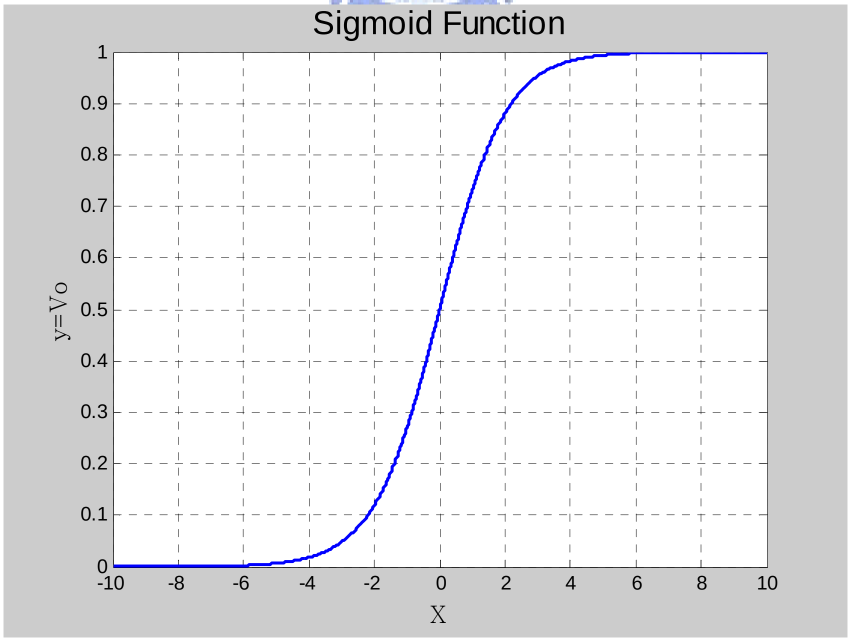


Figure 3-1. The sigmoid function

When x is far away from 0, we will get a saturated output value which is very close to extreme value, i.e., 0 or 1. Now we will discuss the problem which is caused by this saturation. First consider (2.42), which is update rule of weighting factors in output layer

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta \frac{1}{PN}(V_O^{(p,:)} - D^{(p,:)})V_O^{(p,:)}(1-V_O^{(p,:)})V_H^{(r,:)}$$

We define the error signal of the weight between pth output layer neuron and rth hidden layer neuron as

$$\delta_O^{(p,r)} = \frac{1}{PN}(V_O^{(p,:)} - D)V_O^{(p,:)}(1-V_O^{(p,:)})V_H^{(r,:)} \tag{3.2}$$

So (2.42) can be rewritten as

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta\delta_O^{(p,r)} \tag{3.3}$$

Consider (2.43), which is update rule of weighting factors in hidden layer

$$W_{H(t+1)}^{(r,m)} = W_{H(t)}^{(r,m)} - \eta\left(\sum_{p=1}^{P}\frac{1}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right)V_O^{(p,:)}(1-V_O^{(p,:)})W_Y^{(p,r)}\right)V_H^{(r,:)}\left(1-V_H^{(r,:)}\right)X^{(m,:)}$$

For simplicity, we define a new error signal as

$$\bar{\delta}_H^{(p,r)} = \frac{1}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right)V_O^{(p,:)}(1-V_O^{(p,:)}) \tag{3.4}$$

and the error signal of the weight between rth hidden layer neuron and mth input layer neuron as

$$\delta_H^{(r,m)} = \left(\sum_{p=1}^{P}\bar{\delta}_H^{(p,r)}W_Y^{(p,r)}\right)V_H^{(r,:)}\left(1-V_H^{(r,:)}\right)X^{(m,:)} \tag{3.5}$$

So, (2.43) can be rewritten as

$$W_{H(t+1)}^{(r,m)} = W_{H(t)}^{(r,m)} - \eta\delta_H^{(r,m)} \tag{3.6}$$

In (3.2) & (3.4) the term $V_O^{(p,n)}(1-V_O^{(p,n)})$ will propagate back to the hidden and input layers to adjust weighting factors. The term $V_O^{(p,n)}(1-V_O^{(p,n)})$ will be very small if $V_O^{(p,n)}$ is close to extreme values, i.e., 0 or 1. It will make error signal $\delta_O^{(p,r)}$ very small. In this case, the back-propagated error signal $\delta_O^{(p,r)}$ will not actually reflect the true error $V_O^{(p,n)}$-D = Y-D. This will lead to local minima problem. Note that the term $V_O^{(p,n)}(1-V_O^{(p,n)})$ is originated from the derivative of the sigmoid activation function. So it is important not to let the term $V_O^{(p,n)}(1-V_O^{(p,n)})$ closed to 0 or 1 in the beginning phase of training. Therefore the initial weighting factors must be carefully chosen. Further the sigmoid function may not be a good choice for the sake of computation effort and the above mentioned problem. Thus the search for a better and yet simplified activation function with the selection of proper initial weighting factors will be mentioned in 3.2 and 3.3.

3.2 Simplification of activation function

We define the linear function (without saturation) as

$$\varphi_O(x) = ax$$

where a is an adjustable parameter which means slope of the function. The reason of using an unbounded linear function in stead of a saturated one can be found in Appendix A. Usually, we set parameter $a = 1$.With the replacement of the sigmoid activation function by a linear activation function, we can redefine the new error signal of the weight between pth output layer neuron and rth hidden layer neuron. Consider (2.41), which is the update rule of weighting factors for BP algorithm

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta \frac{\partial J}{\partial W_Y^{(p,r)}} = W_{Y(t)}^{(p,r)} - \eta \frac{\partial J}{\partial V_O^{(p,:)}} \frac{\partial V_O^{(p,:)}}{\partial S_O^{(p,:)}} \frac{\partial S_O^{(p,:)}}{\partial W_Y^{(p,r)}} \qquad (3.7)$$

If $\varphi_O(x) = ax$, then the term $\partial V_O^{(p,:)} / \partial S_O^{(p,:)}$ will be equal to a. So $\partial J / \partial V_O^{(p,:)}$, $\partial V_O^{(p,:)} / \partial S_O^{(p,:)}$, and $\partial S_O^{(p,:)} / \partial W_Y^{(p,:)}$ have new presentations as

$$\frac{\partial J}{\partial V_O^{(p,:)}} = \frac{\partial}{\partial V_O^{(p,:)}} \left(\frac{1}{2PN} \sum_{p=1}^{P} \left(V_O^{(p,:)} - D^{(p,:)} \right)^2 \right) = \frac{1}{PN} \left(V_O^{(p,:)} - D^{(p,:)} \right) \qquad (3.8)$$

$$\frac{\partial S_O^{(p,:)}}{\partial W_Y^{(p,r)}} = \frac{\partial}{\partial W_Y^{(p,r)}}\left(\sum_{r=1}^{R} W_Y^{(p,r)} V_H^{(r,:)}\right) = V_H^{(r,:)} \tag{3.9}$$

$$\frac{\partial V_O^{(p,:)}}{\partial S_O^{(p,:)}} = \frac{\partial}{\partial S_O^{(p,:)}}\left(a S_O^{(p,:)}\right) = a \tag{3.10}$$

From previous three equations, we can rewrite the update rule of weighting factors in output layer as

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta \frac{a}{PN}(V_O^{(p,:)} - D^{(p,:)})V_H^{(r,:)} \tag{3.11}$$

And update rule of weighting factors in hidden layer can be expressed as

$$W_{H(t+1)}^{(r,m)} = W_{H(t)}^{(r,m)} - \eta\left(\sum_{p=1}^{P} \frac{a}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right)W_Y^{(p,r)}\right)V_H^{(r,:)}\left(1 - V_H^{(r,:)}\right)X^{(m,:)} \tag{3.12}$$

Hence, the error signals $\delta_O^{(p,r)}$ and $\delta_H^{(r,m)}$ become

$$\delta_O^{(p,r)} = \frac{a}{PN}(V_O^{(p,:)} - D)V_H^{(r,:)} \tag{3.13}$$

$$\delta_H^{(r,m)} = \left(\sum_{p=1}^{P} \frac{a}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right)W_Y^{(p,r)}\right)V_H^{(r,:)}\left(1 - V_H^{(r,:)}\right)X^{(m,:)} \tag{3.14}$$

We can see that the error signal $\delta_O^{(p,r)}$ in (3.13) will not be discounted by the term $V_O^{(p,n)}$(1-$V_O^{(p,n)}$), like that in (2.42) (or (3.3)). This will remove the effect of improper initial weighting factors to let the term $V_O^{(p,n)}$(1-$V_O^{(p,n)}$) close to 0 or 1 in the beginning phase of training. From (3.14), although the term $V_O^{(p,n)}$(1-$V_O^{(p,n)}$) is also eliminated (from (3.4)), we still have the similar term $V_H^{(r,n)}$(1-$V_H^{(r,n)}$) in (3.14). Therefore we should do is to confine the selection of initial weighting factors in a proper range so that the term $V_H^{(r,n)}$(1-$V_H^{(r,n)}$) will not be closed to 1 or 0 in the beginning phase of training. The reason why we can not replace the sigmoid activation function in the hidden layer with the similar linear activation function can be shown in Appendix B.

3.3 Selection of Initial Weighting Factors

For simplicity, we define that

$$\tilde{\delta}_H = \left(\sum_{p=1}^{P} \frac{a}{PN} \left(V_O^{(p,:)} - D^{(p,:)} \right) W_Y^{(p,r)} \right) \tag{3.15}$$

So (3.14), which is the new error signal in hidden layer, can be rewritten as

$$\delta_H^{(r,m)} = \tilde{\delta}_H V_H^{(r,:)} \left(1 - V_H^{(r,:)} \right) X^{(m,:)} \tag{3.16}$$

It can be found that the term $V_H^{(r,n)}(1-V_H^{(r,n)})$ still causes the saturation problem. When $V_H^{(r,n)}$ closes to 0 or 1, the term $V_H^{(r,n)}(1-V_H^{(r,n)})$ will discount back-propagated real error signal (3.15). In (3.16), the term $V_H^{(r,n)}$ is the output of sigmoid activation function, and it is a function of initial weighting factors. If we can confine $V_H^{(r,n)}$ in the range of, say [0.2, 0.8], then the term $(1-V_H^{(r,n)})$ in (3.16) will also be confined in the range of [0.2, 0.8]. Therefore the whole term $V_H^{(r,n)}(1-V_H^{(r,n)})$ in (3.16) will not close to zero if proper initial weighting factors are selected. This will prevent the error signal of hidden layer in (3.16) be discounted by the whole term $V_H^{(r,n)}(1-V_H^{(r,n)})$ in the beginning phase of training. The $V_H^{(r,n)}$ can be expressed as

$$V_H^{(r,n)} = 1 \Big/ \left(1 + \exp\left(-\sum_i W_h^{(j,i)} X_h^{(i,:)} \right) \right) \tag{3.17}$$

where X_h^i is the fixed input data. To decrease the effect of saturation region, we can choose proper initial weighting factors so that the term $V_H^{(r,n)}$ will locate in the range of [0.2, 0.8] at the beginning of training process.

It is a normal practice to use random number generator to select the initial weighting factors. Let

$$w_{\max} = Max(W)$$
$$w_{\min} = Min(W)$$

Where W is the weighting matrix. It is also shown in [17] that the training of NN will have better convergence if the mean of random initial weighting factors is zero. Therefore we let $w_{\min} < 0 < w_{\max}$ and $w_{\max} = -w_{\min}$. We use a sigmoid activation function with parameter a = 1 in hidden layer. Then we must select $w_{\min}$ and $w_{\max}$ to satisfy the following inequality:

$$0.2 = y_s(w_{\min}, \lambda) \le y_s(w, \lambda) \le y_s(w_{\max}, \lambda) = 0.8 \tag{3.18}$$

with

$$\mathrm{y_s}(w, \lambda) = \frac{1}{1+\exp(-w\lambda)} \tag{3.19}$$

We let $\lambda = |x_{\max}|$ in the following Theorem 1, where $|x_{\max}|$ is the maximum absolute value of the elements of input matrix and $w_{\min} \le w \le w_{\max}$. From (3.18), we can have

$$\begin{cases} w_{\max} = \dfrac{\ln 4}{|x_{\max}|} \\ w_{\min} = \dfrac{-\ln 4}{|x_{\max}|} \end{cases} \Rightarrow |w| \le \frac{\ln 4}{|x_{\max}|} \tag{3.20}$$

Theorem 1:

If w satisfies (3.20), then the output of sigmoid function in (3.19), i.e., $y_s(w, |x_{\max}|)$, *will be in the range of [0.2, 0.8].*

Proof:

Know that $w_{\min} \le w \le w_{\max}$

$\because$ $y_s(w, \lambda)$ is a strictly increasing function so if $w\lambda \le \overline{w\lambda} \Rightarrow y_s(w\lambda) \le y_s(\overline{w\lambda})$

$$\therefore \; y_s(w_{\min}, |x_{\max}|) \le y_s(w, |x_{\max}|) \le y_s(w_{\max}, |x_{\max}|)$$

$$\because \; w_{\min} = -\frac{\ln 4}{|x_{\max}|} \quad \text{and} \quad w_{\max} = \frac{\ln 4}{|x_{\max}|}$$

$$y_s\left(w_{\min},|x_{\max}|\right)=\frac{1}{1+\exp(-(-\frac{\ln 4}{|x_{\max}|}|x_{\max}|))}=0.2$$

$$\therefore$$

$$y_s\left(w_{\max},|x_{\max}|\right)=\frac{1}{1+\exp(-\frac{\ln 4}{|x_{\max}|}|x_{\max}|)}=0.8$$

$$\therefore\quad 0.2=y_s\left(w_{\min},|x_{\max}|\right)\le y_s\left(w,|x_{\max}|\right)\le y_s\left(w_{\max},|x_{\max}|\right)=0.8$$

$$\Rightarrow\quad 0.2\le y_s\left(w,|x_{\max}|\right)\le 0.8$$

Q.E.D.

Theorem 2:

If $0.2\le y_s(w,|x_{\max}|)\le 0.8$*, then* $0.2\le y_s(w,|x_i|)\le 0.8$*. Here* $y_s(w,|x_{\max}|)$ *&* $y_s(w,|x_i|)$ *are defined in (3.19) and w is selected from (3.20).*

Proof:

(1) For $w>0$

$$\because\quad 0\le w|x_i|\le w|x_{\max}|$$

$$\therefore\quad 0.5=y_s\left(w,0\right)\le y_s\left(w,|x_i|\right)\le y_s\left(w,|x_{\max}|\right)$$

$$\therefore\quad y_s\left(w,|x_i|\right)\ge 0.5$$

$$\because\quad y_s\left(w,|x_{\max}|\right)\le 0.8$$

$$\therefore\quad 0.5\le y_s\left(w,|x_i|\right)\le 0.8\qquad\Rightarrow\qquad 0.2<y_s\left(w,|x_i|\right)\le 0.8\ \ldots\ldots\text{(a)}$$

(2) For $w<0$

$$\because\quad w|x_{\max}|\le w|x_i|\le 0$$

$$\therefore\quad y_s\left(w,|x_{\max}|\right)\le y_s\left(w,|x_i|\right)\le y_s\left(w,0\right)=0.5$$

$$\therefore\quad y_s\left(w,|x_i|\right)\le 0.5$$

$$\because\quad y_s\left(w,|x_{\max}|\right)\ge 0.2$$

$$\therefore \quad 0.2 \le y_s\left(w,|x_i|\right) \le 0.5 \quad \Rightarrow \quad 0.2 \le y_s\left(w,|x_i|\right) < 0.8 \ldots\ldots\text{(b)}$$

From (a) and (b), we have the final conclusion

$$0.2 \le y_s(w,|x_i|) \le 0.8$$

Q.E.D.

