NEW APPROACHES TO SOLVE THE LOCAL MINIMUM PROBLEM AND IMPROVE THE CLASSIFICATION ABILITY OF LEARNING ALGORITHMS IN MULTI-LAYER FEED-FORWARD NEURAL NETWORKS

XU SHENSHENG

M.Phil

The Hong Kong Polytechnic University

2016

The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

New Approaches to Solve the Local Minimum Problem and Improve the Classification Ability of Learning Algorithms in Multi-Layer Feed-Forward Neural Networks

XU Shensheng

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Philosophy

July 2015

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

XU Shensheng

To the memory of my grandfathers, XU Mingqin and LI Peiyuan

Abstract:

This thesis proposes two new algorithms. They are Wrong Output Modification (WOM) and Threshold of Output Differences (TOD). WOM is used to solve the local minimum problem in training a multi-layer feed-forward network while TOD is used to improve the classification ability in training a multi-layer feed-forward network.

When a searching to find a global minimum is trapped by a local minimum, the change of weights could be zero or extremely small. Thus, the mean square error cannot be further reduced while its value is still so large that the searching cannot find the global minimum. In this circumstance, WOM locates the wrong output values and moves them closer to their corresponding target output values. Thus neuron weights are modified accordingly, and hence the searching can escape from such local minimum. WOM can be applied in different learning algorithms. Our performance investigation shows that learning with WOM can always escape from local minima and converge to a global minimum. Moreover, it obtains better classification ability after training.

TOD monitors the difference of each output value and its corresponding target output value. All differences are used to identify whether a searching finds a global minimum or not. TOD can be applied in different learning algorithms. Our performance investigation shows that by using TOD, a multi-layer feed-forward neural network can be trained in a better way so that its classification ability is better. This improvement is very significant if all features in testing data can be found in training data.

Publications

Some preliminary results and parts of this thesis have been published and have appeared in conference proceedings while some submitted papers are still under review.

- C. C. Cheung, Sean Shensheng Xu and S. C. Ng, “A Systematic Approach to Find a Global Solution in Training a Feed-Forward Neural Network”, will be submitted to *IEEE Transaction on Neural Networks and Learning Systems*, 2015.
- Sean Shensheng Xu, C. C. Cheung and S. C. Ng, “A new algorithm to speed up the convergence rate and the classification ability in Training a Feed-Forward Neural Networks”, will be submitted to *IJCNN 2016*, 2016.
- Sean Shensheng Xu and C. C. Cheung, “A New Terminating Condition to Identify the Convergence of the Learning Process in Multi-Layer Feed-Forward Neural Networks”, *Proceedings of IJCNN 2015*, Killarney, Ireland, July 2015.
- C. C. Cheung, S. C. Ng, A. K. Lui and Sean Shensheng Xu, "Further Enhancements in WOM Algorithm to Solve the Local Minimum and Flat-Spot Problem in Feed-Forward Neural Networks", *Proceedings of IJCNN 2014*, Beijing, China, July 2014.
- C. C. Cheung, S. C. Ng, A. K. Lui and Sean Shensheng Xu, "Solving the Local Minimum and Flat-Spot Problem by Modifying Wrong Outputs for Feed-Forward Neural Networks", *Proceedings of IJCNN 2013*, Dallas, TX, USA, August 2013.

Acknowledgements

I would like to take this opportunity to express my most sincere gratitude to my supervisor, Dr. Lawrence Chi-Chung Cheung, not only for the scientific guidance he has given to me throughout this research study, but also for his constant support in making some research projects possible during these years.

Dr. Cheung inspired my interest in the field of Artificial Intelligence, and he has guided me throughout the learning of Neural Networks. He has given me constructive advice and suggestions when I got stuck in some research problems. He also has given me a lot of guidance, assistance and encouragement in terms of thesis writing and oral presentation.

I would thank Dr. Vanessa Sin-Chun Ng in School of Science and Technology, The Open University of Hong Kong, for her ideas and resources throughout my research development. I also thank Dr. Rex G. Sharman for his proof-reading of this thesis.

Finally, I would like to thank my family for supporting me and giving me a lot of encouragement throughout the years. Without them, I would never have thought of completing my studies.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Artificial Neural Networks

Artificial Intelligence (AI) is a branch of computer science that is related to intelligent automation. John McCarthy [1], as the father of AI, coined the concept of intelligence in 1955 and defined it as "the science and engineering of making intelligent machines" [2] in 2007. Many Artificial Intelligence techniques have been proposed to develop different forms of AI. In soft computing [3], the most popular AI techniques are Artificial Neural Networks (ANNs), Fuzzy Logic (FL) and Evolutionary Algorithm (EA). This thesis focuses on ANNs.

ANNs [4] are biologically inspired networks that consist of processing neurons. The neurons are connected with each other and are capable of receiving and sending signals. The connections can be considered as a function of network weights, and the value of weights represents the strength of connections. The term artificial neural networks can be shortened to neural networks. The main contribution of neural networks is their ability to capture hidden information from known data, and this capturing process is called learning or training of neural networks. Neural networks have been successfully used in many applications, such as classification and clustering [5], pattern recognition [6], signal processing [7], clinical medicine [8], food science [9], chemical engineering [10], and energy systems [11], among others.

Three types of learning can be found in neural networks: Supervised Learning, Unsupervised Learning and Reinforcement Learning. Supervised learning means the

desired output of each input pattern is known. A neural network keeps adjusting weights according to the error signals, so that the actual output can approach its corresponding desired output. Unsupervised learning means the desired output is not known. Thus, no error signals exist to evaluate the actual output and the neural network just tries to get hidden information from the input data. Reinforcement learning can be seen between supervised learning and unsupervised learning. The learning process has to rely on trial-and-error interactions with a dynamic environment.

My research is concerned with multi-layer feed-forward neural networks in supervised learning, because supervised learning in multi-layer feed-forward neural networks is one of the most popular neural network applications. These applications are applied in a wide variety of fields, especially in chemistry related problems [12, 13].

1.2 Multi-Layer Feed-Forward Neural Networks

In a multi-layer feed-forward neural network, neurons are ordered into layers. The network consists of an input layer, one or more hidden layers and an output layer. Fig. 1.1 shows a fully connected feed-forward neural network with one hidden layer. My research is focused on this network structure.

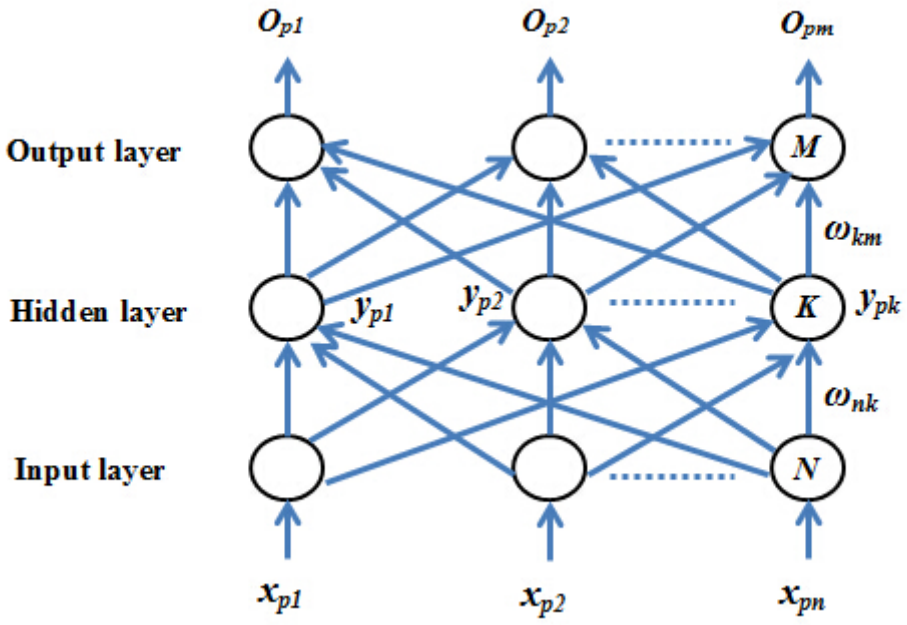

Fig. 1.1 A feed-forward neural network

There are N input neurons, K hidden neurons and M output neurons in the network. From input to output, each neuron (node) is connected to its neighbors. Let y_{pk} and o_{pm} be the outputs of the hidden node and the output node from the input pattern p (p = 1, 2…, P, where P is the number of input patterns) respectively. Moreover, x_{pn} is the input value in the input node n for the input pattern p, t_{pm} is its corresponding desired target which can be found in the training data set, $\overline{\omega}_{nk}$ is the network weight for the input node n and the hidden node k, and ω_{km} is the network weight for the hidden node k and the output node m. The system error of the network (E) at the i-th iteration is defined as [14]:

$$E(i) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M}[t_{pm} - o_{pm}(i)]^2 . \qquad (1.1)$$

Compared with E, the mean square error (*MSE*) is used to determine the convergence

of the learning. The value of E is expected to be not very small since it is used for the computation in neural networks (this computation will be described later in this chapter) and the computation will not be effective if E is too small. On the other hand, the mean square error is an indicator to determine whether the learning converges or not. Sometimes it may be very small when the learning is close to converge. Thus it should not be used in the computation. The definition of the mean square error at the i-th iteration is shown below:

$$MSE(i) = \frac{1}{PM} \sum_{p=1}^{P} \sum_{m=1}^{M} [t_{pm} - o_{pm}(i)]^2 \ . \tag{1.2}$$

1.3 Back-Propagation Algorithm

Back-Propagation (BP) [14] is the most popular supervised learning algorithm widely used in training multi-layer feed-forward neural networks. BP employs the gradient descent method [15] to minimize the mean square error by calculating the gradient of a loss function with respect to all weights in the network. Based on Fig. 1.1, the implementation of the standard BP algorithm is shown below:

1) Before Training: initialization and parameter setting.

The weights of the network need to be initialized. The learning rate μ and the momentum factor α are set to small positive values. The error threshold is set to a very small positive value (0.001 is used as the default error threshold in my thesis), and the iteration number i is equal to 0. The definitions of these parameters will be described later in this section.

2) Forward Pass: calculate the network output o_{pm} and the mean square error.

For an input pattern x_p ($x_p = [x_{p1}....x_{pN}]$),

$$y_{pk}(i) = f(\sum_{n=1}^{N} \bar{\omega}_{nk}(i) x_{pn}) \tag{1.3}$$

and $$o_{pm}(i) = f(\sum_{k=1}^{K} \omega_{km}(i) y_{pk}(i)). \tag{1.4}$$

Sigmoid functions are used as the activation functions for both the hidden and output layers. The mean square error is obtained from Equation (1.2). If it is less than the error threshold, the learning process is terminated and the convergence is met; otherwise, Backward Pass operation is processed. Note that the error threshold can be considered as a terminating condition in learning, and it will be discussed later.

3) Backward Pass: calculate weight-update $\Delta\omega_{km}$ and $\Delta\bar{\omega}_{nk}$ for the next iteration.

The partial derivative of *E(i)* with respect to the weights between hidden neuron and output neuron $\omega_{km}(i)$ can be expressed as:

$$\frac{\partial E(i)}{\partial \omega_{km}(i)} = \frac{\partial E(i)}{\partial o_{pm}(i)} \cdot \frac{\partial o_{pm}(i)}{\partial \omega_{km}(i)} \tag{1.5}$$

where

$$\frac{\partial E(i)}{\partial o_{pm}(i)} = t_{pm} - o_{pm}(i) \tag{1.6}$$

and $$\frac{\partial o_{pm}(i)}{\partial \omega_{km}(i)} = o_{pm}(i)(1 - o_{pm}(i)) y_{pk}(i). \tag{1.7}$$

The partial derivative of *E(i)* with respect to the weights between hidden neuron and input neuron $\omega_{nk}(i)$ can be expressed as:

$$\frac{\partial E(i)}{\partial \bar{\omega}_{nk}(i)} = \sum_{m=1}^{M} \frac{\partial E(i)}{\partial o_{pm}(i)} \cdot \frac{\partial o_{pm}(i)}{\partial y_{pk}(i)} \cdot \frac{\partial y_{pk}(i)}{\partial \bar{\omega}_{nk}(i)} \tag{1.8}$$

where

$$\frac{\partial o_{pm}(i)}{\partial y_{pk}(i)} = o_{pm}(1-o_{pm}(i))\omega_{km}(i) \tag{1.9}$$

and $$\frac{\partial y_{pk}(i)}{\partial \overline{\omega}_{nk}(i)} = y_{pk}(i)(1-y_{pk})x_{pn}. \tag{1.10}$$

The weight-update for the *(i*+1*)*-th iteration can be expressed as:

$$\begin{aligned}\Delta\omega_{km}(i+1) &= -\mu\frac{\partial E(i)}{\partial \omega_{km}(i)} + \alpha\Delta\omega_{km}(i) \\ &= \mu\cdot\textstyle\sum_{p=1}^{P}\delta_{pm}(i)\cdot y_{pk}(i) + \alpha\Delta\omega_{km}(i)\end{aligned} \tag{1.11}$$

and
$$\begin{aligned}\Delta\overline{\omega}_{nk}(i+1) &= -\mu\frac{\partial E(i)}{\partial \overline{\omega}_{nk}(i)} + \alpha\Delta\overline{\omega}_{nk}(i) \\ &= \mu\cdot\textstyle\sum_{p=1}^{P}\overline{\delta}_{pk}(i)\cdot x_{pn} + \alpha\Delta\overline{\omega}_{nk}(i)\end{aligned} \tag{1.12}$$

where

$$\delta_{pm}(i) = (t_{pm} - o_{pm}(i))o_{pm}(i)(1-o_{pm}(i)) \tag{1.13}$$

and $$\overline{\delta}_{pk}(i) = y_{pk}(i)(1-y_{pk}(i))\textstyle\sum_{m=1}^{M}\delta_{pm}(i)\omega_{km}(i). \tag{1.14}$$

So

$$\omega_{km}(i+1) = \omega_{km}(i) + \Delta\omega_{km}(i+1) \tag{1.15}$$

and $$\omega_{nk}(i+1) = \omega_{nk}(i) + \Delta\omega_{nk}(i+1). \tag{1.16}$$

The weights of the network have been updated. Then set $i = i + 1$ and process Forward Pass.

The main advantages of BP are its simplicity and low computational complexity. However, BP cannot overcome two limitations. First, the mean square error surface may have many local minima, and BP may converge to one of them since it calculates the gradient of a loss function. BP cannot escape from a local minimum once it is trapped, and therefore the learning process cannot converge to the global minimum. Second, the convergence rate of BP may be very slow because of the “flat-spot”

problem [16 – 19]. If BP is trapped into a flat-spot area, the change of weights becomes very slow so the change of the mean square error becomes very small. BP has to spend many iterations escaping from a flat-spot area, thereby leading to its slow convergence.

Fig. 1.2 shows the relationship between the mean square error (Vertical Axis) and the iterations (Horizontal Axis) when the BP algorithm is applied in the Five-Bit Counting problem. This Five-Bit Counting problem will be briefly described in Chapter 5. In Fig. 1.2, the mean square error drops quickly at the beginning: it is reduced from 0.3 to 0.015 within 200 iterations. BP can dramatically reduce the mean square error in this problem. However, the convergence rate is slow after the first 200 iterations, with hardly any changes after 1000 iterations because the learning process is trapped (a) into a flat-spot area, and then (b) by a local minimum. The change of the mean square error is shown in Fig. 1.3 by adjusting the scale of Vertical and Horizontal Axis in Fig. 1.2. Since the nature of these local minimum and flat-spot problems are very similar and later our proposed algorithm can solve both problems, for simplicity, these two problems are combined into one and it is called local minimum problem in rest of the thesis.

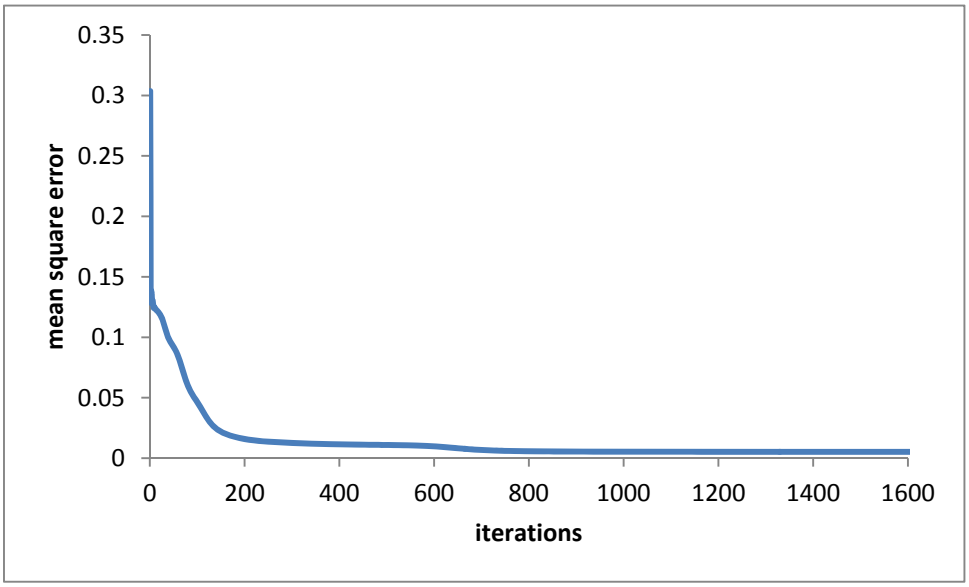

Fig. 1.2 BP is applied in the Five-Bit Counting learning problem

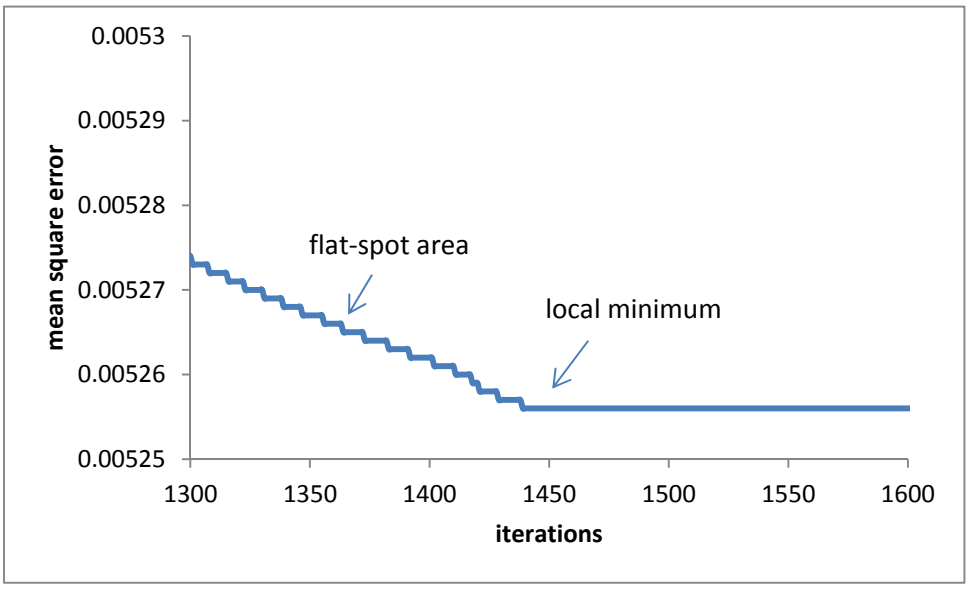

Fig. 1.3 Flat-spot area and local minimum

1.4 Terminating Condition and Misclassification Rate

The initialization of the training of a neural network generates a set of initial weights and sets the error threshold at 0.001. Note that error threshold can be seen as

the terminating condition of learning. Further investigation on this terminating condition does not occur because it is a simple and direct way to estimate the differences between outputs and targets. When the mean square error is less than the error threshold, that means the terminating condition is satisfied, the learning process is completed successfully and the weights of neural network are trained well. It is expected that this trained network should classify all training data correctly.

Using validation sets is another way to terminate the learning and there are two approaches to using them. One is called early stopping. It divides the training data into two sets: for training and validating. The validation error is calculated periodically during training. Training is terminated when the validation error begins to go up (see Fig. 1.4). Early stopping is fast and is the most common way for avoiding error over-fitting (i.e., a neural network trains in too many iterations so that the classification ability of the neural network after training is worse than the optimum one). The other approach is called cross-validation. It divides the data into k subsets of equal size. The network is trained k times and one subset is used each time for validation. Training is terminated when the subset with the smallest validation error is found. The biggest problem with using validation sets is how to split the training data, especially in cases where the size of training data may not be large enough. The split sample cannot contain all the features of the entire data for both training and validation. Moreover, using validation sets as the terminating condition cannot tell whether the learning is trapped by a local minimum or not.