Theorem 3:

If $0.2 \le y_s(w,|x_i|) \le 0.8$, *then* $0.2 \le y_s(w,x_i) \le 0.8$. *Here w is selected from (3.20).*

Proof:

(1) $x_i > 0$

$$\because \quad y_s(w,|x_i|) = y_s(w,x_i)$$

$$\therefore \quad 0.2 \le y_s(w,|x_i|) = y_s(w,x_i) \le 0.8 \quad \Rightarrow \quad 0.2 \le y_s(w,x_i) \le 0.8$$

(2) $x_i < 0$

$$\because \quad y_s(w,x_i) = y_s(w,-|x_i|)$$

$$\Rightarrow \quad y_s\left(w,-|x_i|\right) = \frac{1}{1+\exp(-w(-|x_i|))} = \frac{1}{1+\exp(-(-w)|x_i|)} = y_s\left(-w,|x_i|\right)$$

$$\Rightarrow \quad y_s\left(-w,|x_i|\right) = y_s\left(\tilde{w},|x_i|\right)$$

$$\therefore \quad y_s(w,x_i) = y_s(\tilde{w},|x_i|)$$

From Theorem 2, we have $0.2 \le y_s\left(\tilde{w},|x_i|\right) \le 0.8$

$$\therefore \quad 0.2 \le y_s(w,x_i) \le 0.8$$

Q.E.D.

The above Theorem 3 is our final conclusion, which selects proper weighting factors to let $0.2 \le y_s(w,x_i) = \dfrac{1}{1+\exp(-wx_i)} \le 0.8$, so that the error signal (3.16) in hidden layer will not be discounted in the beginning of training process. This will also reduce the probability of falling into in local minimum on the error surface during the back propagation process.

Example 3-1: Selection of Initial Weighting Factors

Give the following neural network with three layers, two inputs in the input layer, one output neuron in the output layer, and two neurons in the hidden layer. It adopts sigmoid function as hidden layer activation function, and the above mentioned linear function as output layer activation function. The input signals are bound at (-2, 2). The architecture of the neural network is shown in Figure 3-2. Find the initial weighting factors so that the initial outputs of the sigmoid functions in the hidden layer will be in the range of [0.2, 0.8].

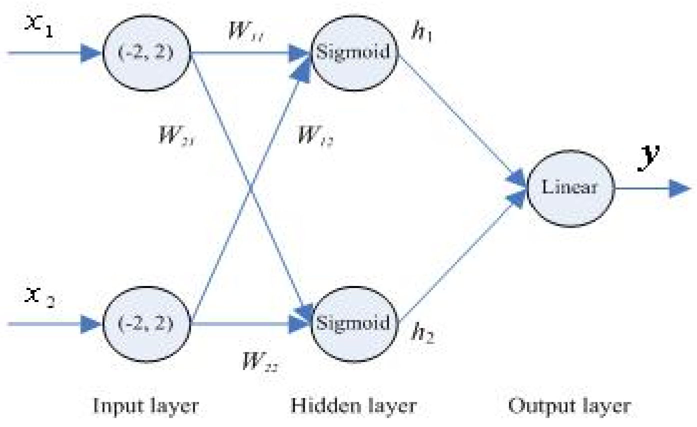


Figure 3-2. Neural Network for Example 3-1

Solution:

Since $x_{\max} = 2$, from (3.19) we can get:

$$w_{\min} = \frac{-\ln 4}{|x_{\max}|} = \frac{-\ln 4}{|2|} = \frac{-\ln 4}{2} = -0.6931 \quad \text{and} \quad w_{\max} = \frac{\ln 4}{|x_{\max}|} = \frac{\ln 4}{|2|} = \frac{\ln 4}{2} = 0.6931$$

Hence, the range of initial weight factors is [-0.6931, 0.6931]. For verification, we assume four different input pair [$\boldsymbol{x_1}$, $\boldsymbol{x_2}$] and randomly choose four initial weighting factors [$\boldsymbol{w_{11}}$, $\boldsymbol{w_{12}}$, $\boldsymbol{w_{21}}$, $\boldsymbol{w_{22}}$] = [**-0.22669**, **-0.15842**, **0.33254**, **-0.17167**] from [-0.6931, 0.6931], we have the following Table 3-1.

Table 3-1 Output of Sigmoid function in hidden layer

$\boldsymbol{x}_1$	-0.01	0.7	-1.91	0.67
$\boldsymbol{x}_2$	1.2	2	-1.5	0.34
$\boldsymbol{h}_1$	0.45318	0.38331	0.66164	0.44874
$\boldsymbol{h}_2$	0.44786	0.47239	0.40670	0.54102

From Table 3-1, we can find that for every input pair [x_1, x_2], the initial outputs of hidden neurons [h_1, h_2] are all in the range of [0.2, 0.8].

END

To summarize the above proposed issues, the following Figure 3-3 shows the modified three-layer NN proposed in this Chapter 3.

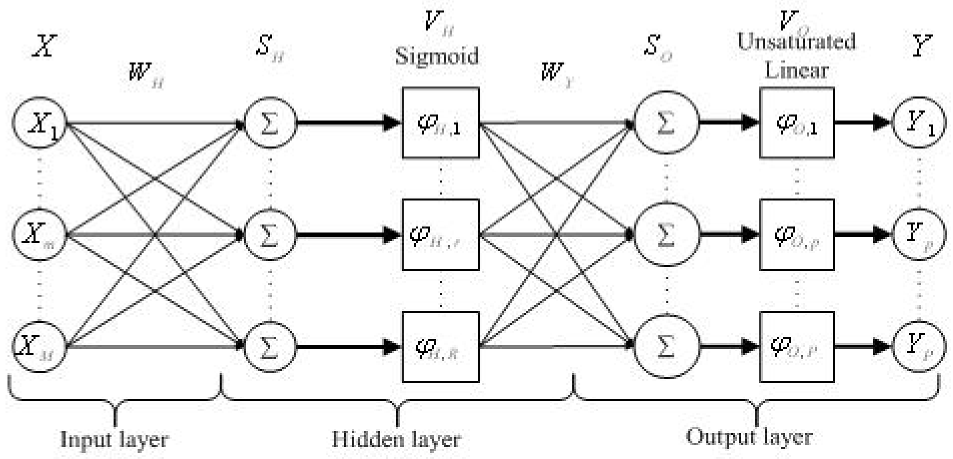


Figure 3-3. Revised Three Layer Neural Network

3.4 Determine the upper-bound of learning rate in each iteration

In the dynamic optimal training algorithm, we use matlab function "*fminbnd*" to find optimal learning rate. The calling sequence of "*fminbnd*" is [27]:

***FMINBND** Scalar bounded nonlinear function minimization.*

X = FMINBND(FUN,x1,x2) starts at X0 and finds a local minimizer X of the function

FUN in the interval x1 <= X <= x2. FUN accepts scalar input X and returns a scalar function value F evaluated at X.

Matlab function "*fminbnd*" can find a global minima value in a given range. Usually, we give a fixed interval to find the optimal learning rate. But it take too mach time to search because of fixed interval. From the experimental result, we can see that the optimal learning rate may go from 0.001 to 1000. It is a very large interval. Further, it can be discovered that most learning rate in each iteration does not exceed 100. So a fixed range will waste a lot of time to find an optimal learning rate during the learning process. Besides, most of other mathematical methods also need an interval to find a global minima value. Indeed, the upper bound of stable learning rate of the three layer NN shown in Figure 3-3 can be found in the following Theorem 4. This upper-bound will be updated in each iteration and changed according to the maxima value of input matrix and output weighting matrix.

Theorem 4:

The upper-bound of the stable learning rate of the three-layer NN shown in Figure 3-3 can be shown as follows:

$$\eta_{up} = \frac{32P \cdot N}{a^2 N\left(16R + P \cdot R \cdot M \cdot W_{Y\max}^2 \cdot X_{\max}^2\right)}$$

where a is the slope of the output layer activation function, W_{Ymax} and X_{max} are the acceptable maximum absolute value for the second (output) layer weighting matrix and for input matrix, respectively.

<u>Proof:</u>

Figure 3-3 is a three-layer NN which has a sigmoid activation function in the hidden layer and a linear activation function with slope is a in the output layer. M is the number of input neurons, R is the number of hidden-layer neurons, P is the number of output-layer neurons, and N is the length of input data vector. k is the time instant. Consider the following cost function to be minimized:

$$J(k) = \frac{1}{2PN} e'(k)e(k) \tag{3.21}$$

where $e(k) = D(k) - y(k)$ is the error vector with $D \in \mathbb{R}^{P\times 1}$ and $y \in \mathbb{R}^{P\times 1}$ denote the desired output and real output vectors, respectively. W is the weighting matrix. We define the update rule of the weighting matrix W in the form:

$$\Delta W(k) = W(k+1) - W(k) = -\eta \cdot \nabla_W J(k) \tag{3.22}$$

where η is a positive variable called learning rate of the learning algorithm based on the update rule (3.22), with cost function (3.21). $\nabla_W J(k)$ is the cost function gradient related to the vector of adjustable weights, which can be expressed as

$$\begin{aligned} \nabla_W J(k) &= \frac{1}{PN}\left[\frac{\partial e(k)}{\partial W}\right]' e(k) \\ &= \frac{\text{-1}}{\text{PN}}\left[\frac{\partial y(k)}{\partial W}\right]' e(k) \end{aligned} \tag{3.23}$$

where $\partial y(k)/\partial W \in \mathbb{R}^{P\times S}$ is the Jacobian of the NN's output related to the weighting vector. The update of $W(k)$ will continue until $J(\cdot) \le J_0$. Here J_0 is the specified tolerance for the process to converge. After convergence, we should have

$\Delta W(k) = W(k+1) - W(k) = -\eta \cdot \nabla_W J(k) \approx 0$

The difference of error vector *e(k)* can be approximated by the first form of Taylor's series

$$\begin{aligned}\Delta e(k) &\cong \frac{\partial e(k)}{\partial W}\Delta W(k) \\ &= e(k+1) - e(k)\end{aligned} \tag{3.24}$$

Consider the Lyapunov function:

$$v(k) = e'(k)e(k)$$

$$\Delta v(k) = v(k+1) - v(k)$$

$$\Rightarrow \Delta v(k) = e'(k+1)e(k+1) - e'(k)e(k) \tag{3.25}$$

$\Delta v(k)$ is the difference between two instants of time in Lyapunov function during the training process. Applying (3.22) and (3.23) into (3.24), and we can rewrite Δ*e(k)* as the following equation.

$$\begin{aligned}\Delta e(k) &\cong \frac{\partial e(k)}{\partial W}\Delta W(k) \\ &= -\frac{\partial y(k)}{\partial W}\left(-\eta\nabla_W J(k)\right) \\ &= \frac{\partial y(k)}{\partial W}\left(-\frac{\eta}{PN}\frac{\partial y(k)}{\partial W}\right)' e(k) \\ &= -\frac{\eta}{PN}\left(\frac{\partial y(k)}{\partial W}\right)\left(\frac{\partial y(k)}{\partial W}\right)' e(k)\end{aligned}$$

We denote $\partial y(k)/\partial W$ by A, and rewrite Δ*e(k)* in the form of

$$\Delta e(k) = -\frac{\eta}{PN}AA'e(k) = -\frac{\eta}{PN}Qe(k) \tag{3.26}$$

where $Q = AA' \in \mathbf{R}^{P\times P}$ is a symmetrical positive definite matrix.

From (3.26), we have

$$e(k+1) - e(k) = -\frac{\eta}{PN}Qe(k) \Rightarrow e(k+1) = e(k) - \frac{\eta}{PN}Qe(k)$$

$$\therefore e(k+1) = \left\{-\frac{\eta}{PN}Q + I^{P\times P}\right\}e(k) \tag{3.27}$$

Substituting (3.27) into (3.25), we can rewrite $\Delta v(k)$ in the form of

$$\begin{aligned}\Delta v(k) &= \left[\left(-\frac{\eta}{PN}Q+I\right)e(k)\right]'\left[\left(-\frac{\eta}{PN}Q+I\right)e(k)\right]-e'(k)e(k)\\
&= e'(k)\left(-\frac{\eta}{PN}Q+I\right)'\left(-\frac{\eta}{PN}Q+I\right)e(k)-e'(k)e(k)\\
&= e'(k)\left(-\frac{\eta}{PN}Q+I\right)^2 e(k)-e'(k)e(k) \qquad (\because Q=Q', I'=I)\\
&= e'(k)\left[\left(-\frac{\eta}{PN}Q+I\right)^2-I\right]e(k)\\
&= e'(k)\left[I^2-2I\frac{\eta}{PN}Q+\left(\frac{\eta}{PN}\right)^2 Q^2-I\right]e(k)\\
&= e'(k)\left(-2\frac{\eta}{PN}Q+\left(\frac{\eta}{PN}Q\right)^2\right)e(k)\\
&\Rightarrow \Delta v(k) = -\frac{\eta}{PN}e'(k)Q\left\{2I-\frac{\eta}{PN}Q\right\}e(k) \qquad (3.28)\end{aligned}$$

Let $W^* \subset \mathbf{R}^R$ be the vector of optimal weight that minimizes the cost function in (3.21) *and* $W^0 \subset W^*$ be the initial weighting factor. By *Lyapunov's second method*, if $\Delta v(k) < 0$ then the learning of NN will be stable and the learning algorithm will converge, i.e., the weighting matrix will converge to W^* from W^0. In (3.28), we have already known that $\eta > 0, P > 0$, $Q > 0$ *,and* $N > 0$. So the term: $B = 2I - (\eta/PN)Q$ must be a positive definite matrix. In other word, all eigenvalues of the matrix B must be positive. We let $\lambda_{\max}(Q(k))$ be the maximum eigenvalue of Q at kth iteration with

$$2-\frac{\eta}{PN}\lambda^*_{\max} > 0 \qquad \Rightarrow \qquad \frac{2PN}{\lambda^*_{\max}} > \eta > 0$$

where $\lambda^*_{\max} = \max\{\lambda_{\max}(Q(k))\}$. The term Q is a time variant matrix, so it is hard to find its maximum eigenvalue for the whole training process. In order to simplify the calculation, we use the trace of Q at kth iteration. Trace of Q is the summation of all eigenvalues of Q, so its value is larger than $\lambda^*_{\max}$. Define $\|\square\|^2$ for 2-norm, $\|x\|^2 = x'x$ for vector x, and $\|A\|^2 = trace(A'A)$ for matrix A. Hence, we have

$$\Rightarrow \frac{2PN}{\lambda^*_{\max}} \geq \frac{2PN}{tr(Q)} = \frac{2PN}{tr(AA')} = \frac{2PN}{\|\partial y(k)/\partial W\|^2} > \eta > 0$$

$$\Rightarrow \quad 0 < \eta < \frac{2PN}{\|\partial y(k)/\partial W\|^2} \tag{3.29}$$

The term $\|\partial y(k)/\partial W\|^2$ in (3.29) can not be easily found. So we can rewrite (3.29) as:

$$0 < \eta < \frac{2PN}{\max\limits_k \left\{ \|\partial y(k)/\partial W\|^2 \right\}} \tag{3.30}$$

The advantage of (3.30) is that we can find a clear and definite presentation for the term $\max\limits_k\{\|\partial y(k)/\partial W\|^2\}$. The maximum value of Jacobian of the NN's output related to the matrix of weighting matrix W can be expressed as

$$\begin{aligned}
\max_k \left\{ \left\| \frac{\partial Y(k)}{\partial W} \right\|^2 \right\} &= \max_k \left\{ trace\left[\left(\frac{\partial Y(k)}{\partial W} \right) \left(\frac{\partial Y(k)}{\partial W} \right)' \right] \right\} \\
&= \max_k \left\{ \left\| \frac{\partial Y(k)}{\partial W_Y} \right\|^2 + \left\| \frac{\partial Y(k)}{\partial W_H} \right\|^2 \right\} \\
&= \max_k \left\{ tr\left[\left(\frac{\partial Y(k)}{\partial W_Y} \right) \left(\frac{\partial Y(k)}{\partial W_Y} \right)' \right] + tr\left[\left(\frac{\partial Y(k)}{\partial W_H} \right) \left(\frac{\partial Y(k)}{\partial W_H} \right)' \right] \right\}
\end{aligned} \tag{3.31}$$

From Figure 3-3, the revised algorithm of the learning process can be expressed as:

$$\begin{cases}
S_H = W_H X \big|_{R\times N} ; V_H = \varphi_H(S_H) \big|_{R\times N} \\
S_O = W_Y V_H \big|_{P\times N} ; V_O = \varphi_O(S_O) \big|_{P\times N} \\
Y = V_O \\
\varphi_H(x) = 1\big/\left(1 + e^{-x}\right) ; \varphi_O(x) = ax
\end{cases} \tag{3.32}$$

where a is the slope of the output layer linear activation function. X is the input matrix with dimension M×N. W_Y is the weighting matrix of output layer with dimension P×R.