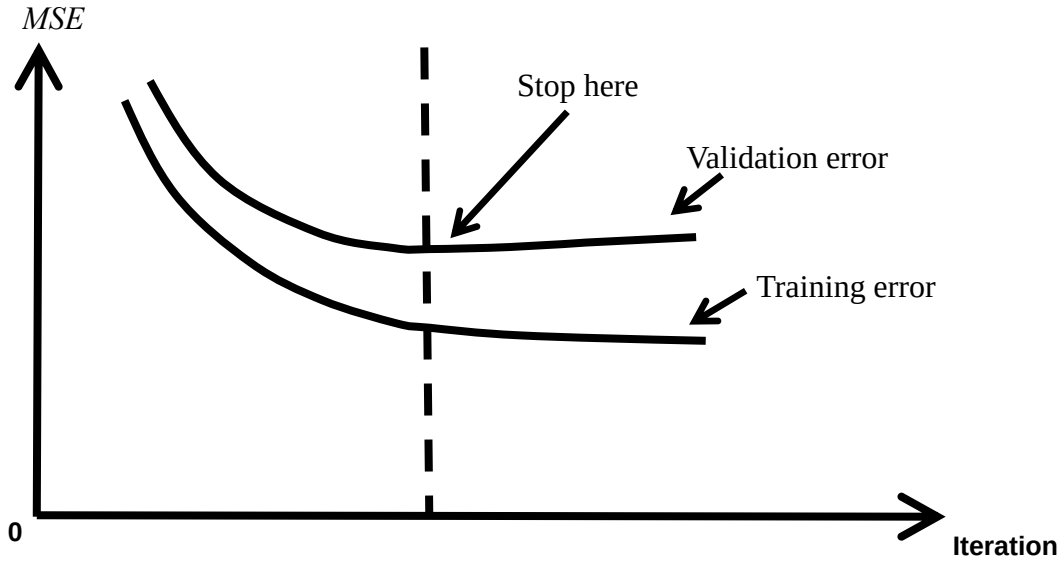

Fig. 1.4 Early stopping

1.5 Thesis Organization

The thesis is organized as follows: Chapter 2 introduces some learning algorithms dealing with aspects of convergence rate and convergence capability to overcome the limitations of the standard BP algorithm. Chapter 3 describes a new algorithm called Wrong Output Modification (WOM) which is used to solve the local minimum problem. Chapter 4 describes another new algorithm called Threshold of Output Difference (TOD) which is used to improve the classification ability. Chapter 5 shows the performance of the proposed algorithms through simulation results and performance comparisons with different learning algorithms in different learning problems. Conclusions are drawn in Chapter 6.

Chapter 2 Background

This thesis proposes (a) a new algorithm to solve the local minimum problem and (b) a new terminating condition to improve the classification capability of a learning algorithm. They can be applied to different learning algorithms in different learning problems. This Chapter briefly describes some popular existing learning algorithms for reference, and also describes some existing algorithms that have been proposed to solve the local minimum problem.

2.1 Other Learning Algorithms

Back-propagation (BP) is simple to apply in different learning problems but it is sometimes slow and easily trapped by a local minimum or into a flat-spot area. Many modifications have been proposed to speed up the learning process or improve the convergence capability of learning algorithms such as Back-propagation using Magnified Gradient Function [20], Quickprop [21], Resilient Back-propagation [22], Levenberg-Marquardt Algorithm [23], Enhanced Two-Phase Method [24] and Fast Learning Algorithm with Promising Convergence Capability [25]. This section introduces the principle(s) of the above learning algorithms and their limitations.

2.1.1 Back-propagation with Magnified Gradient Function

In the standard BP, when the output values of the output layer $o_{pm}(i)$ or the hidden layer $y_{pk}(i)$ approach their extreme binary values (i.e., 0 or 1), the factors

$o_{pm}(i)(1-o_{pm}(i))$ and $y_{pk}(i)(1-y_{pk}(i))$ cannot reflect the true error $(t_{pm}-o_{pm}(i))$ (as shown in Equations (1.13) and (1.14)), and the neuron weights adjustment becomes insignificant or even unchanged. This is why the learning rate of BP is very slow when output values approach their target output values. On the other hand, when output values go to other extremes, the neuron weights adjustment also becomes insignificant. Thus the learning rate of the BP is also very slow but this time the learning is trapped by a local minimum.

Back-propagation with Magnified Gradient Function (MGF) magnifies the factors $o_{pm}(i)(1-o_{pm}(i))$ and $y_{pk}(i)(1-y_{pk}(i))$ by using a power factor (1/*S*), where *S* is a positive real number larger than 1. The original factors are replaced by $[o_{pm}(i)(1-o_{pm}(i))]^{\frac{1}{S}}$ and $[y_{pk}(i)(1-y_{pk}(i))]^{\frac{1}{S}}$. Thus, in Equation (1.13) and (1.14), $\delta_{pm}(i)$ and $\overline{\delta}_{pk}(i)$ can have larger increments when the output values approach 0 or 1, and the change of neuron weights is also larger. The weight-update in MGF can be presented as:

$$\Delta\omega_{km}(i+1)=\mu\cdot\sum_{p=1}^{P}\delta_{pm}^{MGF}(i)\cdot y_{pk}(i)+\alpha\Delta\omega_{km}(i) \tag{2.1}$$

$$\Delta\overline{\omega}_{nk}(i+1)=\mu\cdot\sum_{p=1}^{P}\overline{\delta}_{pk}^{MGF}(i)\cdot x_{pn}+\alpha\Delta\overline{\omega}_{nk}(i) \tag{2.2}$$

where

$$\delta_{pm}^{MGF}(i)=(t_{pm}-o_{pm}(i))[o_{pm}(i)(1-o_{pm}(i))]^{\frac{1}{S}} \tag{2.3}$$

and $$\overline{\delta}_{pk}^{MGF}(i)=[y_{pk}(i)(1-y_{pk}(i))]^{\frac{1}{S}}\sum_{m=1}^{M}\delta_{pm}^{MGF}(i)\omega_{km}(i) \tag{2.4}$$

Based on the above modification, Fig. 2.1 shows how MGF is implemented in the backward pass phase of BP.

For all neuron weights,

$$\delta_{pm}^{MGF}(i) = (t_{pm} - o_{pm}(i))[o_{pm}(i)(1-o_{pm}(i))]^{\frac{1}{S}}$$

$$\Delta\omega_{km}(i+1) = \mu \cdot \sum_{p=1}^{P} \delta_{pm}^{MGF}(i) \cdot y_{pk}(i) + \alpha\Delta\omega_{km}(i)$$

$$\overline{\delta}_{pk}^{MGF}(i) = [y_{pk}(i)(1-y_{pk}(i))]^{\frac{1}{S}} \sum_{m=1}^{M} \delta_{pm}^{MGF}(i)\omega_{km}(i)$$

$$\Delta\overline{\omega}_{nk}(i+1) = \mu \cdot \sum_{p=1}^{P} \overline{\delta}_{pk}^{MGF}(i) \cdot x_{pn} + \alpha\Delta\overline{\omega}_{nk}(i)$$

Fig. 2.1 MGF algorithm

MGF improves the performance of the standard BP in terms of the convergence rate. However, the error overshoot problem (the error signal $\delta_{pm}^{MGF}(i)$ or $\overline{\delta}_{pk}^{MGF}(i)$ is too large so that it overshoots the system error and thus takes more iterations to converge to a global solution) occurs if the magnification is too aggressive; therefore the learning process has to spend more time converging. Finally, the convergence capability of MGF is better than BP through magnifying the true error signal but it cannot totally solve the local minimum problem.

2.1.2 Quickprop

Quickprop is a popular learning algorithm based on BP. In BP, the weight-update is calculated by the partial derivative of the system error with respect to the neuron weights. Quickprop assumes that the relationship between the system error and the change of neuron weights can be approximated by an arms open upward parabola. This parabola is determined by measuring the previous gradient of the error surface ($\partial E(i-1)/\partial\omega$) and the current one ($\partial E(i)/\partial\omega$). Note that the weight-update for each weight is independent. It can be obtained as follows [21]:

$$\Delta\omega(i) = \frac{\partial E(i)/\partial\omega}{\partial E(i-1)/\partial\omega - \partial E(i)/\partial\omega}\Delta\omega(i-1) \tag{2.5}$$

Fig. 2.2 shows how Quickprop is implemented in the backward pass phase of BP.

For all weights and biases

$$\delta_{pm}(i) = (t_{pm} - o_{pm}(i))o_{pm}(i)(1-o_{pm}(i))$$

$$\partial E(i)/\partial\omega_{km} = \sum_{p=1}^{P}\delta_{pm}(i)\cdot y_{pk}(i)$$

$$\bar{\delta}_{pk}(i) = y_{pk}(i)(1-y_{pk}(i))\sum_{m=1}^{M}\delta_{pm}(i)\omega_{km}(i)$$

$$\partial E(i)/\partial\bar{\omega}_{nk} = \sum_{p=1}^{P}\bar{\delta}_{pk}(i)\cdot x_{pn}$$

$$\Delta\omega_{km}(i) = \frac{\partial E(i)/\partial\omega_{km}}{\partial E(i-1)/\partial\omega_{km} - \partial E(i)/\partial\omega_{km}}\Delta\omega_{km}(i-1)$$

$$\Delta\bar{\omega}_{nk}(i) = \frac{\partial E(i)/\partial\bar{\omega}_{nk}}{\partial E(i-1)/\partial\bar{\omega}_{nk} - \partial E(i)/\partial\bar{\omega}_{nk}}\Delta\bar{\omega}_{nk}(i-1)$$

Fig. 2.2 Quickprop algorithm

It is expected that the curve would move directly towards the minimum point of the parabola in the process of modifying the weights. However, once the mean square error reaches a minimum, and that minimum is not the global minimum, the learning will be trapped there and will most probably never escape from it. Compared with BP, the convergence rate of Quickprop is much faster, but the probability that the learning is trapped by a local minimum using Quickprop is higher than that by using BP. Thus the convergence capability of Quickprop is poor.

2.1.3 Resilient Back-propagation

Resilient back-propagation is another popular learning algorithm because it is one of the fastest learning algorithms. In RPROP, each network weight has its individual

update-value (i.e., $\Delta(i)$, i is the number of iterations). Unlike other fast learning algorithms, calculating an update-value does not include the partial derivative of the system error with respect to the weights ($\partial E/\partial \omega$); calculation is based on changes of the partial derivative sign. If the sign does not change in successive iterations (i.e., $\partial E(i-1)/\partial \omega \cdot \partial E(i)/\partial \omega > 0$), the update-value will be increased by an increasing factor (η^+) to take a bigger step than last time and move faster; otherwise (i.e., $\partial E(i-1)/\partial \omega \cdot \partial E(i)/\partial \omega < 0$), the update-value will be decreased by a decreasing factor (η^-) to take a smaller step next time and get closer to the minimum of the error surface. The update-value is adapted as below [22]:

$$\Delta(i) = \begin{cases} \eta^+ \Delta(i-1), & \text{if } \partial E(i-1)/\partial \omega \cdot \partial E(i)/\partial \omega > 0 \\ \eta^- \Delta(i-1), & \text{if } \partial E(i-1)/\partial \omega \cdot \partial E(i)/\partial \omega < 0 \\ \Delta(i-1), & \text{else} \end{cases} \tag{2.6}$$

After that, the weight-update can be calculated by the update-value by using Equation (2.7). If the derivative is positive, the weight will be decreased by its update-value; otherwise, the weight will be increased. The rules of weight-update are shown below [22]:

$$\Delta\omega(i) = \begin{cases} -\Delta(i), & \text{if } \partial E(i)/\partial \omega > 0 \\ +\Delta(i), & \text{if } \partial E(i)/\partial \omega < 0 \\ 0, & \text{else} \end{cases} \tag{2.7}$$

Fig. 2.3 shows the implementation of RPROP.