W_H is the weighting matrix of hidden layer with dimension R×M. By chain rule, term $\left[\partial Y(k)/\partial W_Y \quad \partial Y(k)/\partial W_H\right]$ can be expressed as

$$\begin{cases} \dfrac{\partial Y(k)}{\partial W_Y} = \dfrac{\partial \varphi_O(k)}{\partial S_O}\dfrac{\partial S_O(k)}{\partial W_Y} \\ \dfrac{\partial Y(k)}{\partial W_H} = \dfrac{\partial \varphi_O(k)}{\partial S_O}\dfrac{\partial S_O(k)}{\partial \varphi_H}\dfrac{\partial \varphi_H(k)}{\partial S_H}\dfrac{\partial S_H(k)}{\partial W_H} \end{cases} \tag{3.33}$$

From (3.32) and (3.33), the update rule of synaptic weighting matrix (W = [W_Y, W_H]) of the neural network during the learning process can be defined by

$$\begin{cases} \Delta W(k) = \begin{cases} 0 & \text{if } J(\cdot) \le J_0 \\ \dfrac{\eta}{PN}\left[\dfrac{\partial Y(k)}{\partial W_Y} \quad \dfrac{\partial Y(k)}{\partial W_H}\right]' e(k) = \dfrac{\eta}{PN}\left[a\dfrac{\partial S_O(k)}{\partial W_Y} \quad aW_Y\dfrac{\partial \varphi_H(k)}{\partial S_H}\dfrac{\partial S_H(k)}{\partial W_H}\right]' e(k) & \text{if } J(\cdot) > J_0 \end{cases} \\ \Delta W(0) = 0 \end{cases} \tag{3.34}$$

We define that $W_{Y\,max}$ and X_{max} are the maximum absolute value for the element of output layer weighting matrix and for the element of input matrix, respectively. From (3.32) to (3.34), we can find $\partial Y(k)/\partial W_Y$ as

$$\max_k\left\{tr\left(\frac{\partial Y(k)}{\partial W_Y}\left(\frac{\partial Y(k)}{\partial W_Y}\right)'\right)\right\} = \max_k\left\{tr\left(aV_H V_H{}'a\right)\right\}$$

$$\Rightarrow \max_k\left\{tr\left(\frac{\partial Y(k)}{\partial W_Y}\left(\frac{\partial Y(k)}{\partial W_Y}\right)'\right)\right\} = \max_k\left\{tr\left(a^2\begin{bmatrix} V_{H(1,1)}^2 + \ldots V_{H(1,N)}^2 & & \\ & \ddots & \\ & & V_{H(R,1)}^2 + \ldots V_{H(R,N)}^2 \end{bmatrix}\right)\right\}$$

$$\Rightarrow \max_k\left\{tr\left(\frac{\partial Y(k)}{\partial W_Y}\left(\frac{\partial Y(k)}{\partial W_Y}\right)'\right)\right\} = a^2 RNV_{H\max}^2 \tag{3.35}$$

From (3.32) to (3.34), we can find $\partial Y(k)/\partial W_H$ as

$$\max_k\left\{tr\left(\frac{\partial Y(k)}{\partial W_H}\left(\frac{\partial Y(k)}{\partial W_H}\right)'\right)\right\} = \max_k\left\{tr\left(aW_Y\frac{\partial \varphi_H(k)}{\partial S_H}XX'\left[\frac{\partial \varphi_H(k)}{\partial S_H}\right]' W_Y{}'a\right)\right\}$$

$$= \max_k\left\{tr\left(a^2\left|\frac{\partial \varphi_H(k)}{\partial S_H}\right|^2 W_Y W_Y{}' XX'\right)\right\}$$

$$\therefore \max_k \left\{ tr\left(\frac{\partial Y(k)}{\partial W_H} \left(\frac{\partial Y(k)}{\partial W_H} \right)' \right) \right\} = a^2 \left| \frac{\partial \varphi_H(k)}{\partial S_H} \right|^2_{\max} PRW^2_{Y\max} MNX^2_{\max} \tag{3.36}$$

In (3.35), V_H is the output of sigmoid function, so we can get $V_{H\max} = 1 \rightarrow V^2_{H\max} = 1$.

In (3.36), $|\partial \varphi_H(k)/\partial S_H|$ is the slope of sigmoid function with slope parameter a =1, so its maximum value is $a/4$ =0.25. $|\partial \varphi_H(k)/\partial S_H|_{\max} = 1/4 \rightarrow |\partial \varphi_H(k)/\partial S_H|^2_{\max} = 1/16$

Substituting (3.35) and (3.36) into (3.31), we can obtain

$$\begin{aligned} \max_k \left\{ \left\| \frac{\partial Y(k)}{\partial W} \right\|^2 \right\} &= a^2 RNV^2_{H\max} + a^2 \left| \frac{\partial \varphi_H(k)}{\partial S_H} \right|^2_{\max} PRMNW^2_{Y\max} X^2_{\max} \\ &= a^2 RN + \frac{a^2 PRMNW^2_{Y\max} X^2_{\max}}{16} \end{aligned} \tag{3.37}$$

Then we can say that the system is stable and will converge to W^* if

$$0 < \eta < \frac{2PN}{\max_k \left\{ \|\partial Y(k)/\partial W\|^2 \right\}}$$

Substituting (3.37) into (3.30), then we get

$$0 < \eta < \frac{32P \cdot N}{a^2 N \left(16R + P \cdot R \cdot M \cdot W^2_{Y\max} \cdot X^2_{\max}\right)} \qquad \text{Q.E.D.}$$

<u>**Example 3-2: Finding the upper-bound of optimal learning rate**</u>

Give the following neural network with three layers, four inputs in the input layer, three output neurons in the output layer, and four neurons in the hidden layer. W_H is a 4×4 matrix and W_Y is a 3×4 matrix. The architecture of the neural network is shown in Figure 3-4. The sigmoid function is adopted as hidden layer activation function, and a linear function as output layer activation function. The input signals are bounded at (-2, 2). The slope parameter of linear activation function is given as 1. Try to get an upper bound for the learning rate at first iteration.

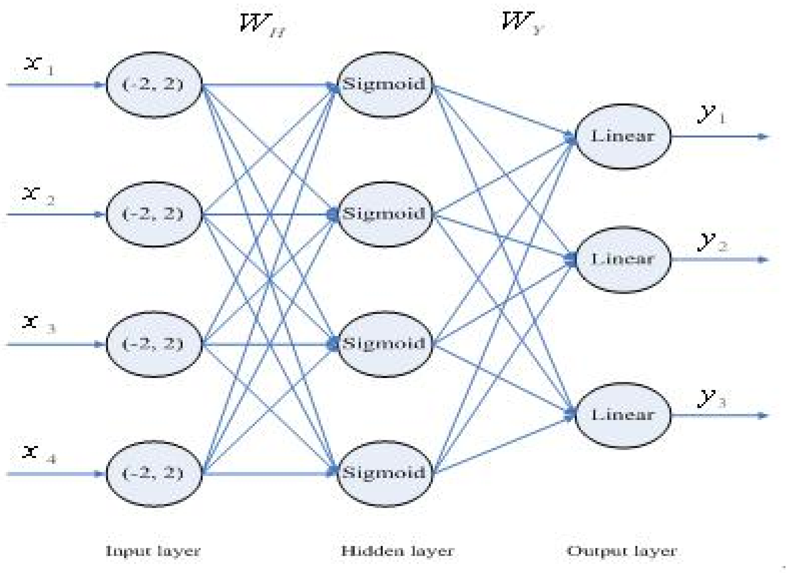


Figure 3-4. Neural network for Example 3-2

Solution:

Give a=1, M=4, R=4, P=3, and X_{max}=2. From (3.19) we can get:

$$W_{\min} = \frac{-\ln 4}{|X_{\max}|} = \frac{-\ln 4}{|2|} = \frac{-\ln 4}{2} = -0.6931 \text{ and } W_{\max} = \frac{\ln 4}{|X_{\max}|} = \frac{\ln 4}{|2|} = \frac{\ln 4}{2} = 0.6931$$

Hence, the range of initial weighting factors is [-0.6931, 0.6931]. Therefore W_{Ymax}=0.6931. Using Theorem 4, we can get the upper bound for the learning rate at first iteration:

$$\eta_{upper-bound} = \frac{32P \cdot N}{a^2 N\left(16R + P \cdot R \cdot M \cdot W_{Y\max}^2 \cdot X_{\max}^2\right)} = \frac{32 \cdot 3}{1^2(16 \cdot 4 + 3 \cdot 4 \cdot 4 \cdot (0.6931)^2 \cdot 2^2)}$$

$$= 0.614461$$

END

3.5 Conclusion

From Section 3.1 to Section 3.4, we can summarize the revised dynamic optimal training algorithm for a modified three-layer neural network as shown in Figure 3-3.

Math model in matrix format:

The math model of revised dynamic optimal training algorithm is shown as:

(1) Feed-forward process:

$$S_H = W_H X\big|_{R\times N}$$

$$V_H = \varphi_H(S_H)\big|_{R\times N}$$

$$S_O = W_Y V_H\big|_{P\times N}$$

$$Y = V_O = \varphi_O(S_O)\big|_{P\times N}$$

$$E = Y - D\big|_{P\times N} \text{ and } J = \frac{1}{2PN}\operatorname{trace}\left(EE'\right)$$

(2) Back-forward process:

$$W_{(t+1)} = W_{(t)} - \Delta W_{(t)} = W_{(t)} - \eta \frac{\partial J}{\partial W}\bigg|_t$$

Update rule of synaptic weight factors:

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta\,\delta_{O-revised}^{(p,r)} \tag{3.38}$$

$$W_{H(t+1)}^{(r,m)} = W_{H(t)}^{(r,m)} - \eta\,\delta_{H-revised}^{(r,m)} \tag{3.39}$$

$$\delta_{O-revised}^{(p,r)} = \frac{a}{PN}(V_O^{(p,:)} - D)V_H^{(r,:)} \tag{3.40}$$

$$\delta_{H-revised}^{(r,m)} = \left(\sum_{p=1}^{P} \frac{a}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right)W_Y^{(p,r)}\right)V_H^{(r,:)}\left(1 - V_H^{(r,:)}\right)X^{(m,:)} \tag{3.41}$$

<u>Algorithm 2: Revised dynamic optimal training algorithm</u>

<u>Step 0</u>: Give input matrix X and desired output matrix D. Find the maxima absolute value of input matrix: $|X_{\max}|$. Compute the bound of initial weight factors $[w_{\min}, w_{\max}]$ by using (3.20).

<u>Step 1</u>: Set initial weight factor W_H and W_Y, which are random values in the bound of $[w_{\min}, w_{\max}]$. Set maximum acceptable final value of the cost function J_0.

<u>Step 2</u>: Iteration count k=1.

<u>Step 3</u>: Start feed-forward part of back-propagation training process.

<u>Step 4</u>: Compute error function E(k)=Y(k)-D(k) and cost function J(k). If J(k) is smaller than acceptable value J_0, the algorithm goes to <u>Step 9</u>. If no, it goes to <u>Step 5</u>.

<u>Step 5</u>: Find the maxima absolute value of W_Y: $|W_{Y\max}|$. Compute learning rate upper-bound: η_{up} by using <u>Theorem 4</u>.

<u>Step 6</u>: Using Matlab routine "fminbnd" with the search interval $[0.01, \eta_{up}]$ to solve the nonlinear function $\Delta J(k) = J(k+1) - J(k)$ and find the stable learning rate η_{opt}.

<u>Step 7</u>: Start back-forward part of back-propagation training process. Update the synaptic weights matrix to yield $W_H(k+1)$ and $W_Y(k+1)$ by using (3.38) and (3.39) respectively.

<u>Step 8</u>: Iteration count k = k+1, and algorithm goes back to <u>Step 3</u>.

<u>Step 9</u>: End the algorithm

CHAPTER 4

Experimental Results

In this chapter, we will solve the classification problems of XOR and Iris Data via the *revised dynamic optimal training algorithm* discussed in Chapter 3. The training result will be compared with *dynamic optimal training algorithm* and *back-propagation training algorithm* using a fixed learning rate.

4.1 Example 1: The XOR problem

The task is to train the network to produce the Boolean "Exclusive OR" (**XOR**) function of two variables. The **XOR** operator yields true if exactly one (but not both) of two conditions is true, otherwise the **XOR** operator yields false. The truth table of **XOR** function is shown in Figure 4-1.

A	B	Y
T	T	F
T	F	T
F	T	T
F	F	F

Figure 4-1. The truth table of XOR function: Y = A $\oplus$ B

4.1.1 System modeling and Problem definition

We need only consider four input data (0,0), (0,1), (1,1), and (1,0) in this problem. The first and third input patterns are in class 0, which means the XOR operator yields "False" when the input data is (0,0) or (1,1). The second and fourth input patterns are in class 1, which means the XOR operator yields "True" when the input data is (0,1) or (1,0). The distribution of the input data is shown in Figure 4-2. Because there are two variables of XOR function, we choose the input layer with two neurons and the output layer with one neuron. Then we use one hidden layer with two neurons to solve XOR problem [22], as shown in Figure 4-3. The architecture of the neural network is 2-2-1 network.