Repeat

Calculate the *MSE*

Computer the gradient $\partial E(i)/\partial \omega$

For all weights and biases

If $\partial E(i-1)/\partial \omega \cdot \partial E(i)/\partial \omega > 0$

$\Delta(i) = \eta^{+}\Delta(i-1)$

$\Delta\omega(i) = -sign(\partial E(i)/\partial \omega)\cdot \Delta(i)$

$\omega(i+1) = \omega(t) + \Delta\omega(i)$

$\partial E(i-1)/\partial \omega = \partial E(i)/\partial \omega$

Else if $\partial E(i-1)/\partial \omega \cdot \partial E(i)/\partial \omega < 0$

$\Delta(i) = \eta^{-}\Delta(i-1)$

$\partial E(i-1)/\partial \omega = 0$

Else

$\Delta\omega(i) = -sign(\partial E(i)/\partial \omega)\cdot \Delta(i)$

$\omega(i+1) = \omega(t) + \Delta\omega(i)$

$\partial E(i-1)/\partial \omega = \partial E(i)/\partial \omega$

Until (converged)

Fig. 2.3 RPROP algorithm

RPROP is very fast since it has an effective variable step size mechanism to modify the weights. Moreover, RPROP is robust compared with BP; it does not require specifying any parameters (e.g., learning rate or momentum) for different learning problems. However, the rule of weight-update is still based on derivatives of

the system error. Thus, RPROP always converges to the first minimum, but that may not be the global minimum. Therefore, it still suffers from the local minimum problem.

2.1.4 Levenberg-Marquardt Algorithm

Levenberg–Marquardt algorithm (LM) is well known as one of the most efficient learning algorithms. It combines the speed of the Gauss–Newton algorithm with the stability of the gradient descent algorithm. The weight-update of LM is:

$$\Delta\omega = \left[J^T J + \mu I\right]^{-1} J^T e \tag{2.8}$$

where $$J = \begin{bmatrix} \frac{\partial E_{11}}{\partial \omega_1} & \frac{\partial E_{11}}{\partial \omega_2} & \cdots & \frac{\partial E_{11}}{\partial \omega_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial E_{1M}}{\partial \omega_1} & \frac{\partial E_{1M}}{\partial \omega_2} & \cdots & \frac{\partial E_{1M}}{\partial \omega_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial E_{P1}}{\partial \omega_1} & \frac{\partial E_{P1}}{\partial \omega_2} & \cdots & \frac{\partial E_{P1}}{\partial \omega_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial E_{PM}}{\partial \omega_1} & \frac{\partial E_{PM}}{\partial \omega_2} & \cdots & \frac{\partial E_{PM}}{\partial \omega_N} \end{bmatrix}, \tag{2.9}$$

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.10}$$

and $$e = \begin{bmatrix} E_{11} \\ \cdots \\ E_{1M} \\ \cdots \\ E_{P1} \\ \cdots \\ E_{PM} \end{bmatrix}. \quad (2.11)$$

Note that μ is a small positive real number called combination coefficient that is defined as 0.1 here. J is the Jacobian matrix, J^T is the transpose of the Jacobian matrix, I is the identity matrix and e is the error vector for all the training patterns.

During the learning, if the current *MSE* is decreased because of the updated weights, the combination coefficient μ is decreased by a factor of 10 (i.e., $\mu/10$) for the next update to speed up the learning rate (the Gauss–Newton algorithm is obtained when μ is zero). If the current *MSE* is increased because of the updated weights, the combination coefficient μ is increased gradually (i.e., $\mu*10$) to avoid causing the error overshoot problem (the gradient-descent method is obtained when the μ approaches positive infinity). The implementation of LM is shown in Fig. 2.4:

Repeat

Calculate the *MSE(i)*

Computer change of weights and update the weights

$$\Delta\omega = [J^T J + \mu I]^{-1} J^T e$$

$$\omega = \omega + \Delta\omega$$

Calculated the *MSE(i+1)*

If *MSE(i+1)< MSE(i)*

$$\mu = \mu \div 10$$

Else

$$\mu = \mu \cdot 10$$

Until (converged)

Fig. 2.4 LM algorithm

LM is an efficient learning algorithm for small-sized networks. If the network size is large, the memory cost for the Jacobian matrix will become huge. Thus it spends a lot of computation time computing the weight-updates, so the convergence process of LM is sometimes even slower than the standard BP. Moreover, LM also suffers from the local minimum problem since the Gauss–Newton algorithm or gradient descent method is not capable of jumping out of a local minimum.

2.1.5 Enhanced Two-Phase Method

The idea of the Enhanced Two-Phase Method (E2P) is to apply two different algorithms in two learning phases respectively. When the learning is trapped by a

local minimum, it switches to another learning algorithm and hopes that this new learning algorithm can escape from the local minimum. In E2P, the change of mean square error (ΔMSE) is used to identify the existence of the local minimum, and another learning algorithm will be applied if ΔMSE is not changed (e.g., $\Delta MSE = 0$) or decreases very slowly (e.g., ΔMSE is less than 0.001 within 1000 iterations). It is expected that the conversion could adjust weights of the network and help the learning escape from the local minimum, and then the original learning algorithm will be re-applied. The E2P algorithm is shown in Fig. 2.5 [24].

```
Repeat
    Learning Algorithm = Method 1
    Calculate the ΔMSE
    If the learning is trapped
        Learning Algorithm = Method 2
    Else
        Learning Algorithm = Method 1
Until (converged)
```

Fig. 2.5 E2P algorithm

From the simulation results shown in [24], the performance of E2P is much better than some existing popular learning algorithms. E2P not only speeds up the convergence rate but also improves the global convergence capability. However, the

percentage of converged runs over 100 different runs with different initial weights cannot reach 100% (94% and 93% for the Five-bit Counting problem and Wine problem respectively [24]). It means that E2P can partially solve the local minimum problem.

2.1.6 Fast Learning Algorithm with Promising Convergence Capability

To solve the local minimum problem completely, a systematic approach called Fast Learning Algorithm with Promising Convergence Capability (PCC) is proposed in [25]. The motivation is the drawback of E2P. Using E2P, it is still possible that the learning is trapped by a local minimum because neither learning algorithm may escape from the local minimum. Moreover, the switched learning algorithm may make the learning move to another local minimum. Two learning algorithms are simply not good enough to handle all local minima.

PCC keeps trying to escape from local minima. The learning process is regarded as particular stages according to the different learning algorithms. This is because that each learning algorithm has its own characteristics to speed up convergence and its own path to jump out of the local minima (e.g., RPROP is quite fast but easily trapped by local minima, while MGF is slow but has a better convergence capability). The PCC algorithm is shown in Fig. 2.6 [25].

Begin

MGF = MGF (S=2)

Use Method RPROP, First = true, TempE = 0

Repeat

Calculate the ΔMSE

If $(T_1 \le \Delta MSE \le 0)$ and (First is true) // Trapped first time

Restore the initial weights

Use Method MGF

Count = 0, First = false

If $(T_2 \le \Delta MSE \le 0)$ and (First is true) and (Method = MGF)

Use Method RPROP, TempE = *MSE*

If $(T_3 \le \Delta MSE \le 0)$ and (First is true) and (Method = RPROP)

Restore the new initial weights

If (*Count* < 2)

Use Method MGF, *Count* = *Count* + 1

Else

Use Method MGF (S=5), Restore the initial weights

If (Method = RPROP) and (First is false) and (*MSE* < TempE – 10^{-5})

Record the current weights, TempE = 0

Until (converged)

End

Fig. 2.6 PCC algorithm

Note that three different thresholds (T_1, T_2, T_3) are used in the PCC algorithm to identify the existence of local minima. The values of these thresholds are highly related to the learning problem and the chosen learning algorithm. From the simulation results in [25], PCC was faster than the original learning algorithm and could always solve the local minimum problem in the Iris and Breast Cancer problems. However, it did not show that PCC can work in other learning problems.

2.2 Limitations

MGF, Quickprop, RPROP and LM mainly focus on the flat-spot problem. They speed up the learning process and their convergence rates are faster than the standard BP. However, the learning is still easily trapped by a local minimum and their global convergence capabilities are still poor.

E2P and PCC can partially solve both flat-spot and local minimum problems. Their global convergence capabilities are much better than some existing learning algorithms, but they still have their own limitations. E2P cannot solve the local minimum problem totally. The biggest problem with PCC is that its parameter setting is highly related to the characteristics of the specific learning problem. Thus it is difficult to determine suitable parameters for different problems.

Chapter 3 Wong Output Modification (WOM) Algorithm

3.1 Motivations

When learning is trapped by a local minimum, the change of weights is zero or extremely small, and the mean square error cannot be further reduced but its value is greater than the fixed error threshold (FET). In this thesis, FET is defined as ε_T which is usually a very small positive real number (default is 0.001), i.e.,

$$MSE(i) = \frac{1}{PM}\sum_{p=1}^{P}\sum_{m=1}^{M}[t_{pm} - o_{pm}(i)]^2 > \varepsilon_T \qquad (3.1)$$

The performance investigation revealed that some output values of training patterns are close or equal to other extremes compared with their corresponding target output values. Such output values make the mean square error larger than ε_T. Note that the range of an output value is 0 to 1 since the sigmoid function is used as the activation function for both the hidden and output layers. For example: an output value is close to 1 (e.g., 0.95) but its corresponding target value is 0, and vice versa. Such output values are declared as wrong output values. Note that it is normal that some output values sometimes move in the opposite direction in training. Ultimately they will move back to their corresponding target output values when the learning is about to converge. However, if they never move back, the mean square error cannot be further decreased to be less than ε_T.

Fig. 3.1 shows the learning process trained by RPROP is trapped by a local minimum in the Five-bit Counting problem (described later in Chapter 5). The mean square error drops quickly at the beginning of the learning. However, the convergence

rate decreases significantly after 200 iterations. After that, the convergence rate becomes very slow. Finally, the convergence rate is very close to zero (after 500 iterations), which means the mean square error cannot decrease further (see Fig. 3.2). In this situation, the 5^{th} target output value in the first pattern and the second target output value in the 32^{nd} pattern are both one but their corresponding output values are very close to zero. Wrong output values are found in these two patterns. Thus, the mean square error equals 0.010417 which is larger than the error threshold, whereby the learning is trapped by the local minimum.

Based on this observation that wrong outputs exist when the learning is trapped by a local minimum, we wondered if the local minimum problem could be solved if wrong outputs could be removed.