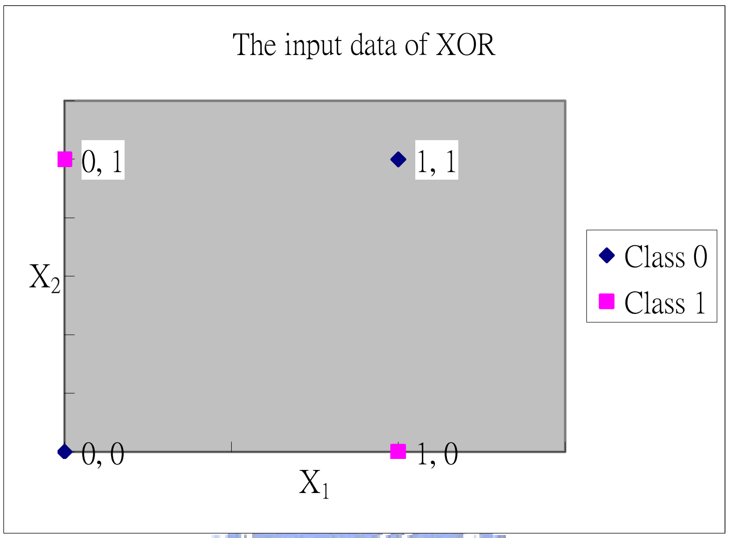


Figure 4-2. The distribution of XOR input data sets

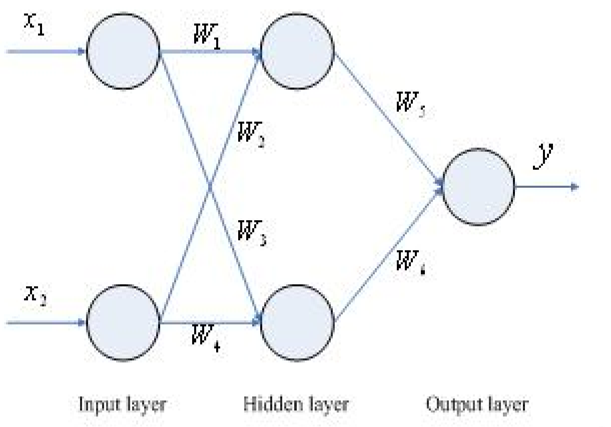


Figure 4-3. The neural network for solving XOR

Hence, we have the weight matrix which is between input layer and hidden layer in the form

$$W_H = \begin{bmatrix} W_{H(1,1)} & W_{H(1,2)} \\ W_{H(2,1)} & W_{H(2,2)} \end{bmatrix} = \begin{bmatrix} W_1 & W_2 \\ W_3 & W_4 \end{bmatrix}$$

And the weight matrix between hidden layer and output layer can be expressed as

$$W_Y = \begin{bmatrix} W_{Y(1,1)} & W_{Y(1,2)} \end{bmatrix} = \begin{bmatrix} W_5 & W_6 \end{bmatrix}$$

4.1.2 Experimental result analysis

We first compare the training experimental result by respectively using *Back-propagation Algorithm* (BPA), *Dynamic Optimal Training Algorithm* (DOA), and *Revised Dynamic Optimal Training Algorithm* (RDOA). Table 4-1 shows experimental results of three kinds of algorithm. All results are averaged over 20 trials. For BPA, we use a learning rate of 0.9 and the initial weight are drawn from a uniform distribution over interval [-1, 1]. For DOA, we search optimal learning rate by using Matlab routine “fminbnd” with searching range [0.01, 100] and the initial weight are drawn from a uniform distribution over interval [-1, 1]..

Table 4-1 Training results of XOR problems for a 2-2-1 neural network with three different kinds of algorithm: BPA, DOA, and RDOA

	BPA	DOA	RDOA
J (final normalized total square error)	0.063825 (diverged result)	0.000617	0.000994
T (training time, sec)	Diverged	17.617	22.63
SI (convergence iteration when J(k)<0.0078125)	Diverged	2684	775

ST (convergence time, sec)	Diverged	4.72	1.75
C (convergence probability in 20 trials)	0%	20%	70%

J is total square error and it can be expressed in the form of

$$J = \frac{1}{2PN}\sum_{k}^{4} e_k^{\ 2} = \frac{1}{8}\sum_{k}^{4} e_k^{\ 2}$$

For XOR problem, $P=1$ and $N=4$. e_k is error signal for k^{th} element of output vector.

T is total training time for 10000 training iteration. ***SI*** is first iteration at which the value of ***J*** is smaller than 0.005. We assume that when $e_k=0.1$, it is enough to say that the network can classify XOR data set. So it means that when $J=0.005$, the network can achieve our requirement. ***ST*** is actual settling time and it can be calculate in the form of [$ST = T \times SI \div (total\ iteration)$]. ***C*** is times of successful convergence for training neural network in 20 trials.

For DOA and RDOA, we averaged the value of J, T, and S only when we have convergence. But the column of BPA listed in Table 4-1 is somehow different from others. For BPA, we averaged the value of *J* and *T* for all 20 trials. Because the output of network trained by BPA can not get a correct result for all 20 trials. From Table 4-1, we can find that BPA is the worst algorithm to train a neural network. The convergence probability *C* of BPA is 0%, so BPA has no settling iteration. From row *J* in Table 4-1, DOA has a better normalized total square error (.000617) than RDOA (.000994) if both algorithms have correct results. But the small J in both algorithms are small enough to classify XOR data set. From row *C*, we see that RDOA has a greater probability (70%) than DOA (20%) to have a correct result. It means that RDOA has a high capability of jumping out local minimum. From row *ST*, we can know that RDOA only spend 1.75 seconds to get normalized total square error J smaller than 0.005 or get into a steady state. RDOA is 2.7 times faster than DOA (4.72 sec).

The best result of using BP algorithm with fixed learning rate 0.9 to train the XOR is shown Figure 4-4. The best result of using DO algorithm with initial weight drawn from a uniform distribution over interval [-1, 1] to train the XOR is shown in Figure 4-5. The best result of using RDO algorithm to train XOR is shown in Figure 4-6. The comparison of the best results of three algorithms is shown in Figure 4-7-1 and Figure 4-7-2.

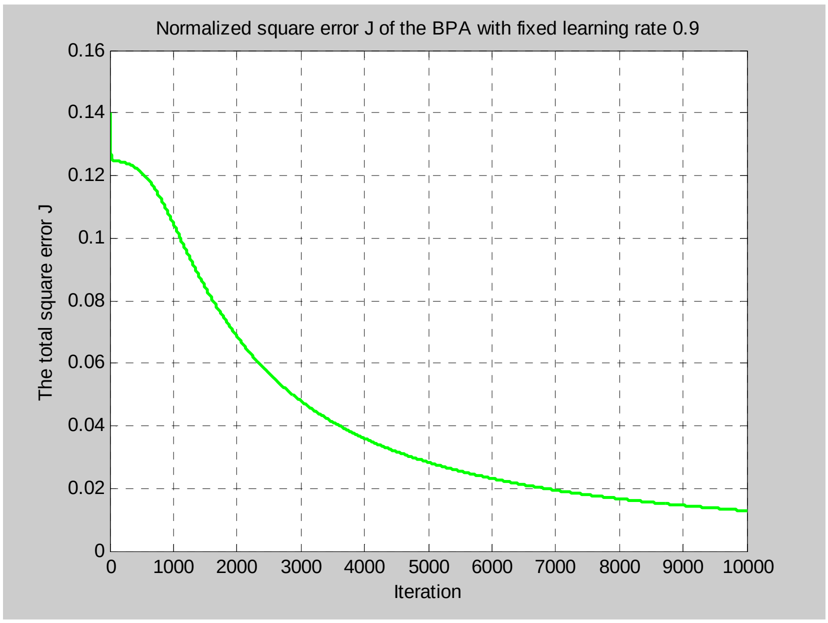


Figure 4-4. Normalized square error *J* of the standard BPA with fixed η = 0.9

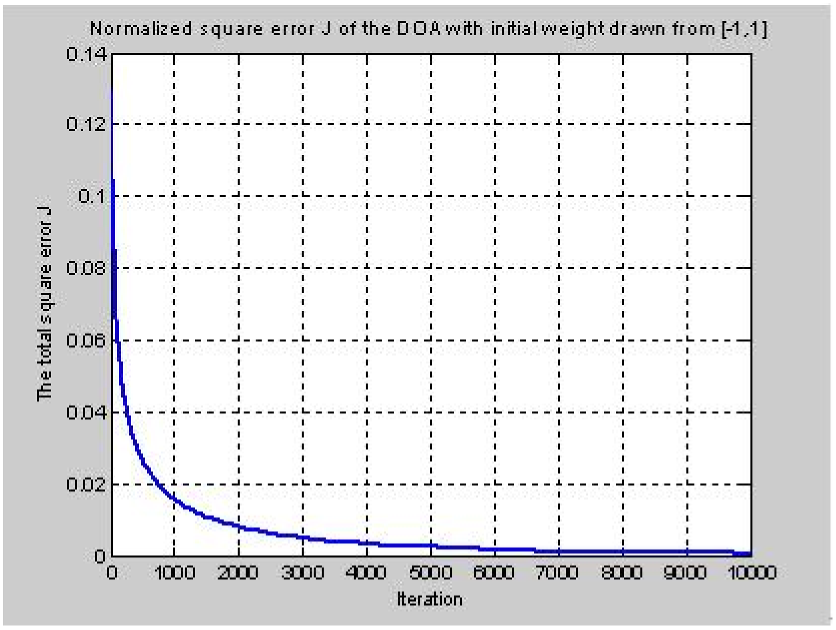


Figure 4-5. The best normalized square error *J* of the DOA in 20 trials

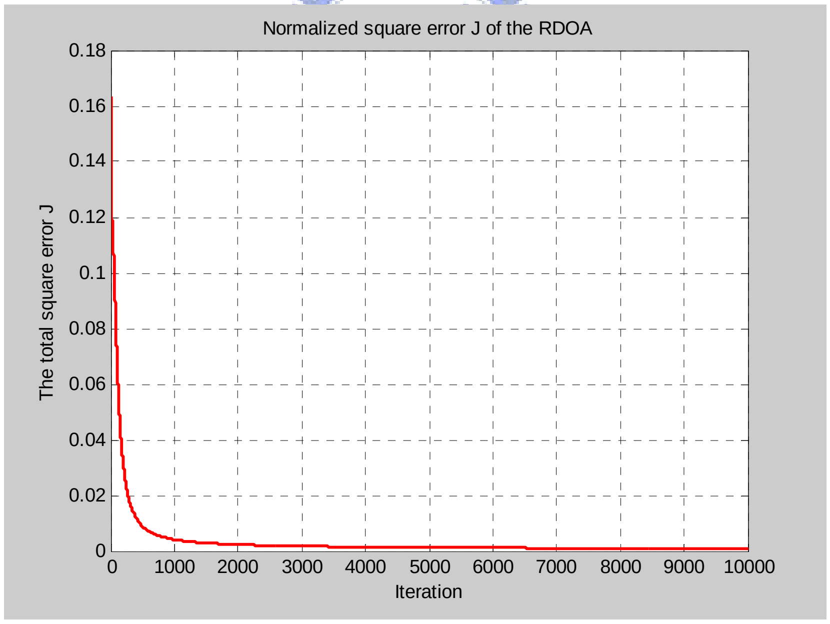


Figure 4-6. The best normalized square error *J* of the RDOA in 20 trials

In Figure 4-7-1, RDOA and DOA have almost the same performance at the end of training process. They are different at the beginning of training process. In Figure 4-7-2, it is obvious that RDOA achieves the minimum acceptable square error 0.005 around the 800th iteration. At the same iteration, DOA still has a higher square error 0.02. So RDOA can train the XOR data set faster than that of DOA .

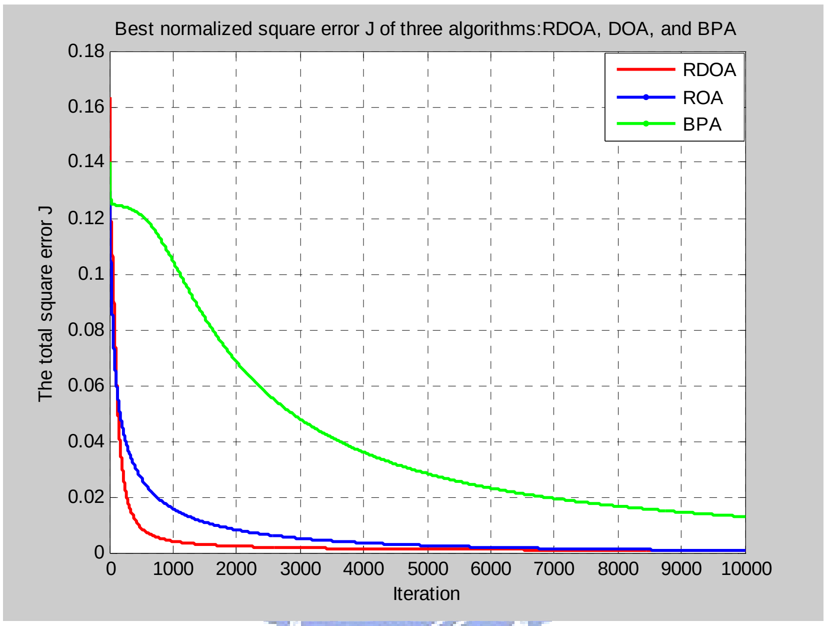


Figure 4-7-1. Training error of RDOA, DOA, and BPA

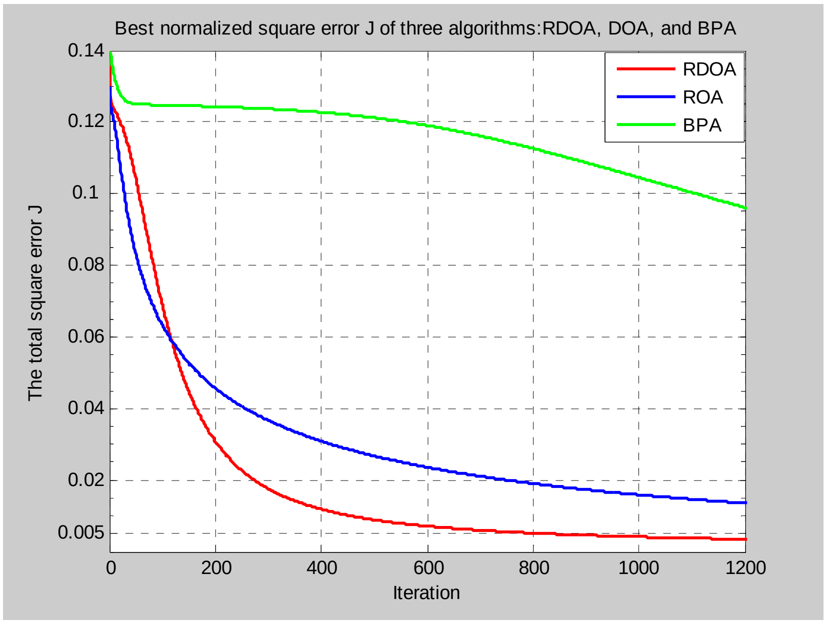


Figure 4-7-2. The close look of training error

Tables 4-2 ~ 4-4 show the complete training results of three algorithm. F in the column C denotes failed result of convergence. S in the column C denotes successful result of convergence.

Table 4-2 Detailed Training results of XOR problems for a 2-2-1 neural network with Back-propagation algorithm.

	BAP_fixed rate_0.9 (Have tuning initial weight)			
	J (total square error)	T (cost time) sec	SI (convergent iteration)	C (convergence)
1	0.0842000	1.34	N/A	**F**
2	0.0890000	1.64	N/A	**F**
3	0.0891000	1.67	N/A	**F**
4	0.0899000	1.73	N/A	**F**
5	0.0507000	1.64	N/A	**F**
6	0.0841000	1.60	N/A	**F**
7	0.0891000	1.53	N/A	**F**
8	0.0133000	1.68	N/A	**F**
9	0.0634000	1.51	N/A	**F**
10	0.0844000	1.46	N/A	**F**
11	0.0160000	1.46	N/A	**F**
12	0.0515000	1.54	N/A	**F**
13	0.0227000	1.59	N/A	**F**
14	0.0129000	1.5	N/A	**F**
15	0.0908000	1.59	N/A	**F**
16	0.0634000	1.62	N/A	**F**
17	0.0841000	1.59	N/A	**F**
18	0.0577000	1.60	N/A	**F**
19	0.0894000	1.81	N/A	**F**
20	0.0508000	1.64	N/A	**F**
Average	**0.0638250**	**1.59135**	**N/A**	**0%**

Table 4-3 Detailed Training results of XOR problems for a 2-2-1 neural network with Dynamic Optimal Training algorithm.

	XOR_Dynamic Optimal(Have tuning initial weight)			
	J (total square error)	T (cost time) sec	SI (convergent iteration)	C (convergence)
1	0.0008415	17.28	3189	**S**
2	0.0833000	24.25	N/A	**F**
3	0.0004563	17.67	2330	**S**
4	0.0360000	35.35	N/A	**F**
5	0.0003109	18.04	2007	**S**
6	0.0008610	17.46	3209	**S**
7	0.0902000	23.28	N/A	**F**
8	0.0625000	40.35	N/A	**F**
9	0.0360000	32.35	N/A	**F**
10	0.0360000	35.43	N/A	**F**
11	0.0880000	23.34	N/A	**F**
12	0.0360000	34.09	N/A	**F**
13	0.1143000	24.29	N/A	**F**
14	0.0902000	22.92	N/A	**F**
15	0.0902000	22.57	N/A	**F**
16	0.0902000	22.78	N/A	**F**
17	0.0625000	40.75	N/A	**F**
18	0.0360000	32.96	N/A	**F**
19	0.0625000	40.40	N/A	**F**
20	0.0880000	23.07	N/A	**F**
Average	**0.0006174**	**17.617**	**2683.75**	**20%**

Table 4-4 Detailed Training results of XOR problems for a 2-2-1 neural network with Revised Dynamic Optimal Training algorithm.

	XOR_Revised Dynamic Optimal			
	J (total square error)	T (cost time) sec	SI (convergent iteration)	C (convergence)
1	0.0007546	20.68	798	S
2	0.0009093	24.78	1173	S
3	0.0296000	24.56	N/A	F
4	0.0305000	27.12	N/A	F
5	0.0015000	29.32	582	S
6	0.0007413	21.43	416	S
7	0.0007545	22.03	798	S
8	0.0007598	21.31	957	S
9	0.0384000	19.84	N/A	F
10	0.0296000	26.70	N/A	F
11	0.0015000	30.64	582	S
12	0.0879000	14	N/A	F
13	0.0297000	27.09	N/A	F
14	0.0007413	22.81	416	S
15	0.0015000	29.48	591	S
16	0.0008408	12.92	966	S
17	0.0008372	14.18	935	S
18	0.0015000	30.67	736	S
19	0.0007652	22.26	1068	S
20	0.0008195	14.34	829	S
Average	**0.0009945**	**22.636143**	**774.78571**	**70%**

Table 4-5 shows the training result via revised dynamical optimal training (RDOA) for XOR problem. Table 4-6 shows the training result via dynamical optimal training (DOA) for XOR problem. To compare Table 4-5 with Table 4-6, we can see that the training result via revised dynamical optimal training is faster with better result than other approaches.

Table 4-5. The training result for XOR using revised dynamical optimal training

Training Results \ Iterations	1	1000	5000	10000
W_1 (after trained)	**1.2480**	**5.1970**	**5.8972**	**8.4681**
W_2 (after trained)	**0.2963**	**5.2356**	**5.9167**	**8.4703**
W_3 (after trained)	**-0.7454**	**1.3695**	**1.8095**	**0.9573**
W_4 (after trained)	**-0.0388**	**1.3708**	**1.8100**	**0.9573**
W_5 (after trained)	**1.0849**	**5.6486**	**8.1003**	**9.1849**
W_6 (after trained)	**0.7266**	**-5.8580**	**-8.2528**	**-9.3141**
Actual Output Y for $(x_1, x_2) = (0,0)$	**0.9058**	**-0.10308**	**-0.0759**	**-0.0644**
Actual Output Y for $(x_1, x_2) = (0,1)$	**0.9785**	**0.9503**	**0.9871**	**0.9918**
Actual Output Y for $(x_1, x_2) = (1,0)$	**1.0768**	**0.9503**	**0.9871**	**0.9918**
Actual Output Y for $(x_1, x_2) = (1,1)$	**1.1218**	**0.1496**	**0.0634**	**0.0484**
J	**0.2606**	**0.0047**	**0.0012**	**0.0008**

Table 4-6. The training result for XOR using dynamical optimal training

Training Results \ Iterations	1	1000	5000	10000
W_1 (after trained)	**0.6428**	**6.5146**	**7.6615**	**8.1360**
W_2 (after trained)	**0.2309**	**6.5302**	**7.6665**	**8.1391**
W_3 (after trained)	**-0.1106**	**0.8578**	**0.9246**	**0.9454**
W_4 (after trained)	**0.5839**	**0.8579**	**0.9246**	**0.9454**
W_5 (after trained)	**0.8436**	**14.698**	**25.604**	**32.201**
W_6 (after trained)	**0.4764**	**-18.751**	**-32.213**	**-40.337**
Actual Output Y for $(x_1, x_2) = (0,0)$	**0.6592**	**0.1167**	**0.0354**	**0.0162**
Actual Output Y for $(x_1, x_2) = (0,1)$	**0.6848**	**0.8226**	**0.9260**	**0.9589**
Actual Output Y for $(x_1, x_2) = (1,0)$	**0.6852**	**0.8226**	**0.9260**	**0.9589**
Actual Output Y for $(x_1, x_2) = (1,1)$	**0.7086**	**0.2381**	**0.0984**	**0.0554**
J	**0.1419**	**0.0166**	**0.0027**	**0.0008**

4.2 Example 2: Classification of Iris Data Set

In this example, our task is to train the neural network to classify Iris data sets [23], [24]. Generally, Iris has three kinds of subspecies: setosa, versicolor, virginica. The classification will depend on the length and width of the petal and the length and width of the sepal.

4.2.1 System modeling and Problem definition

The total Iris data are shown in Figures 4-8-1 and 4-8-2. The first 75 samples of total data are the training data, which are shown in Figures 4-9-1 and 4-9-2. The Iris data samples are available in [28]. There are 150 samples of three species of the Iris flowers in this data. We choose 75 samples to train the network and using the other 75 samples to test the network. We will have four kinds of input data, so we adopt the network which has four nodes in the input layer and three nodes in the output layer for this problem. Then the architecture of the neural network is a 4-4-3 network as shown in Figure 4-10. In which, we use the network with four hidden nodes in the hidden layer. When the input data set belongs to class setosa, the output of network will be expressed as [1 0 0]. For class versicolor, the output is set to [0 1 0]. For class virginica, the output is set to [0 0 1].

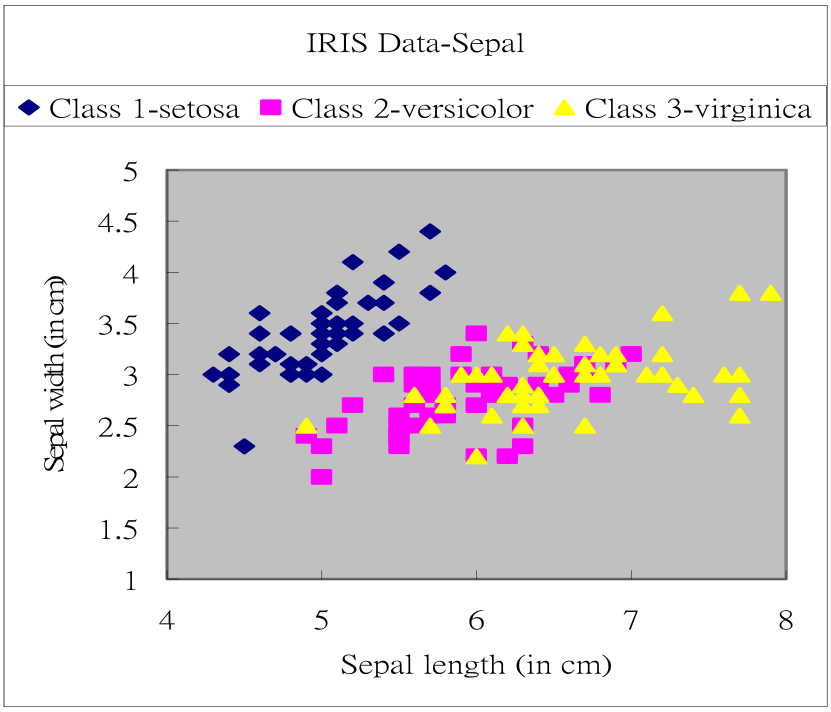


Figure 4-8-1. The total Iris data set (Sepal)

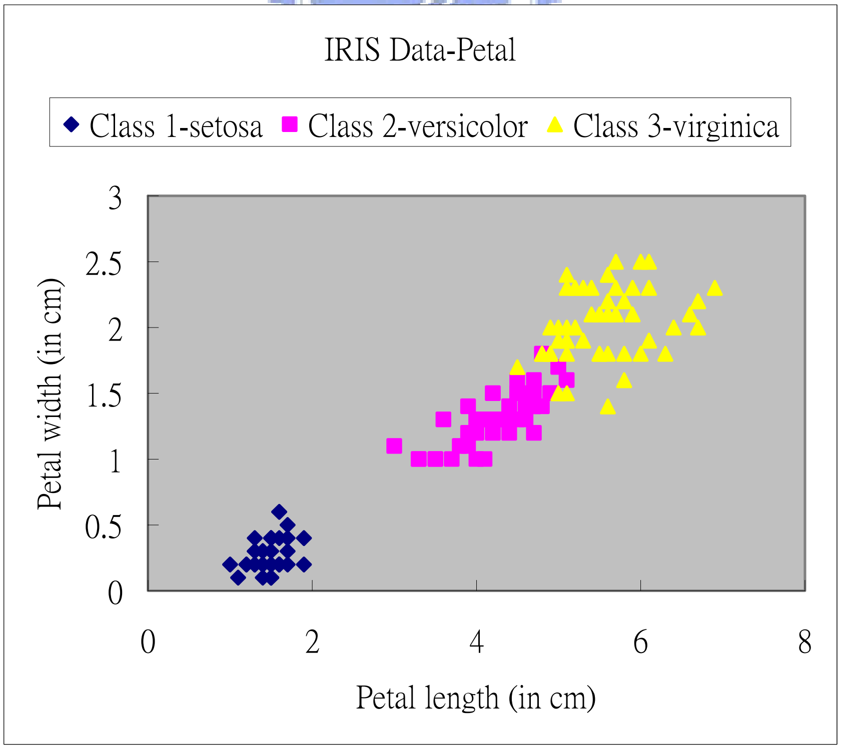


Figure 4-8-2. The total Iris data set (Petal)

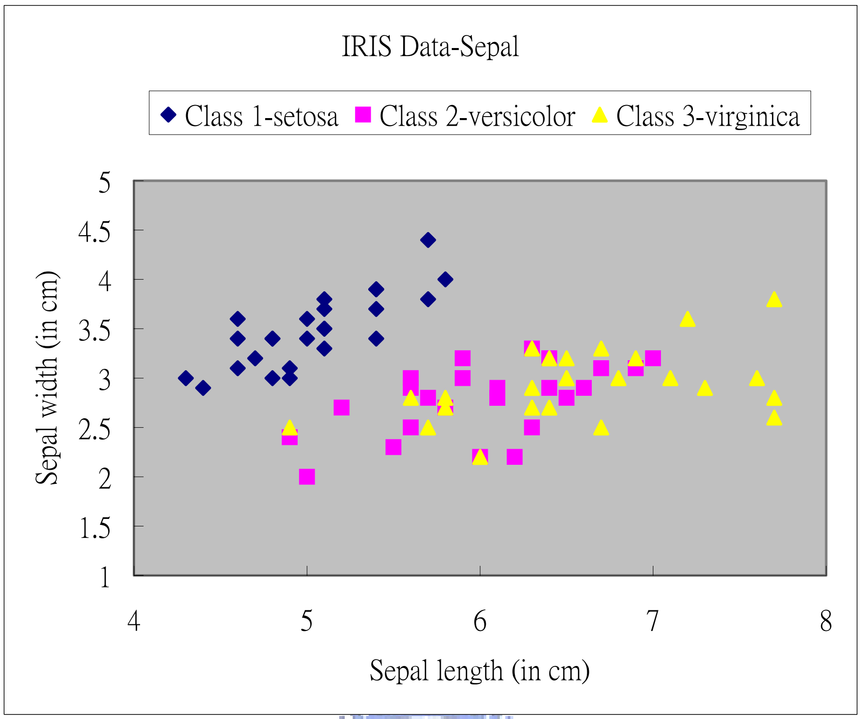


Figure 4-9-1. The training set of Iris data (Sepal)

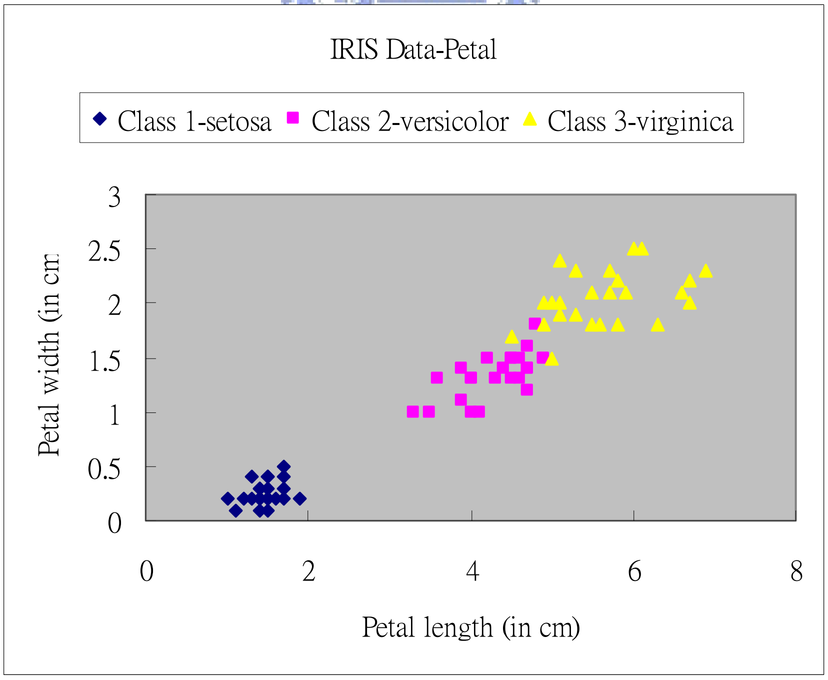


Figure 4-9-2. The training set of Iris data (Petal)