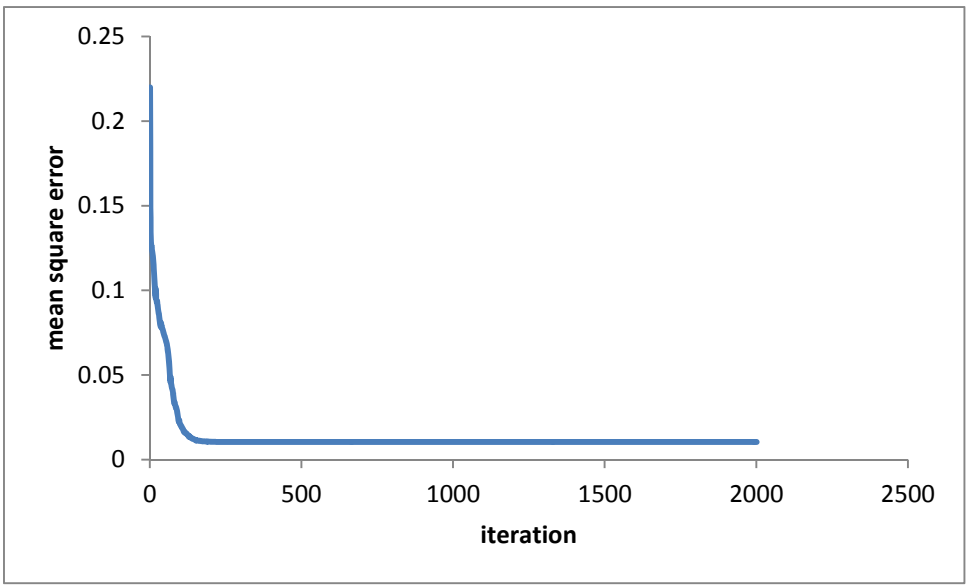

Fig. 3.1 RPROP is trapped by a local minimum in Five-bit Counting

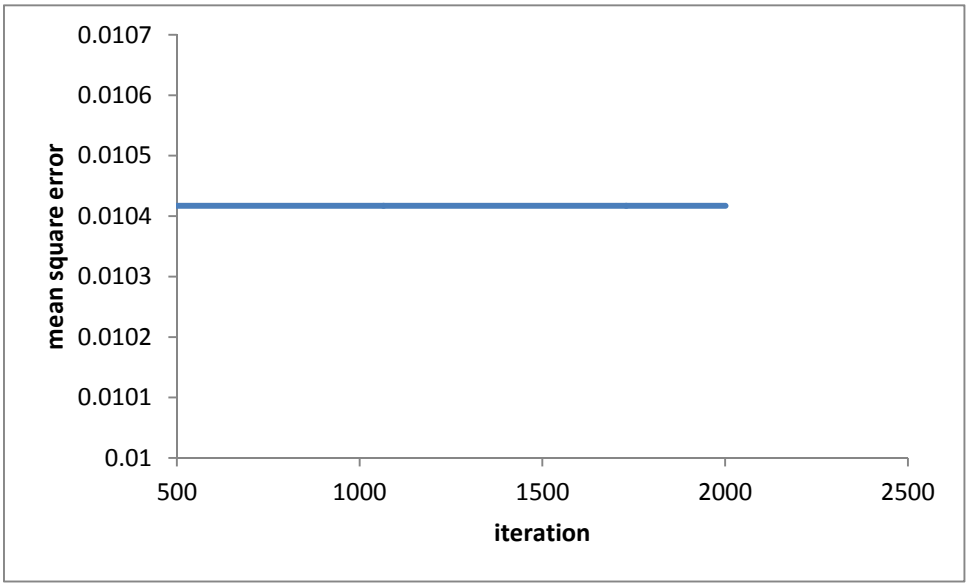

Fig. 3.2 Mean square error is unchanged

3.2 Proposed Algorithm: Wrong Output Modification

Based on the above findings, we intend to locate wrong output values and modify them according to their corresponding target output values, so as to guide them in the right direction(s) [26].

Theorem 1: *The learning is declared to be trapped by a local minimum if the change of the mean square error is very close to zero but the mean square error is larger than the error threshold.*

Proof: Typically the input-output relationship of a neural network is

$$o_{pm}(i) = \Phi(\Omega_m(i), X_m^{(p)}) \tag{3.2}$$

where $\Omega_m(i)$ is the set of neuron weights related to the output m at the i^{th} iteration, $X_m^{(p)}$ is the set of input values related to the output m and pattern p, and Φ is a complicated real function. By using (3.1) and (3.2), we know that

$$MSE(i) = h(\Omega(i)) \tag{3.3}$$

where h is a real function. Moreover, $MSE(i)$ and $\Omega(i)$ are the mean square error and the set of all neuron weights at the i^{th} iteration (i.e., $\Omega_m(i) \subset \Omega(i)$) respectively. [24] shows that if $\Omega^*(i)$ is a local minimum of h, it has two properties:

$$h(\Omega^{(g)}) < h(\Omega^*(i)) \tag{3.4}$$

and $$\nabla h(\Omega^*(i)) = 0 \tag{3.5}$$

where $\Omega^{(g)}$ is a global minimum of h and $\nabla h(\Omega^*(i))$ is the gradient of h at $\Omega^*(i)$. Since the system error is zero when the learning converges to a global solution, by using Equations (3.4) and (3.5) there are two conditions to identify the existence of a local minimum:

Condition 1: Usually a threshold (i.e., an error threshold — it should be a very small positive real number) is used to identify whether the learning converges to a global solution or not. By considering Equation (3.4), the same threshold can be used to identify a local minimum. If the learning converges to a minimum but the minimum is greater than the threshold, the minimum is declared to be a local minimum.

Condition 2: In Equation (3.5), it is found that the change of the mean square error is zero when the learning is trapped by a local or global minimum. Thus, the change of the mean square error can signal the existence of a local minimum. When the change of the mean square error is less than a very small threshold (e.g, 10^{-6}), it signals that the learning has converged to a minimum. Note that the change of the mean square error is zero when the learning converges to a local or global minimum. Thus, a minimum is declared to be local only if both conditions (Condition 1 and

Condition 2) are met.

Theorem 2: *When the learning is trapped into a local minimum, the values of neuron weights that generate wrong output values are not equal to those of the same neuron weights when the learning converges to a global minimum.*

From Equation (3.2), we have

$$o_{pm}^{(l)} = \Phi(\Omega_m^{(l)}, X_m^{(p)}) \tag{3.6}$$

where $o_{pm}^{(l)}$ and $\Omega_m^{(l)}$ are the output value of output *m* with pattern *p* and the set of neuron weights related to output *m,* respectively, when the learning is trapped into a local minimum. On the other hand, when the learning converges to a global solution, we have

$$o_{pm}^{(g)} = t_{pm} = \Phi(\Omega_m^{(g)}(k), X_m^{(p)}) \tag{3.7}$$

where $o_{pm}^{(g)}$ and $\Omega_m^{(g)}$ are the desired output value of output *m* with pattern *p* and is the set of neuron weights related to output *m* respectively when the learning converges to the k^{th} global solution. Note that

$$\Omega_m^{(g)}(k) \in \Omega_m^{(G)} = \{\Omega_m^{(g)}(k), k = 1, 2, \ldots\} \tag{3.8}$$

where $\Omega_m^{(G)}$ is the set of global solutions related to output *m*. According to Theorem 2,

$$o_{pm}^{(l)} \neq o_{pm}^{(g)} \Rightarrow \Omega_m^{(l)} \notin \Omega_m^{(G)}. \tag{3.9}$$

Suppose that $o_{pm}^{(l)} \neq o_{pm}^{(g)}$ but $\Omega_m^{(l)} \in \Omega_m^{(G)}$. This means *k* exists such that $\Omega_m^{(l)} = \Omega_m^{(g)}(k)$.

However, $o_{pm}^{(l)} = \Phi(\Omega_m^{(l)}, X_m^{(p)}) = \Phi(\Omega_m^{(g)}(k), X_m^{(p)}) = o_{pm}^{(g)}$. Thus, contradiction occurs.

It is proved that when the learning is trapped into a local minimum, the values of neuron weights that generate wrong output values do not equal those of the same neuron weights when the learning converges to a global minimum. Through modifying outputs, the changes in such neuron weights may help the learning to escape from the local minimum. Thus, wrong output values should be modified to change such neuron weights. However, wrong outputs cannot be simply moved to their desired output values because this may violate the trend of the original learning process significantly and thus the learning may become very unstable. Therefore the modification should be small but sufficient so that the weight update equations gradually move the neuron weight back to their appropriate values through the modification. If that works, the learning can escape from the local minimum. Note that the modification will be made once when the learning is trapped by a local minimum. Thus, the stability of the learning can still be maintained.

Based on this idea, a new algorithm is proposed as shown in Fig. 3.3 [26]. In this algorithm, ΔMSE is the difference of the mean square error within 10 iterations. Based on Theorem 1, a local minimum is declared if ΔMSE is less than 10^{-8} or less than 10^{-5} within 1,000 iterations.

Initialization:

WOT_1 = 0.95, WOT_0 = 0.05

AOV_1 = 0.9, AOV_0 = 0.1

Repeat

If (t_{pm} = 0) and ($o_{pm} \geq$ WOT_1) then o_{pm} = AOV_1

If (t_{pm} = 1) and ($o_{pm} \leq$ WOT_0) then o_{pm} = AOV_0

Process the selected fast learning algorithm

For all patterns *p*

Calculate the change of the mean square error, *ΔMSE*

If it is trapped by a local minimum

Check the past history

If it is a new local minimum

AOV_1 = WOT_1 – 0.05

AOV_0 = WOT_0 + 0.05

Else // it has been visited before

AOV_1 = AOV_1 – 0.1

If (AOV_1 = 0.1)

then WOT_1 = WOT_1 – 0.1

AOV_1 = WOT_1 – 0.05

If (WOT_1 = 0.55)

then WOT _1 = 0.95

AOV_1 = WOT_1 – 0.05

AOV_0 = AOV_0 + 0.1

If (AOV_0 = 0.9)

then WOT_0 = WOT_0 + 0.1

AOV_0 = WOT_0 + 0.05

If (WOT_0 = 0.45)

then WOT_0 = 0.05

AOV_0 = WOT_0 + 0.05

Update the corresponding weights

Until converge

Fig. 3.3 The WOM algorithm (initial version) [26]

In Fig. 3.3, AOV_0 and AOV_1 are assigned output values for the output close to 0 and 1 respectively. Moreover, WOT_0 and WOT_1 are wrong output thresholds for the output close to 0 and 1 respectively. In this additive procedure, some output values that are very close to another extreme (i.e., WOT_0 and WOT_1 are close to 0 and 1 at the beginning respectively) will be modified and a small modification is expected to be enough to escape from a local minimum. If the learning cannot escape from such local minimum, the wrong output thresholds will be adjusted (i.e., WOT_0 increases and WOT_1 decreases) so that more output values are considered as wrong outputs and the modification will be further increased.

Fig. 3.4 shows the effect of WOM when applied in the case that RPROP is trapped by a local minimum in Fig. 3.1. The mean square error cannot be further reduced at the 311st iteration, and a local minimum is identified here. The mean square error changes obviously when the WOM is applied. The learning escapes from the local minimum, and finally converges at the 502nd iteration (i.e., the mean square error is less than 0.001).

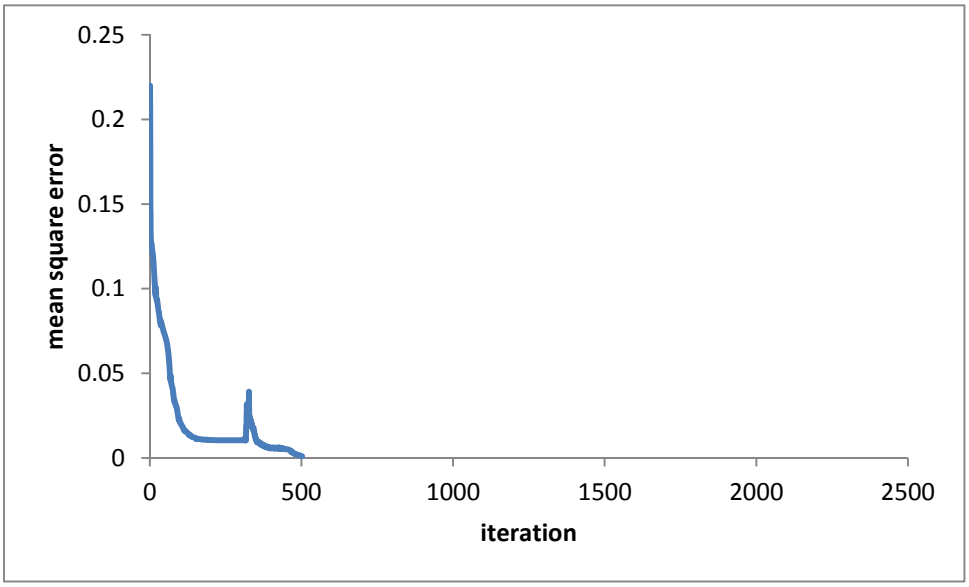

Fig. 3.4 RPROP jumps out of the local minimum by using WOM

The performance investigation using WOM demonstrated that it can significantly improve the global convergence capability of a learning algorithm. The detailed description of the performance investigation is included in Chapter 5. However, the local minimum problem cannot be totally solved because of several limitations:

1. *The setting to declare wrong output values is too conservative*: In Fig. 3.3, the upper bound of WOT_0 is 0.45 while the lower bound of WOT_1 is 0.55. It means an output cannot be considered as a wrong output if the difference between the output value and its corresponding target output value is less than 0.55 (an output can be declared as a wrong output if the difference is more than 0.5). This setting is too conservative so that, occasionally in some cases, the learning cannot escape from a local minimum because some wrong outputs cannot be included in the modification process.
2. *The modification of wrong output values does not adapt to the current status*

of the learning process: The modification of wrong output values is fixed in the above WOM algorithm, but modification(s) should be related to the stability of the learning. When the modified output value is far away from its corresponding target output value but the learning cannot escape from a local minimum, stability is not a critical issue and thus modification should be increased significantly so that the modified output value gets closer to the target output value. It is hoped that the chance to escape from the local minimum can be significantly increased. When the modified output value is close to its corresponding target output value but the learning still cannot escape from a local minimum, stability now becomes critical (the learning may become unstable) and thus the modification should be increased slightly to maintain stability.

3. *The number of modifications is limited*: Since the modification is linearly additive, the number of modifications is limited. In [26], the number of modifications is limited to 16 times.
4. *Modifications occasionally cannot help the learning to escape from a local minimum*: In our performance investigation, WOM helped the learning escape from a local minimum (using different learning algorithms in different learning problems) most of the time. Occasionally, however, the learning cannot escape from a local minimum no matter how we modified wrong output values. We studied such special cases very carefully and found that it happens because the global solutions are far away from the local minimum

and the local minimum is always more attractive than all global solutions. The investigation of this part is described in Chapter 5.

5. *Too many iterations are required to confirm that the learning cannot escape from a local minimum*: When the learning is trying unsuccessfully to escape from a local minimum, our algorithm takes many iterations to confirm that it reverts to the local minimum. Thus, even if the learning finally escapes from the local minimum after trying a number of times, the convergence rate will have dropped significantly.

The WOM algorithm has been improved to overcome the above limitations [27, 28]. The improved algorithm is shown in Fig. 3.5, Fig. 3.6 and Fig. 3.7. In Fig. 3.6, Γ and k are set to 0.95 and 0.9 respectively while ε_k and ε_Γ are set to 0.9. The performance investigation shows that these parameter settings are robust for the algorithm to perform effectively. They are adequate if they are not close to zero so that stability of the learning can be maintained.

Repeat

Process the selected fast learning algorithm

Calculate the change of the mean square error, ΔMSE

If it is trapped by a local minimum,

Process the procedure *Escape* to escape from the local minimum.

If the procedure *Escape* was processed,

Process the procedure *FastChecking* to identify whether the learning

goes back to the previous local minimum or not.

Update the corresponding weights.

Until converge

Fig. 3.5 The WOM algorithm (final version) [27, 28]

Procedure *Escape*

Begin

If it is the first time to be trapped by a local minimum,

For all output values,

Δ **= |Wrong output value – Desired output value|**

If ($\Delta > \Gamma$**)**

$\Delta \leftarrow k \times \Delta$ **where** $0 < k < 1$**.**

New output value = |Desired output value – Δ**|**

Endfor

Endif

Else

If it is still trapped by the local minimum,

// Move closer to desired outputs

$k \leftarrow k \times \varepsilon_k$ **where** $0 < \varepsilon_k < 1$**.**

For all output values,

Δ **= |Wrong output value – Desired output value|**

If ($\Delta > \Gamma$**)**

$\Delta \leftarrow k \times \Delta$

New output value = |Desired output value – Δ|

Endfor

Endif

Else

If it is still trapped by the local minimum and Δ is too small,

// Involve more outputs

$\Gamma \leftarrow \Gamma \times \varepsilon_{\Gamma}$ **where 0 <** ε_{Γ} **< 1.**

For all output values,

Δ **= |Wrong output value – Desired output value|**

If ($\Delta > \Gamma$**)**

$\Delta \leftarrow k \times \Delta$

New output value = |Desired output value – Δ**|**

Endfor

Endif

Else

If it is still trapped by the local minimum

and Δ **is sufficiently small**

and all possible wrong output values are involved,

Generate a new set of initial weights but the region of weights that is trapped by the local minimum is excluded.

Re-start the learning.

Endif

End

Fig. 3.6 The procedure *Escape*

Procedure *FastChecking*

Begin

After each β iterations,

For k = 1, 2, …, K, // number of local minima

Count **$\leftarrow$ 0;**

For l = 1, 2, …, L, // number of neural weights

If ($\omega_l^{(\beta)} > \omega_l > \omega_l^{(k)}$ **) or (** $\omega_l^{(\beta)} < \omega_l < \omega_l^{(k)}$ **),**

// this weight approaches to k^{th} local minimum

Count **=** ***Count*** **+ 1;**

Endfor

If (*Count* $\geq \Psi$), // confirm to approach, $\Psi = 0.8L$

Process the procedure Escape.

Endfor

End

Fig. 3.7 The procedure *FastChecking*

The following improvements have been made in this updated algorithm:

1. *All possible wrong output values can be modified*: In Fig. 3.5, only some extreme wrong output values will be considered at the beginning in order to maintain stability of learning. If the learning cannot escape from a local minimum, more wrong output values will be involved until all possible wrong output values are involved.
2. *The modification of wrong output values adapts to the current status of the learning process*: In Fig. 3.5, a multiplication modification is used in the algorithm. Thus the change of the modification is large at the beginning so that the change of the learning is large and it has a high probability of escaping from a local minimum. When the modified output value is closer to its target output value, the change is smaller to maintain stability of the learning.
3. *The number of modifications is extended significantly*: Since the modification is changed by multiplying a factor, the number of modifications is much greater than the previous algorithm. In [27], the modification can be made 73 times before the procedure to re-generate initial weights proceeds (this procedure will be described later).
4. *A procedure to re-generate initial weights is introduced occasionally if the learning cannot escape from a local minimum using WOM*: When all wrong output values have been involved and modified so that they are close to their corresponding output values but the learning still cannot escape from a local

minimum, we concede that the local minimum is too far away from all global solutions and thus WOM cannot effectively help the learning. Thus we re-generate all initial weights and hope that the learning can escape from such a local minimum. Note that the previous region of neuron weights is excluded to avoid the new initial weights leading the learning to such a local minimum again.

Theorem 3. If the learning re-starts with a new set of initial weights which is re-generated excluding the region of weights that is trapped by the local minimum, the probability that the learning converges to a global solution this time is higher than the probability that the learning converges to a global solution with a set of randomly-generated initial weights.

Proof. Let $\omega_i^{(I)}$ and $\omega_i^{(I')}$ be the initial values of ωi when it is generated at the beginning and when the learning re-starts. Let $\omega_{i,u}$ and $\omega_{i,l}$ be the upper and lower bounds of $\omega_i^{(I)}$. Then,

$$\omega_i^{(I)} \in \{\omega_i : \omega_{i,u} \geq \omega_i \geq \omega_{i,l}\}$$

and

$$\omega_i^{(I)} \in \{\omega_i : \omega_{i,u} \geq \omega_i \geq \max(\omega_i^{(I)}, \omega_i^{(L)}) \text{ or } \min(\omega_i^{(I)}, \omega_i^{(L)}) \geq \omega_i \geq \omega_{i,l}\} . \quad (3.10)$$

Let R_i and R_i' be the valid ranges of ω_i when it is randomly generated at the beginning and when the learning re-starts. Moreover, let $R_{i,j}^{(G)}$ be the range of ω_i such that the learning converges to the j^{th} global solution. Then, we have

$$R_i = \omega_{i,u} - \omega_{i,l}, \tag{3.11}$$

and $$R'_i = \omega_{i,u} - \omega_{i,l} - \left|\omega_i^{(I)} - \omega_i^{(L)}\right|. \tag{3.12}$$

Let p and p' be the probabilities that the learning converges to a global solution at the beginning and after the learning re-starts, respectively. Then,

$$p = \frac{\sum_j \prod_i R_{i,j}^{(G)}}{\sum_i R_i} \tag{3.13}$$

and $$p' = \frac{\sum_j \prod_i R_{i,j}^{(G)}}{\sum_i R'_i}. \tag{3.14}$$

Based on Equations (3.13) and (3.14), we know $R_i > R'_i$ for all weights and therefore $p' > p$.

Theorem 4. This algorithm can finally converge to a global solution, if it exists.

Proof. Let $p^{(n)}$ be the probability that the learning converges to a global solution after the learning re-starts the n^{th} time. Through theorem 3, we know

$$p^{(n+1)} > p^{(n)}. \tag{3.15}$$

Let p_f be the probability that the learning finally converges to a global solution, then we have

$$p_f = 1 - \prod_{n=0} (1 - p^{(n)}) > 1 - (1-p)^n. \tag{3.16}$$

When $n \to \infty$, $p_f = 1$.