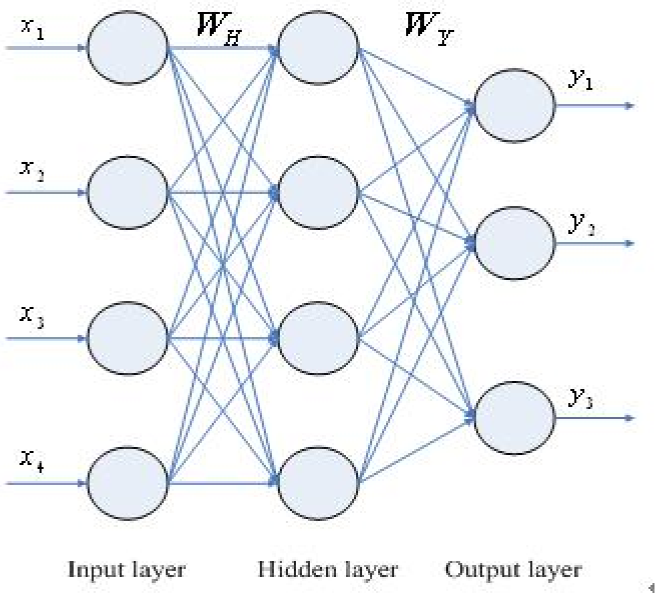


Figure 4-10. The neural network for solving Iris problem

We have the weight matrix which is between input layer and hidden layer in the form

$$W_H = \begin{bmatrix} W_{H(1,1)} & W_{H(1,2)} & W_{H(1,3)} & W_{H(1,4)} \\ W_{H(2,1)} & W_{H(2,2)} & W_{H(2,3)} & W_{H(2,4)} \\ W_{H(3,1)} & W_{H(3,2)} & W_{H(3,3)} & W_{H(3,4)} \\ W_{H(4,1)} & W_{H(4,2)} & W_{H(4,3)} & W_{H(4,4)} \end{bmatrix}_{4 \times 4}$$

And the weight matrix between hidden layer and output layer can be expressed as

$$W_Y = \begin{bmatrix} W_{Y(1,1)} & W_{Y(1,2)} & W_{Y(1,3)} & W_{Y(1,4)} \\ W_{Y(2,1)} & W_{Y(2,2)} & W_{Y(2,3)} & W_{Y(2,4)} \\ W_{Y(3,1)} & W_{Y(3,2)} & W_{Y(3,3)} & W_{Y(3,4)} \end{bmatrix}_{3 \times 4}$$

4.2.2 Experimental result analysis

We first compare the training experimental result of network by respectively using *Back-propagation Algorithm* (BPA), *Dynamic Optimal Training Algorithm* (DOA), and *Revised Dynamic Optimal Training Algorithm* (RDOA). Table 4-7 shows experimental results of three kinds of algorithm. All results are averaged over 20 trials.

For BPA, we use a learning rate of 0.01 for 10 trials and a learning rate of 0.1 for another 10 trials. Initial weights of BPA are drawn from a uniform distribution over interval [-1, 1]. For DOA, we search optimal learning rate by using Matlab routine "fminbnd" with searching range [0.01, 1000] and the initial weight are drawn from a uniform distribution over interval [-1, 1].

Table 4-7 Training results of IRIS problems for a 4-4-3 neural network with three different kinds of algorithm: BPA, DOA, and RDOA

	BPA	DOA	RDOA
J (final normalized total square error)	0.079515 (diverged result)	0.000626	0.003836
T (training time, sec)	Diverged	41.09	22.87
SI (convergence iteration when J(k)<0.0078125)	Diverged	654	529
ST (convergence time, sec)	Diverged	2.68	1.21
C (convergence probability in 20 trials)	0%	20%	95%

J is total square error and it can be expressed in the form of

$$J = \frac{1}{2PN}\sum_{h}^{P}\sum_{k}^{N} e_{kh}^{\ 2} = \frac{1}{450}\sum_{h}^{3}\sum_{k}^{75} e_{kh}^{\ 2}$$

For IRIS problem, $P=3$ and $N=75;$ e_{kh} is error signal for (kth ,hth) element of output vector; ***T*** is total training time for 10000 training iteration; ***SI*** is first iteration at which the value of ***J*** is smaller than 0.0078125. We assume that when $e_k=0.125$, it is enough to say that the network can classify IRIS data set. So it means that when $J=0.0078125$, the network can achieve our requirement. ***ST*** is actual settling time and it can be calculate in the form of [$ST = T \times SI \div$ (*total iteration*)]. ***C*** is times of successful convergence for training neural network in 20 trials.

From row *J* in Table 4-7, DOA has a better normalized total square error (.000626) than RDOA (.003836) if both algorithms have correct results. But the Js in both algorithms are small enough to classify IRIS data set. From row *C*, we see that RDOA has a greater probability (95%) than DOA (20%) to have a correct result. It means that RDOA has a high capability of jumping out local minimum. From row *ST*, we can know that RDOA only spends 1.21 seconds to get normalized total square error J smaller than 0.0078125 or get into a steady state. RDOA is 2.21 times faster than DOA (2.68sec). The result of using BP algorithm with fixed learning rate 0.1 to train the IRIS is shown Figure 4-11. The result of using DO algorithm with initial weight drawn from a uniform distribution over interval [-1, 1] to train the IRIS is shown in Figure 4-12. The result of using RDO algorithm to train IRIS is shown in Figure 4-13. The comparison of the results of three algorithms is shown in Figure 4-14-1 and Figure 4-14-2.

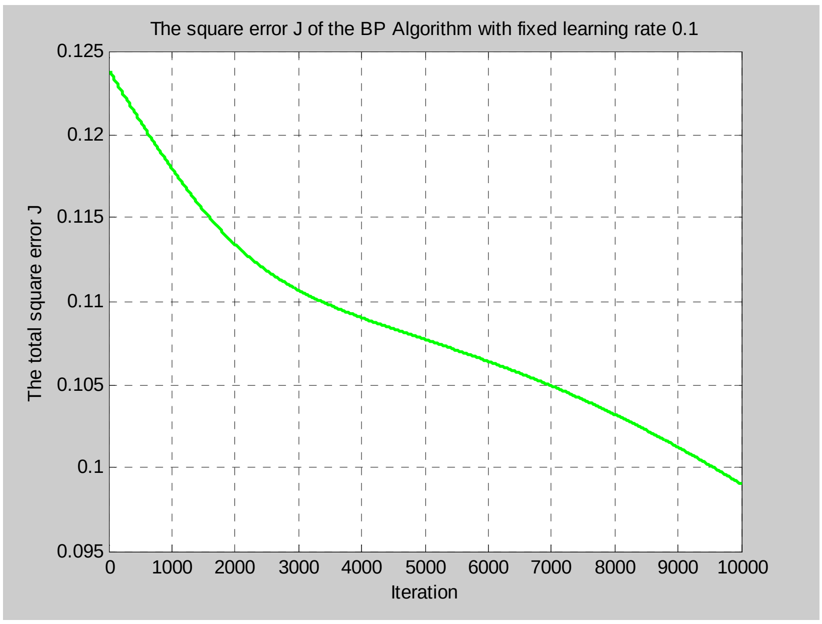


Figure 4-11. Normalized square error *J* of the standard BPA with fixed η = 0.1

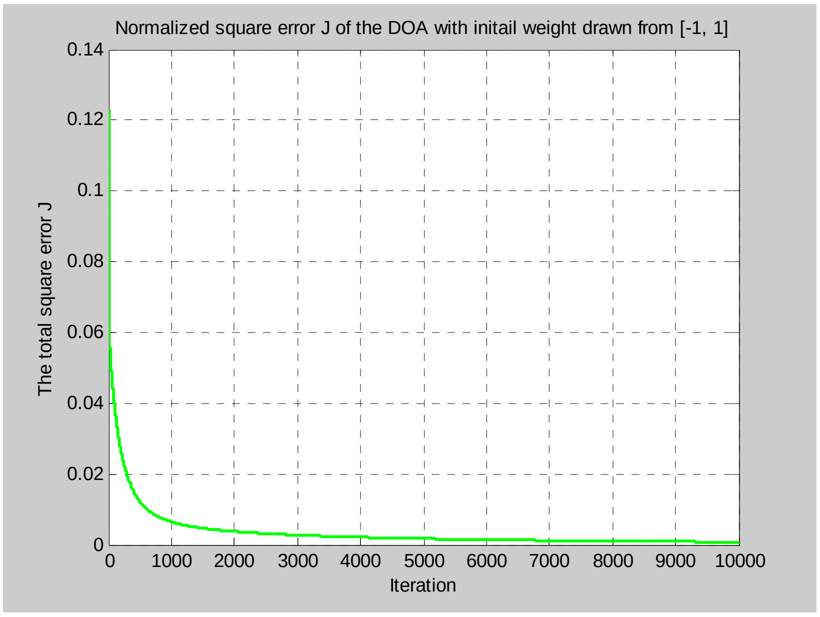


Figure 4-12. The normalized square error *J* of the DOA

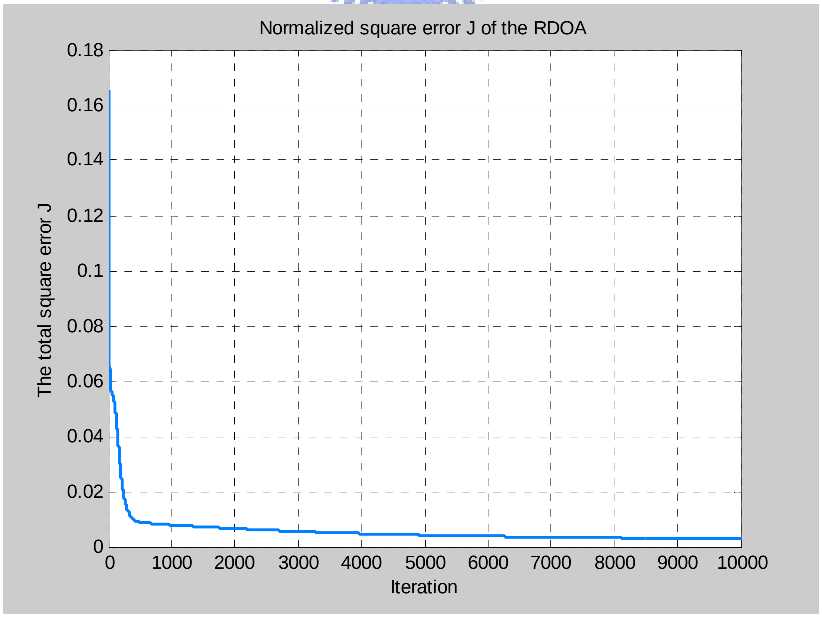


Figure 4-13. The normalized square error *J* of the RDOA.

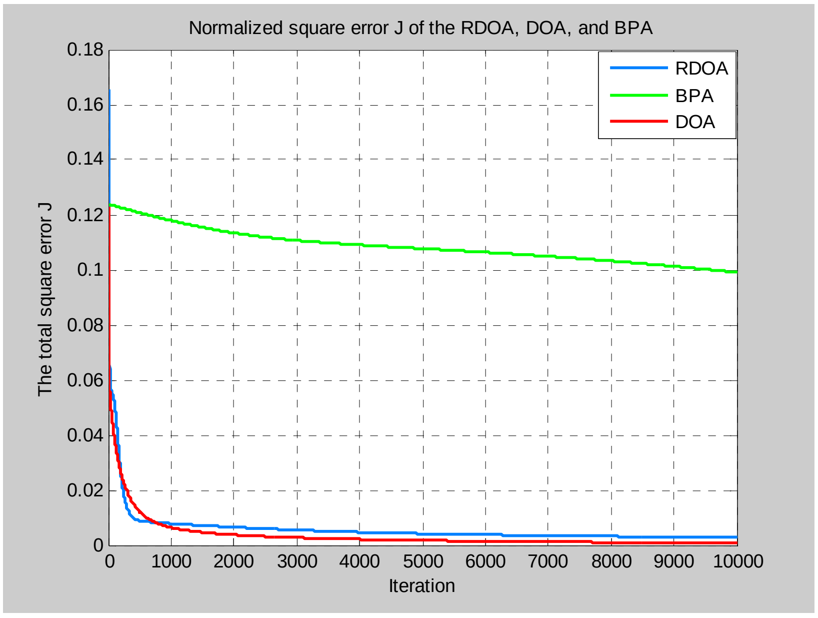


Figure 4-14-1. Training error of RDOA, DOA, and BPA

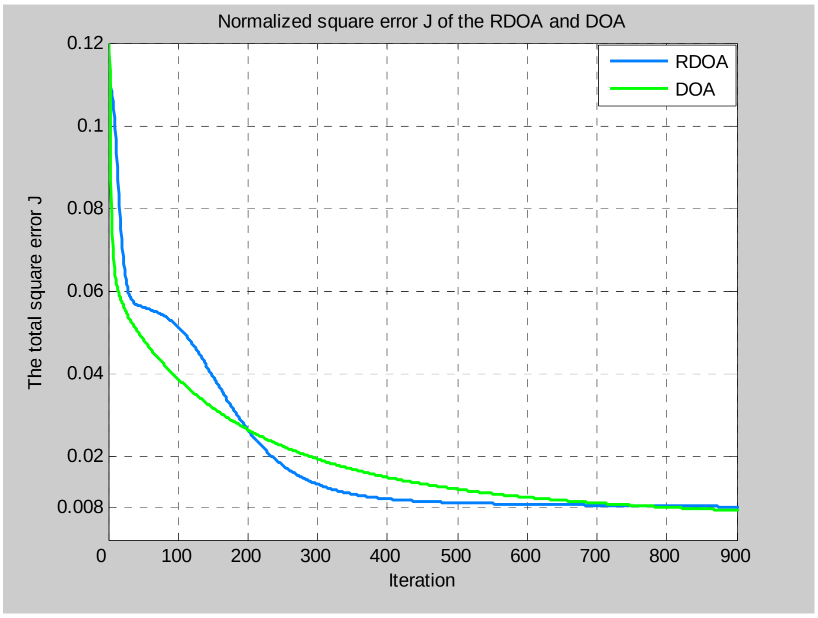


Figure 4-14-2. The close look of training error for Iris problem.

In Figure 4-14-1, DOA has a slightly better performance than that RDOA has at the end of training process. Both of them have acceptable performance at the end of training process. In Figure 4-14-2, it is obvious that RDOA and DOA both achieve the minimum acceptable square error 0.0078125 around the 800th iteration. RDOA has a faster convergent rate at very beginning of training process and the convergent rate of RDOA slow down when network gets into a steady state. Tables 4-8 ~ 4-10 show the complete training results of three algorithms for Iris classification problem. F in the column C denotes failed result of convergence. S in the column C denotes successful result of convergence.

Table 4-8 Detailed Training results of Iris problems for a 4-4-3 neural network with Back-propagation algorithm.

	Iris_BPA with fixed learning rate 0.01(first 10 trials) and 0.1 (rest 10 trials)			
	J (total square error)	T (cost time) sec	E (convergent iteration)	C (convergence)
1	0.1007000	2.656	N/A	F
2	0.1092000	2.625	N/A	F
3	0.1072000	2.938	N/A	F
4	0.1089000	3.031	N/A	F
5	0.1098000	3.125	N/A	F
6	0.1088000	3.078	N/A	F
7	0.0991000	3.141	N/A	F
8	0.1085000	3.093	N/A	F
9	0.1072000	3.297	N/A	F
10	0.1087000	3.063	N/A	F
11	0.0272000	3.235	N/A	F
12	0.0266000	3.156	N/A	F
13	0.0436000	3.078	N/A	F
14	0.0411000	3.094	N/A	F
15	0.0387000	3.109	N/A	F
16	0.0324000	3.157	N/A	F
17	0.0614000	3.235	N/A	F
18	0.0982000	3.094	N/A	F
19	0.0509000	3.172	N/A	F
20	0.0301000	3.203	N/A	F
Average	**0.0759150**	**3.079**	**N/A**	**0%**

Table 4-9 Detailed Training results of Iris problems for a 4-4-3 neural network with Dynamic Optimal Training algorithm

	Iris_Dynamic Optimal Training Algorithm			
	J (total square error)	T (cost time) sec	E (convergent iteration)	C (convergence)
1	0.1250000	76.156	N/A	**F**
2	0.1111000	50.922	N/A	**F**
3	0.0833000	81.469	N/A	**F**
4	0.0005039	40.985	615	**S**
5	0.0404000	44.735	N/A	**F**
6	0.1250000	76.594	N/A	**F**
7	0.1389000	41.516	N/A	**F**
8	0.0556000	38.828	N/A	**F**
9	0.0004618	41.453	661	**S**
10	0.1296000	76.625	N/A	**F**
11	0.1296000	49.656	N/A	**F**
12	0.1250000	76.625	N/A	**F**
13	0.0004626	40.782	569	**S**
14	0.1111000	62.031	N/A	**F**
15	0.0833000	81.703	N/A	**F**
16	0.0004516	41.172	771	**S**
17	0.0833000	81.969	N/A	**F**
18	0.1296000	67.359	N/A	**F**
19	0.0574000	44.719	N/A	**F**
20	0.0556000	49.875	N/A	**F**
Average	**0.0006266**	**41.098**	**654**	**20%**

Table 4-10 Detailed Training results of Iris problems for a 4-4-3 neural network with Revised Dynamic Optimal Training algorithm.

	Iris_Revised Dynamic Optimal Training Algorithm			
	J (total square error)	T (cost time) sec	E (convergent iteration)	C (convergence)
1	0.0037000	22.875	353	S
2	0.0043000	19.265	442	S
3	0.0039000	24	345	S
4	0.0027000	22.203	489	S
5	0.0044000	18.781	447	S
6	0.0035000	23.312	456	S
7	0.0039000	18.797	576	S
8	0.0037000	32.25	507	S
9	0.0048000	22.36	353	S
10	0.0046000	22.34	695	S
11	0.0040000	21.078	759	S
12	0.0041000	25.828	538	S
13	0.0042000	24.86	560	S
14	0.0028000	21.578	496	S
15	0.0033000	22.672	891	S
16	0.0556000	34.156	N/A	F
17	0.0029000	22.547	374	S
18	0.0039000	21.343	429	S
19	0.0042000	24.453	832	S
20	0.0040000	24.047	503	S
Average	**0.0038368**	**22.873105**	**528.68421**	**95%**

From Table 4-10, we choose one successful convergent result for the test of real classification performance. After 10000 training iterations, the resulting weighting factors and total square error *J* are shown below. The actual output and desired output of 10000 training iteration are shown in Table 4-11 and the testing output and desired output are shown in Table 4-12. After we substitute the above weighting matrices into the network to perform real testing, we find that there is no classification error by using training set (the first 75 data set). However there are 4 classification errors by using testing set (the later 75 data set), which are index 34, 55, 57, 59 in Table 4-12. This is better than that of using DO [19], which generate 5 classification errors.

$$W_H = \begin{bmatrix} -2.9855 & -2.5472 & 3.5321 & 4.9206 \\ 0.7139 & 0.8761 & -1.7383 & -1.4481 \\ -0.2861 & 0.4348 & -0.1717 & -0.5248 \\ 0.6269 & 0.1752 & 0.5665 & 0.1341 \end{bmatrix}$$

$$W_Y = \begin{bmatrix} 0.0874 & 1.1824 & -0.1976 & -0.0696 \\ -1.1929 & -1.6309 & 1.3376 & 1.0304 \\ 1.1153 & 0.3748 & -0.8871 & 0.0136 \end{bmatrix}$$

Normalized total square error $\boldsymbol{J} = 0.043$

Table 4-11. Actual and desired outputs after 10000 training iterations

Index	Actual Output			Desired Output		
	Class 1	**Class 2**	**Class 3**	**Class 1**	**Class 2**	**Class 3**
1	1.0069	-0.0060	0.0003	1.0000	0.0000	0.0000
2	0.9983	-0.0374	0.0295	1.0000	0.0000	0.0000
3	0.9976	0.0033	-0.0030	1.0000	0.0000	0.0000
4	0.9803	0.0148	0.0014	1.0000	0.0000	0.0000
5	1.0038	0.0168	-0.0153	1.0000	0.0000	0.0000
6	1.0041	-0.0166	0.0133	1.0000	0.0000	0.0000
7	0.9881	0.0290	-0.0153	1.0000	0.0000	0.0000
8	1.0001	-0.0052	0.0047	1.0000	0.0000	0.0000
9	0.9736	0.0178	0.0017	1.0000	0.0000	0.0000
10	0.9966	-0.0143	0.0134	1.0000	0.0000	0.0000
11	1.0116	-0.0145	0.0052	1.0000	0.0000	0.0000
12	0.9875	0.0220	-0.0075	1.0000	0.0000	0.0000
13	0.9971	-0.0160	0.0130	1.0000	0.0000	0.0000
14	0.9934	0.0327	-0.0278	1.0000	0.0000	0.0000
15	1.0197	-0.0072	-0.0059	1.0000	0.0000	0.0000
16	1.0132	0.0151	-0.0173	1.0000	0.0000	0.0000

17	1.0145	-0.0145	0.0028	1.0000	0.0000	0.0000
18	1.0061	-0.0185	0.0107	1.0000	0.0000	0.0000
19	1.0141	-0.0493	0.0318	1.0000	0.0000	0.0000
20	1.0026	0.0168	-0.0133	1.0000	0.0000	0.0000
21	1.0052	-0.0475	0.0359	1.0000	0.0000	0.0000
22	1.0015	-0.0067	0.0058	1.0000	0.0000	0.0000
23	1.0022	0.0618	-0.0536	1.0000	0.0000	0.0000
24	0.9779	-0.0397	0.0494	1.0000	0.0000	0.0000
25	0.9636	0.0430	-0.0046	1.0000	0.0000	0.0000
26	0.0088	1.0283	-0.0430	0.0000	1.0000	0.0000
27	-0.0115	1.0568	-0.0463	0.0000	1.0000	0.0000
28	-0.0329	1.0571	-0.0293	0.0000	1.0000	0.0000
29	-0.0194	0.9345	0.0867	0.0000	1.0000	0.0000
30	-0.0322	1.0271	0.0009	0.0000	1.0000	0.0000
31	-0.0371	0.9821	0.0604	0.0000	1.0000	0.0000
32	-0.0365	1.0444	-0.0069	0.0000	1.0000	0.0000
33	0.0958	0.9842	-0.0740	0.0000	1.0000	0.0000
34	-0.0104	1.0518	-0.0449	0.0000	1.0000	0.0000
35	-0.0140	0.9531	0.0685	0.0000	1.0000	0.0000
36	0.0219	1.0018	-0.0202	0.0000	1.0000	0.0000
37	-0.0122	1.0405	-0.0265	0.0000	1.0000	0.0000
38	0.0263	1.0146	-0.0444	0.0000	1.0000	0.0000
39	-0.0412	0.9983	0.0444	0.0000	1.0000	0.0000
40	0.0974	0.9636	-0.0598	0.0000	1.0000	0.0000
41	0.0289	1.0098	-0.0438	0.0000	1.0000	0.0000
42	-0.0348	0.8541	0.1881	0.0000	1.0000	0.0000
43	0.0336	1.0402	-0.0705	0.0000	1.0000	0.0000
44	-0.0298	0.7413	0.2845	0.0000	1.0000	0.0000
45	0.0229	1.0356	-0.0562	0.0000	1.0000	0.0000
46	-0.0244	0.5388	0.4914	0.0000	1.0000	0.0000
47	0.0429	1.0043	-0.0495	0.0000	1.0000	0.0000
48	-0.0324	0.6639	0.3668	0.0000	1.0000	0.0000
49	-0.0381	1.0577	-0.0176	0.0000	1.0000	0.0000
50	0.0203	1.0256	-0.0489	0.0000	1.0000	0.0000
51	0.0068	-0.0792	1.0720	0.0000	0.0000	1.0000
52	0.0043	-0.0094	1.0096	0.0000	0.0000	1.0000
53	0.0035	0.0104	0.9829	0.0000	0.0000	1.0000

54	0.0009	0.0310	0.9709	0.0000	0.0000	1.0000
55	0.0071	-0.0666	1.0589	0.0000	0.0000	1.0000
56	0.0081	-0.0659	1.0524	0.0000	0.0000	1.0000
57	0.0025	0.0255	0.9840	0.0000	0.0000	1.0000
58	-0.0005	0.0567	0.9402	0.0000	0.0000	1.0000
59	0.0058	-0.0313	1.0236	0.0000	0.0000	1.0000
60	0.0067	-0.0466	1.0372	0.0000	0.0000	1.0000
61	-0.0204	0.4357	0.5845	0.0000	0.0000	1.0000
62	0.0017	0.0540	0.9446	0.0000	0.0000	1.0000
63	0.0000	0.0795	0.9187	0.0000	0.0000	1.0000
64	0.0065	-0.0461	1.0426	0.0000	0.0000	1.0000
65	0.0072	-0.0685	1.0624	0.0000	0.0000	1.0000
66	0.0052	-0.0062	1.0020	0.0000	0.0000	1.0000
67	-0.0089	0.2004	0.8098	0.0000	0.0000	1.0000
68	-0.0022	0.0721	0.9265	0.0000	0.0000	1.0000
69	0.0122	-0.1202	1.0997	0.0000	0.0000	1.0000
70	-0.0030	0.1452	0.8597	0.0000	0.0000	1.0000
71	0.0054	-0.0152	1.0078	0.0000	0.0000	1.0000
72	0.0045	-0.0077	1.0092	0.0000	0.0000	1.0000
73	0.0087	-0.0727	1.0577	0.0000	0.0000	1.0000
74	-0.0154	0.3805	0.6345	0.0000	0.0000	1.0000
75	-0.0002	0.0598	0.9410	0.0000	0.0000	1.0000

Table 4-12. Actual and desired outputs in real testing

Index	Actual Output			Desired Output		
	Class 1	**Class 2**	**Class 3**	**Class 1**	**Class 2**	**Class 3**
1	0.9895	-0.0394	0.0390	1.0000	0.0000	0.0000
2	0.9890	-0.0205	0.0254	1.0000	0.0000	0.0000
3	1.0067	-0.0167	0.0096	1.0000	0.0000	0.0000
4	1.0100	-0.0286	0.0158	1.0000	0.0000	0.0000
5	0.9795	0.0168	0.0017	1.0000	0.0000	0.0000
6	0.9821	-0.0049	0.0169	1.0000	0.0000	0.0000
7	1.0102	-0.0727	0.0511	1.0000	0.0000	0.0000
8	1.0029	0.0720	-0.0556	1.0000	0.0000	0.0000
9	1.0101	0.0424	-0.0373	1.0000	0.0000	0.0000

10	0.9936	-0.0236	0.0228	1.0000	0.0000	0.0000
11	1.0106	-0.0309	0.0150	1.0000	0.0000	0.0000
12	1.0193	-0.0468	0.0239	1.0000	0.0000	0.0000
13	1.0013	0.0409	-0.0327	1.0000	0.0000	0.0000
14	0.9836	0.0191	-0.0073	1.0000	0.0000	0.0000
15	1.0035	-0.0167	0.0115	1.0000	0.0000	0.0000
16	1.0064	-0.0080	0.0015	1.0000	0.0000	0.0000
17	0.9601	-0.0460	0.0601	1.0000	0.0000	0.0000
18	0.9864	0.0387	-0.0241	1.0000	0.0000	0.0000
19	0.9819	-0.0258	0.0352	1.0000	0.0000	0.0000
20	0.9797	0.0186	0.0040	1.0000	0.0000	0.0000
21	0.9906	-0.0335	0.0317	1.0000	0.0000	0.0000
22	1.0002	0.0298	-0.0212	1.0000	0.0000	0.0000
23	0.9887	0.0185	-0.0081	1.0000	0.0000	0.0000
24	1.0090	-0.0039	-0.0015	1.0000	0.0000	0.0000
25	1.0039	-0.0179	0.0111	1.0000	0.0000	0.0000
26	0.0133	1.0264	-0.0442	0.0000	1.0000	0.0000
27	-0.0327	1.0483	-0.0214	0.0000	1.0000	0.0000
28	-0.0448	0.9100	0.1304	0.0000	1.0000	0.0000
29	-0.0350	0.9818	0.0546	0.0000	1.0000	0.0000
30	0.1751	0.8639	-0.0424	0.0000	1.0000	0.0000
31	0.0249	1.0284	-0.0512	0.0000	1.0000	0.0000
32	0.0612	1.0022	-0.0619	0.0000	1.0000	0.0000
33	0.0426	1.0154	-0.0574	0.0000	1.0000	0.0000
＊34	***-0.0102***	***0.2448***	***0.7696***	***0.0000***	***1.0000***	***0.0000***
35	-0.0300	0.7176	0.3229	0.0000	1.0000	0.0000
36	-0.0284	1.0562	-0.0232	0.0000	1.0000	0.0000
37	-0.0232	1.0546	-0.0354	0.0000	1.0000	0.0000
38	-0.0281	1.0090	0.0141	0.0000	1.0000	0.0000
39	0.0020	1.0648	-0.0603	0.0000	1.0000	0.0000
40	-0.0149	0.9960	0.0220	0.0000	1.0000	0.0000
41	-0.0351	0.9481	0.0936	0.0000	1.0000	0.0000
42	-0.0345	1.0454	-0.0091	0.0000	1.0000	0.0000
43	0.0157	1.0352	-0.0503	0.0000	1.0000	0.0000
44	0.0952	0.9744	-0.0660	0.0000	1.0000	0.0000
45	-0.0228	1.0192	0.0081	0.0000	1.0000	0.0000
46	0.0048	1.0731	-0.0714	0.0000	1.0000	0.0000