The above theorems demonstrate that the procedure of re-generating initial weights can guarantee the learning can finally converge to a global solution. Actually, during performance investigation, this procedure was executed at most once and the learning could converge to a global solution quickly after that.

5. *A fast checking procedure to identify whether the learning goes back to the local minimum is introduced to speed up the convergence rate*: The idea underlying this procedure is to check the tendency of each neural weight rather than the mean square error. This fast checking procedure can speed up checking by monitoring the change of neuron weights; that is, whether they have a trend (e.g., over 80% of neuron weights) to any of the existing local minima (i.e., local minima that have been visited before). If the trend is confirmed, it means the learning may revisit such a local minimum in the near future. The escape procedure should be processed in advance and not wait for the learning to be trapped by the local minimum again. Fig. 3.7 shows the fast checking procedure. Note that β is a small positive integer (e.g., 10), K is the number of local minima, L is the total number of neuron weights, and Ψ is the number of neuron weights approaching a local minimum.

Theorem 5. If most of neuron weights are approaching a local minimum, the probability that the learning will be trapped by this local minimum is very high.

Proof. Consider a simplified model to calculate the probability. Assume that the probability of a neuron weight approaching a local minimum is 0.5 (either approach it or not). Moreover, it is assumed that a weight must approach this local minimum if all other weights are approaching it. Finally, assume that the relationship between this probability and the number of neuron weights approaching the local minimum is linear. Then, given that N_w is the number of neuron weights approaching a local minimum, the probability that the learning

finally converges to such minimum is

$$\prod_{x=N_w}^{L-1} \frac{1}{2}\left(1+\frac{x}{L-1}\right) \tag{3.17}$$

where L is the total number of neuron weights. If L = 10 and 80% of neuron weights are approaching a local minimum (i.e., N_w = 8), the probability that the learning converges to this local minimum is 0.9444. If L = 40, the probability is reduced to 0.69, which is still a high probability. Additionally, the changes of neuron weights inside a neural network are usually highly correlated and thus this probability is even higher than in the above simplified model.

Therefore, this checking approach is very promising. The performance investigation showed that N_w is sufficiently large if it is set to 80% of the total number of neuron weights for all learning algorithms in all learning problems. The performance investigation found that the number of iterations for checking is significantly reduced and hence the convergence rate of a learning algorithm with this procedure is similar to the original learning algorithm.

Chapter 4 Threshold of Output Difference (TOD)

4.1 Motivations

Classification is an important contribution of a neural network. During training, the learning will be terminated when the mean square error is less than the fixed error threshold. That means this learning is declared to converge to a global minimum and the network has been well trained. It is expected that this network should classify the training data perfectly, and the network can be used to classify new data. That is why a neural network can act as a universal approximator [30 – 32], so many applications use neural networks for classification or prediction. However, our experiments found that the original training data sometimes cannot be correctly classified by the trained network. The problem is the terminating condition in the algorithm.

Consider the following example: RPROP is applied to the Breast Cancer learning problem (explained in Chapter 5). Note that the number of training patterns in the Breast Cancer problem is 699, ε_T is 0.001, and the maximum number of iterations is 5000. After 5000 iterations, RPROP still cannot converge to a global minimum. The mean square error is larger than 0.001. After training, all training patterns have been used for testing in this network. In this case, five out of 699 patterns are misclassified — the misclassification rate (MCR) is 0.7153%. The result is reasonable because the learning does not converge to a global minimum.

RPROP with WOM was applied to the same case. WOM solved the local minimum problem, and the mean square error was less than 0.001 after 2,049

iterations. That means the learning process converged to a global minimum this time. Thus, it is expected that a well trained network can classify all training patterns correctly. However, one out of 699 patterns was still misclassified — the MCR was 0.1431% which is not equal to zero.

From Equation (1.2), it is known that the mean square error is related to the size of the training data (*P*) and the number of output nodes (*M*). It is calculated by the sum of the entire training patterns. Note that *P* and *M* may be very different for different learning problems, but the error threshold is fixed (i.e., $\varepsilon_T = 0.001$) for all learning problems. Thus, there is a problem: ε_T may be too tough to meet if *P* and *M* are small. The learning has to spend unnecessary iterations to meet the criteria (make sure $MSE < \varepsilon_T$) when all outputs in all training patterns are already very close to their corresponding target outputs. On the other hand, ε_T may be too easy to meet if *P* and *M* are large; the setting of ε_T is not small enough so that several outputs in training patterns may not be trained completely, but the terminating condition is still met (i.e., $MSE < \varepsilon_T$). Thus, we conclude that using a fixed error threshold as a terminating condition may not be good enough to measure whether the learning converges to a global minimum or not, and whether the trained network may be suitable or not for classifying testing data.

4.2 New Terminating Condition: Threshold of Output Differences

Classification is one of the most important applications of neural networks. In the classification process, if the difference between a real output and its corresponding

target output is less than 0.5, it is declared that this output is correctly classified. For example, if a target output is one and its corresponding real output is 0.55, it is declared that 0.55 is equal to one and this output is correctly classified. The value of a real output is not required to be very close to one (e.g., 0.9).

Based on this principle of classification, a new termination condition called Threshold of Output Differences (TOD) is proposed to identify whether the learning converges to a global minimum or not [29]. TOD is:

$$\left| t_{pm} - o_{pm} \right| < 0.5 \tag{4.1}$$

where

$p = 1, 2, \ldots, P$ and $m = 1, 2, \ldots, M$.

It is claimed that the learning converges to a global minimum if the difference between each training pattern output and its corresponding target output is less than 0.5, which means all training patterns can be classified correctly. TOD is dynamic compared with the traditional termination condition (i.e., FET), and that is suitable for different learning problems. If P and M are small, this new termination condition (i.e., TOD) is easily satisfied and its convergence rate is faster. If P and M are large, this new termination condition may be difficult to satisfy but the trained network will perform better in classification.

TOD is implemented as shown in Fig. 4.1. Note that TOD can be applied in different learning algorithms since its implementation is independent of the operations of a learning algorithm.

Initialization:

Initialize all weights ω_{km}, $\overline{\omega}_{nk}$ **and set *Count* = 0.**

Repeat

For *p* = 1, 2, …, *P*

For *m* = 1, 2, …, *M*

$$y_{pk}(i) = f(\sum_{n=1}^{N} \overline{\omega}_{nk}(i) x_{pn})$$

$$o_{pm}(i) = f(\sum_{k=1}^{K} \omega_{km}(i) y_{pk}(i))$$

If $\left| t_{pm} - o_{pm} \right| < 0.5$

***Count* = *Count* + 1**

Else

***Count* = 0**

End For

End For

If *Count* = *P* × *M*,

The training is completed and the convergence is declared to be satisfied.

Else

Compute the changes of the weights for the next iteration by using a learning algorithm (e.g., BP, Quickprop).

Then update all weights.

Until converge

Fig. 4.1 Implementation of TOD

Note that a smaller threshold (e.g., 0.4 or 0.3) in training may not improve the classification rate in testing because of the existence of the over-fitting problem in neural networks.

4.3 Examples

Consider the same case used in Section 4.1. TOD is used as the terminating condition instead of the traditional FET. RPROP with TOD cannot converge within 5000 iterations, and the MCR is still 0.7153% (five out of 699 are misclassified). The MCR is not improved because of the limitation of RPROP (i.e., the local minimum problem), not TOD.

RPROP with WOM and TOD converges after 2,063 iterations. In this case, the convergence rate is a little bit slower than RPROP with WOM and FET. However, the MCR is equal to zero. Under the new terminating condition, all training patterns can be classified correctly (i.e., MCR = 0) by this trained network.

Chapter 5 Numerical Results

5.1 Introduction

This chapter describes the Wrong Output Modification (WOM) and Threshold of Output Differences (TOD) algorithms in detail with examples. Their performance with different learning algorithms in different learning problems (applications) is also described in this chapter.

In this chapter, five popular learning algorithms (BP, MGF, Quickprop, RPROP and LM, all of which were introduced in Chapter 2) were applied to six different benchmark learning problems to investigate how WOM and TOD performed. These benchmark learning problems are XOR, Three-bit Parity, Five-bit Counting, Iris, Wine, and Breast Cancer. These data sets can be found in the UCI Machine Learning Repository [33], which is a famous database for testing the performance of learning algorithms.

5.2 Learning Problems

Brief descriptions of the learning problems and their network configurations are shown in Table 5.1 and 5.2, where *N*, *K*, and *M* represent the number of input, hidden, and output nodes, and μ and α are the learning rate and the momentum of BP for those learning problems. Note that all these parameters are optimized in the different learning problems. They are found by using trial-and-error. Based on the number of training patterns, these six learning problems can be classified into two sets: simple

problems (i.e., XOR, Three-bit Parity and Five-bit Counting) and difficult problems (i.e., Iris, Wine and Breast Cancer).

Learning Problem	Description
XOR	"Give two binary inputs a and b, and output a⊕b." [33]
Three-bit Parity	"Give three binary inputs and output the odd parity of all inputs." [33]
Five-bit Counting	"Count the number of 1s from the five input units." [33]
Iris	"The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant." [33]
Wine	"These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines." [33]
Breast Cancer	"These data were obtained from the University of Wisconsin Hospitals, Madison, from Dr. William H. Wolberg. The databases reflect this chronological grouping of the data." [33]

Table 5.1 Learning problems (descriptions)

Learning Problem	Network Architecture *N-K-M*	Parameter Setting (μ, α)	Number of Training Patterns	Problem Difficulty
XOR	2-2-1	(0.5, 0.7)	4	Low
Three-bit Parity	3-3-1	(0.6, 0.9)	8	Low
Five-bit Counting	5-12-6	(0.1, 0.7)	32	Low
Iris	4-15-3	(0.02, 0.05)	150	High
Wine	13-10-3	$(10^{-8}, 0.1)$	178	High
Breast Cancer	9-20-1	(0.005, 0.03)	699	High

Table 5.2 Learning problems (parameter setting)

5.3 Simulation Environment

The simulation programs were written in C and MATLAB, and integrated development environments were Visual Studio 2010 and MATLAB 6.5 respectively. All experiments were processed in a personal computer with Windows 7.

Neural Network Toolbox is an important component in MATLAB, and it is easy to create, train, and simulate neural networks. It not only supports supervised learning with multi-layer feed-forward networks, but also provides many popular network training functions (e.g., Levenberg-Marquardt algorithm) for modeling complex nonlinear systems. In other words, LM can be called directly without dealing with complex matrix operations. In our experiments, LM with and without WOM were implemented in the Wine and Breast Cancer problems under a MATLAB environment. However, the training function and the terminating condition are packaged in MATLAB, and the FET acts as the terminating condition. Only the value of this threshold can be adjusted (i.e., $\varepsilon_T = 0.001$) before training. Thus, our proposed terminating condition (i.e., TOD) was not applied in LM under a MATLAB environment.