47	-0.0109	1.0595	-0.0437	0.0000	1.0000	0.0000
48	0.0058	1.0457	-0.0522	0.0000	1.0000	0.0000
49	0.2174	0.8265	-0.0462	0.0000	1.0000	0.0000
50	-0.0036	1.0498	-0.0426	0.0000	1.0000	0.0000
51	-0.0231	0.3941	0.6255	0.0000	0.0000	1.0000
52	-0.0200	0.4748	0.5455	0.0000	0.0000	1.0000
53	-0.0199	0.4459	0.5768	0.0000	0.0000	1.0000
54	0.0071	-0.0578	1.0504	0.0000	0.0000	1.0000
＊55	***-0.0445***	***0.7323***	***0.3064***	***0.0000***	***0.0000***	***1.0000***
56	-0.0017	0.0873	0.9093	0.0000	0.0000	1.0000
＊57	***-0.0472***	***0.7346***	***0.3045***	***0.0000***	***0.0000***	***1.0000***
58	0.0078	-0.0705	1.0617	0.0000	0.0000	1.0000
＊59	***-0.0369***	***0.6932***	***0.3441***	***0.0000***	***0.0000***	***1.0000***
60	-0.0051	0.1123	0.8975	0.0000	0.0000	1.0000
61	0.0048	0.0028	0.9858	0.0000	0.0000	1.0000
62	0.0059	-0.0518	1.0478	0.0000	0.0000	1.0000
63	-0.0092	0.1984	0.8135	0.0000	0.0000	1.0000
64	-0.0199	0.4646	0.5589	0.0000	0.0000	1.0000
65	-0.0105	0.2538	0.7538	0.0000	0.0000	1.0000
66	0.0081	-0.0618	1.0517	0.0000	0.0000	1.0000
67	-0.0087	0.2647	0.7399	0.0000	0.0000	1.0000
68	0.0043	-0.0094	1.0096	0.0000	0.0000	1.0000
69	0.0072	-0.0599	1.0511	0.0000	0.0000	1.0000
70	0.0080	-0.0679	1.0583	0.0000	0.0000	1.0000
71	0.0037	0.0463	0.9480	0.0000	0.0000	1.0000
72	0.0018	0.0824	0.9154	0.0000	0.0000	1.0000
73	-0.0071	0.2081	0.7990	0.0000	0.0000	1.0000
74	0.0043	-0.0169	1.0162	0.0000	0.0000	1.0000
75	-0.0049	0.1520	0.8593	0.0000	0.0000	1.0000

CHAPTER 5

Conclusions

In this thesis, a revised dynamic optimal training algorithm (RDOA) for modified three-layer neural network has been proposed. The RDOA includes three modifications proposed in Chapter 3. The three modifications are: "*Simplification of activation function*", "*Selection of proper initial weighting factors*", and "*Determination of upper-bound of learning rate in each iteration*". Simplification of activation function will enhance the back-propagated error signal, and hence improves the convergence rate of dynamic optimal training algorithm (DOA). By finding proper initial weighting factors, the probability of escaping local minima will be increased. Also the finding of the upper-bound of stable learning rate can guarantee the convergence of training process and this will speed up the search of optimal learning rate. The classification problems of XOR and Iris data are proposed in Chapter 4. They are solved by using the revised dynamical optimal training for a modified three-layer neural network with sigmoid activation functions in hidden layer and linear activation function in output layer. Excellent results have obtained for XOR problem and Iris data problem, which indicate that the RDOA is considerably faster than DOA and BPA. In addition, the RDOA is easier to find global convergent results than those by using DOA and BPA. So RDOA can speed up convergence rate and has a higher chance of escaping local minima than the chance of DOA.

REFERENCES

[1] T. Yoshida, and S. Omatu, “Neural network approach to land cover mapping,” *IEEE Trans. Geoscience and Remote,* Vol. 32, pp. 1103-1109, Sept. 1994.

[2] H. Bischof, W. Schneider, and A. J. Pinz, “Multispectral classification of Landsat-images using neural networks,” *IEEE Trans, Gsoscience and Remote Sensing,* Vol. 30, pp. 482-490, May 1992.

[3] M. Gopal, L. Behera, and S. Choudhury, “On adaptive trajectory tracking of a robot manipulator using inversion of its neural emulator,” *IEEE Trans. Neural Networks*, 1996.

[4] L. Behera, “Query based model learning and stable tracking of a roboot arm using radial basis function network,” *Elsevier Science LTd., Computers and Electrical Engineering*, 2003.

[5] F. Amini, H. M. Chen, G. Z. Qi, and J. C. S. Yang, “Generalized neural network based model for structural dynamic identification, analytical and experimental studies,” *Intelligent Information Systems*, Proceedings 8-10, pp. 138-142, Dec. 1997.

[6] K. S. Narendra, and S. Mukhopadhyay, “Intelligent control using neural networks,” *IEEE Trans., Control Systems Magazine*, Vol. 12, Issue 2, pp.11-18, April 1992.

[7] L. Yinghua, and G. A. Cunningham, “A new approach to fuzzy-neural system modeling,” *IEEE Trans., Fuzzy Systems*, Vol. 3, pp. 190-198, May 1995.

[8] L. J. Zhang, and W. B. Wang, “Scattering signal extracting using system modeling method based on a back propagation neural network,” *Antennas and Propagation Society International Symposium*, 1992. AP-S, 1992 Digest. Held in

Conjuction with: URSI Radio Science Meting and Nuclear EMP Meeting, IEEE 18-25, Vol. 4, pp. 2272, July 1992

[9] P. Poddar, and K. P. Unnikrishnan, "Nonlinear prediction of speech signals using memory neuron networks," *Neural Networks for Signal Processing* [1991], Proceedings of the 1991 IEEE Workshop 30, pp. 395-404, Oct. 1991.

[10] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, Vol. 65, pp.386-408, 1958.

[11] F. Rosenblatt, "Principles of Neurodynamics", Spartan books, New York, 1962.

[12] R. P. Lippmann, "An introduction to computing with neural networks," *IEEE ASSP Magazine*, 1987.

[13] D. E. Rumelhart et al., Learning representations by back propagating error," *Nature*, Vol. 323, pp. 533-536, 1986

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing, Exploration in the Microstructure of Cognition*, Vol. 1, D. E. Rumelhart and J. L. McClelland, eds. Cambridge, MA: MIT Press, 1986.

[15] K. C. Tan, and H. J. Tang, "New Dynamical Optimal Learning for Linear Multilayer FNN," *IEEE Trans. Neural Networks*, Vol. 15, No. 6, pp. 1562-1568, Nov. 2004.

[16] S. C. Ng, S. H. Leung, and A. Luk, "The Generalized Back-Propagation Algorithm with Convergence Analysis," *Circuits and Systems, 1999. ISCAS '99, Proceedings, 1999 IEEE International Symposium*, Vol. 5, pp. 612-615, 30 May-2 June 1999.

[17] H. Lari-Najafi, M. Nasiruddin, and T. Samad, "Effect of initial weights on Back-Propagation and its variations," *Syst., Man, Cybern., Proceedings of international Conference*, Vol. 1, pp. 218-219, 14-17 Nov 1989.

[18] C. H. Wang, H. L. Liu, and C. T. Lin, “Dynamic Optimal Learning Rates of a Certain Class of Fuzzy Neural Networks and its Applications with Genetic Algorithm,” *IEEE Trans. Syst., Man, Cybern. Part* B, Vol. 31, pp. 467-475, June 2001.

[19] Y. Y. Chi, and C. H. Wang, “Dynamic Optimal Training of A Three-Layer Neural Network with Sigmoid Function,” *thesis of master degree, National Chiao Tung University*, June 2005.

[20] J. J. F. Cerqueira, A. G. B. Palhares, and M. K. Madrid, “A Complement to the Back-Propagation Algorithm: An Upper Bound for the Learning Rate,” *IEEE Trans. Neural Networks*, *Proceedings of the IEEE-INNS-ENNS international Conference*, Vol. 4, pp. 517-522, 24-27 July 2000.

[21] J. J. F. Cerqueira, A. G. B. Palhares, and M. K. Madrid, “A Simple adaptive Back-Propagation Algorithm for Multilayered Feedforward Perceptrons,” *IEEE Trans. Syst., Man, Cybern., 2002 IEEE international Conference*, Vol. 3, Oct 2002.

[22] L. Behera, S. Kumar, and A. Patnaik, “A novel learning algorithm for feedforeward networks using Lyapunov function approach,” *Intelligent Sensing and Information Processing*, *Proceedings of international Conference*, pp. 277-282, 2004.

[23] M. A. AL-Alaoui, R. Mouci, M. M. Mansour, and R. Ferzli, “A Cloning Approach to Classifier Training,” *IEEE Trans. Syst., Man, Cybern. Part* A, Vol. 32, pp. 746-752, Nov. 2002.

[24] R. Kozma, M. Kitamura, A. Malinowski, and J. M. Zurada, “On performance measures of artificial neural networks trained by structural learning algorithms,” *Artificial Neural Networks and Expert Systmes*, *Proceedings, Second New Zealand International Two-Stream Conference*, pp.22-25, Nov. 20-23, 1995.

[25] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. 1, 3rd edition, New York: John Wiley, 1968

[26] B. Widrow, and Jr. M. E. Hoff, *"Adaptive switching circuits," IRE WCSEON Convention Record*, pp. 96-104, 1960

[27] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, 1976

[28] Iris Data Samples [Online]. Available: **ftp.ics.uci.edu/pub/machine-learning data- bases/iris/iris.data.**

APPENDIX A

In Section 3.1, we have proposed an unbounded linear activation function in the NN training process. In this section, we will explain the reason why it is not suitable to use bounded linear activation function in the NN training process.

Defining an unbounded linear function as:

$$\varphi(x) = ax$$

Defining a saturated linear function as

$$\varphi_b(x) = \begin{cases} -u_1, & \text{if } x \in \left(-\infty, -u_1\right] \\ ax, & \text{if } x \in (-u_1, u_2) \\ u_2, & \text{if } x \in \left[u_2, \infty\right) \end{cases}$$

For u_1, $u_2 > 0$, lower-bound and upper-bound are donated as u_1 and u_2, respectively. a is an adjustable parameter which means slope of the function. Usually, we set parameter $a = 1$. It is a piece-wise continuous linear function. Between u_1 and u_2, $\varphi_b(x)$ is the same as an unbounded linear activation function.

Consider update rule of weighting factors in output layer

$$W_{Y(t+1)}^{(p,r)} = W_{Y(t)}^{(p,r)} - \eta \frac{\partial J}{\partial V_O^{(p,:)}} \frac{\partial V_O^{(p,:)}}{\partial S_O^{(p,:)}} \frac{\partial S_O^{(p,:)}}{\partial W_Y^{(p,r)}}$$

We only change the activation function, so the presentation of $\partial J / \partial V_O^{(p,:)}$ and $\partial S_O^{(p,:)} / \partial W_Y^{(p,:)}$ is still the same as:

$$\frac{\partial J}{\partial V_O^{(p,:)}} = \frac{\partial}{\partial V_O^{(p,:)}} \left(\frac{1}{2PN} \sum_{p=1}^{P} \left(V_O^{(p,:)} - D^{(p,:)} \right)^2 \right) = \frac{1}{PN} \left(V_O^{(p,:)} - D^{(p,:)} \right) \qquad \text{(a.1)}$$

$$\frac{\partial S_O^{(p,:)}}{\partial W_Y^{(p,r)}} = \frac{\partial}{\partial W_Y^{(p,r)}} \left(\sum_{r=1}^{R} W_Y^{(p,r)} V_H^{(r,:)} \right) = V_H^{(r,:)} \qquad \text{(a.2)}$$

If $\varphi_O(x) = \varphi_b(x)$, then $\partial V_O^{(p,:)} / \partial S_O^{(p,:)}$ can be rewritten as

$$\frac{\partial V_O^{(p,:)}}{\partial S_O^{(p,:)}} = \frac{\partial}{\partial S_O^{(p,:)}}\left(\varphi_b(S_O^{(p,:)})\right) = \begin{cases} \partial(-u_1)/\partial S_O^{(p,:)} & = 0, \quad if \ S_O^{(p,:)} \in (-\infty, -u_1] \\ \partial(aS_O^{(p,:)})/\partial S_O^{(p,:)} & = a, \quad if \ S_O^{(p,:)} \in (-u_1, u_2) \\ \partial(u_2)/\partial S_O^{(p,:)} & = 0, \quad if \ S_O^{(p,:)} \in [u_2, \infty) \end{cases} \tag{a.3}$$

From (a.1), (a.2), and (a.3), we can rewrite update rule of output layer (2.42) as

$$W_{Y(t+1)}^{(p,r)} = \begin{cases} W_{Y(t)}^{(p,r)} - \eta \times 0, & if \ S_O^{(p,:)} \in (-\infty, -u_1] \\ W_{Y(t)}^{(p,r)} - \eta \dfrac{a}{PN}(V_O^{(p,:)} - D^{(p,:)})V_H^{(r,:)}, & if \ S_O^{(p,:)} \in (-u_1, u_2) \\ W_{Y(t)}^{(p,r)} - \eta \times 0, & if \ S_O^{(p,:)} \in [u_2, \infty) \end{cases} \tag{a.4}$$

Hence, the update rule of hidden layer (2.43) becomes to

$$W_{H(t+1)}^{(r,m)} = \begin{cases} W_{H(t)}^{(r,m)} - \eta \times 0, & if \quad S_O^{(p,:)} \in (-\infty, -u_1] \\ W_{H(t)}^{(r,m)} - \eta \left(\sum_{p=1}^{P} \dfrac{a}{PN}\left(V_O^{(p,:)} - D^{(p,:)}\right) W_Y^{(p,r)} \right) V_H^{(r,:)}\left(1 - V_H^{(r,:)}\right) X^{(m,:)}, & if \quad S_O^{(p,:)} \in (-u_1, u_2) \\ W_{H(t)}^{(r,m)} - \eta \times 0, & if \quad S_O^{(p,:)} \in [u_2, \infty) \end{cases} \tag{a.5}$$

From (a.4) & (a.5), we can discover that, if we use a bounded linear activation function, the weight matrix will not be updated in the saturation region of piece-wise linear function. In the saturation region, term $V_O^{(p,:)}$ is a constant, so its differential term is zero. Hence, the output of neural network will not change anymore and will make the network get into a steady state. Then the training process is stop. A narrow range of linear region will make system too early to enter a steady state; even total-square-error is still at a high level. A wide one is much better, but it still may make the system stop to update the weight matrix. Compare with sigmoid activation function, although it is also a saturated function, but its differential term is not zero in its saturation region. So the training process is capable of jumping out the local steady state which doesn't have an acceptable result.

In a word, we can not use a saturated linear activation function; because its fixed boundary lets the network lose the capability of online updating when the network once gets stuck into saturation region.

APPENDIX B

We consider the activation function at the first (hidden) layer. Assume that we use the unbounded activation function in both hidden and output layer. Without boundary, the output of the activation function can be any real value. During the training process of neural network, this unlimited output may cause that the weight becomes to an unreasonable large value which is not acceptable in the real condition. So, a saturated activation function is needed in the NN. Now we assume that a saturated activation function has been given in the NN.

From Appendix A and Section 3.2, we have already known that a saturated linear activation function is forbidden and second layer activation function has been corrected to a unbounded linear one. Hence, we can only put the saturated nonlinear activation function (ex: sigmoid function) in the hidden layer.