The performance investigations were divided into two parts: Part A focuses on the convergence rate (CR) and the convergence capability (CC). CR is the number of iterations spent to meet the terminating condition; while CC is the probability that the terminating condition is met in training. Part B focuses on the classification ability. The average percentage of misclassified patterns in testing patterns (MCR) is used as the performance measure. Note that the sets of initial weights were randomly

generated between -0.3 and 0.3. The maximum number of iterations was 1,000,000 for each run, which means the training was declared not to converge if the terminating condition cannot be met within 1,000,000 iterations.

We focus on one hidden layer network in my experiments since usually the structure is good enough to solve learning applications [34]. Usually it is less than 15% of applications need two hidden layers. Actually, we have already applied the proposed algorithm WOM and TOD to neural networks with two hidden layers (see section 5.8 in the thesis). The results are promising. Moreover, the numbers of hidden nodes used in the performance investigation are optimized by using trial-and-error in different learning applications.

5.4 The Effect of the Procedure to Re-generate Initial Weights

As mentioned in Chapter 3, WOM occasionally cannot escape from a local minimum and thus it is essential to re-generate the set of initial weights. This subsection shows some numerical results to explain why it is necessary to re-generate the set of initial weights.

In the escape procedure, if all wrong outputs are included but the training still cannot escape from the local minimum, the initial weights will be re-generated and the training will be re-started. This is because such a local minimum is very far away from all global minima. Let *DW(i)* be the difference of neuron weights between the i^{th} local minimum and the global minimum that the learning finally converges to:

$$DW(i) = \frac{1}{B}\sum_{b=1}^{B}\left|\frac{\omega_b^{LM}(i) - \omega_b^{GM}}{\omega_b^{GM}}\right| \quad (5.1)$$

Note that B is the number of neuron weights. Its value is related to the structure of learning problems. Let $\omega_b^{LM}(i)$ be neuron weights when the training is trapped by the i^{th} local minimum, and ω_b^{GM} be neuron weights when the training converges to the global minimum.

Consider some learning algorithms with WOM applied to the three difficult learning problems (i.e., Breast Cancer, Wine, and Iris). All trainings converge to a global minimum at the end since the local minimum problem can be solved by WOM. The comparisons of *DW(i)* are shown in Table 5.3. In the first seven cases, all local minima are close to the global minimum (*DW(i)* ≤ 1.27) and thus WOM can effectively escape from all local minima and finally converge to a global solution. In the last three cases, the procedure to re-generate initial weights was implemented and the learning finally converged to a global minimum. The data show that all local minima were far away from the global minimum (*DW(i)* ≥ 7.95 and the difference can be up to 261.31) and thus we need this procedure to escape from local minima. Note that this procedure was executed once only when it happened, after which the learning could converge to a global minimum.

Case	Fast Learning Algorithm	Learning Problem	Number of local minima visited	Re-generate?	*DW(i)*
1	RPROP	Breast Cancer	1	No	*DW(1)* = 0.88
2	RPROP	Breast Cancer	2	No	*DW(1)* = 0.97 *DW(2)* = 0.43
3	BP	Breast Cancer	1	No	*DW(1)* = 1.16
4	Quickprop	Wine	1	No	*DW(1)* = 0.21
5	RPROP	Wine	1	No	*DW(1)* = 0.04
6	RPROP	Iris	4	No	*DW(1)* = 1.27 *DW(2)* = 1.17 *DW(3)* = 0.84 *DW(4)* = 0.10
7	Quickprop	Iris	4	No	*DW(1)* = 0.98 *DW(2)* = 0.96 *DW(3)* = 1.01 *DW(4)* = 0.05
8	Quickprop	Wine	3	Yes	*DW(1)* = 89.38 *DW(2)* = 261.31* *DW(3)* = 0.02
9	RPROP	Wine	4	Yes	*DW(1)* = 7.95 *DW(2)* = 8.77 *DW(3)* = 9.34 *DW(4)* = 8.81*
10	Quickprop	Iris	5	Yes	*DW(1)* = 26.31 *DW(2)* = 26.26* *DW(3)* = 1.91 *DW(4)* = 0.47 *DW(5)* = 0.083

Table 5.3 Difference of neuron weights in different cases

*: The procedure to re-generate the initial weights was implemented.

5.5 Performance Comparison between WOM (initial version) and WOM (final version)

As mentioned in Chapter 3, the initial version of WOM was not good enough [26] and thus WOM has been significantly improved in [27] and [28]. In this subsection, some numerical results are shown to illustrate its improved performance in terms of convergence rate and global convergence capability. The performance comparison is shown in Table 5.4.

In this table, the first row shows different versions of WOM. The first column shows different algorithms applied to the learning algorithm. “Original” represents the original Quickprop. The two numbers inside a cell show the performance of a learning algorithm in a learning problem: the upper one is the average number of iterations that the learning takes to converge to a global minimum (i.e., CR) while the lower one is the percentage that the learning converges to a global minimum in 100 runs (i.e., CC).

For example, in this table, the third cell of the second row shows that the average number of iterations to converge to a global minimum is 985 (i.e., CR = 985) and the percentage that the learning converges using Quickprop is 38% (i.e., CC = 38%). The second cell of the second row shows CR = null and CC = 0%. This means that learning with the corresponding learning algorithm cannot converge to a global minimum in any of 100 runs. That does not mean that such a learning algorithm cannot help the learning to converge but the probability that the learning converges to a global minimum is so small that it cannot be found in 100 runs. It is found that Quickprop has fast convergence rates in Wine and Breast Cancer but very poor global

convergence capability in all three learning problems. WOM (initial version) can improve the global convergence capability significantly but it cannot totally solve the local minimum problem. The procedure Escape can further improve the global convergence capability and they are close to 100%. But the local minimum problem cannot be completely solved. The procedure FastChecking can significantly reduce the number of iterations to check whether it goes back to the previous local minimum or not and thus the convergence rate becomes similar to the original algorithm. The final version of WOM can totally solve the local minimum problem with similar convergence rate(s) to the original algorithm. Note the results in this table and also in the rest of the tables are slightly different from those shown in [27] because the procedure to re-generate initial weights was not implemented in [27] and the parameter setting was also slight different to [27] due to maintaining the stability of the learning of some learning algorithms in some learning problems.

Quickprop	Iris	Wine	Breast Cancer
Original	null 0%	985 38%	2,366 4%
With WOM (initial version)	30,264 56%	1,399 87%	3,283 100%
With WOM (initial version) + Procedure Escape (see Fig. 3.6)	26,007 95%	4,560 96%	3,518 100%
With WOM (initial version) + Procedure FastChecking (see Fig. 3.7)	13,776 64%	756 87%	3,024 100%
With WOM (final version)	132,669 100%	7,378 100%	3,221 100%

Table 5.4 The performance of Quickprop with different versions of WOM in three learning problems

5.6 Full performance comparison (Part A): Convergence Rate and Capability

In Part A, each learning problem (application) was carried out 100 times with 100 different sets of initial weights, and all data (patterns) were used for training. The performance comparisons of BP, MGF, Quickprop, RPROP and LM in different learning problems are shown in Tables 5.5 to 5.9.

In these tables, the cell on the upper left corner of a table identifies the learning algorithm used in training. The first row shows different learning problems. The first column shows different algorithms applied to the learning algorithm. “Original” represents the original learning algorithm with the traditional terminating condition FET ($\varepsilon_T = 0.001$). “TOD” represents the original learning algorithm with TOD. “WOM” represents the original learning algorithm with WOM and the traditional terminating condition FET. “WOM + TOD” represents the original learning algorithm with WOM and TOD. There are two numbers inside a cell to show the performance of a learning algorithm in a learning problem: the upper one is the average number of iterations to converge to a global minimum (i.e., CR); while the lower one is the percentage the learning converged to a global minimum in 100 runs (i.e., CC).

For example, in Table 5.5, the second cell of the last row shows that the average number of iterations for the learning to converge to a global minimum using Backpropagation (BP) algorithm with WOM and TOD is 2,078 (i.e., CR = 2,078) and the percentage the learning converges using BP with WOM and TOD is 100% (i.e., CC = 100% - the learning converges to a global minimum in all runs). Finally, some

cells reveal that CR = null and CC = 0%. This means that learning with the corresponding learning algorithm cannot converge to a global minimum in any of the 100 runs. It does not mean that this learning algorithm cannot help the learning to converge but the probability that learning converges to a global minimum is so small that it cannot be found in 100 runs. This happens because the learning rate of the learning algorithm is too small when the learning is close to a global minimum that it finally stops before it arrives at the global minimum. Note that WOM and TOD cannot help in such cases because this issue is related to the nature of the learning algorithms but not the local minimum problem and classification ability.

Compared to the “Original” and the “WOM”, their convergence rates are similar, except Quickprop or RPROP in XOR, and RPROP in Iris. For the XOR problem, Quickprop or RPROP are so aggressive that they overshoot the system error and thus the training takes many number of iterations before it converges to the global minimum. For the Iris problem, many local minima exist during the training and WOM spends many number of iterations escaping them. Note that “WOM” always makes the learning meet the terminating condition (i.e., $MSE < \varepsilon_T$) for all these learning problems. This means that by using WOM, learning always converges to a global minimum. Their convergence capabilities are improved significantly.

Compared to “Original” and “TOD”, without affecting the convergence capability, “TOD” always gives a faster convergence rate. It can also be found in the comparison between “WOM” and “WOM + TOD” because TOD dynamically adapts to the nature of learning problems and thus the training can be terminated precisely.

BP	XOR	Three-bit Parity	Five-bit Counting	Iris	Wine	Breast Cancer
Original	2,581 98%	356 100%	null 0%	559,337 14%	null 0%	130,513 47%
TOD	1,665 98%	271 100%	null 0%	548,918 13%	null 0%	129,985 47%
WOM	2,993 100%	356 100%	2,390 100%	129,891 100%	null 0%	12,477 100%
WOM + TOD	2,078 100%	271 100%	2,017 100%	118,119 100%	null 0%	9,860 100%

Table 5.5 Performance comparisons in BP (Part A)

MGF	XOR	Three-bit Parity	Five-bit Counting	Iris	Wine	Breast Cancer
Original	1,152 99%	228 100%	10,176 100%	39,159 100%	null 0%	7,829 99%
TOD	987 99%	207 100%	10,136 100%	33,448 100%	null 0%	7,827 99%
WOM	1,233 100%	228 100%	793 100%	44,210 100%	null 0%	8,619 100%
WOM + TOD	1,069 100%	207 100%	752 100%	38,379 100%	null 0%	8,523 100%

Table 5.6 Performance comparisons in MGF (Part A)

Quickprop	XOR	Three-bit Parity	Five-bit Counting	Iris	Wine	Breast Cancer
Original	60 50%	87 100%	480 63%	null 0%	846 38%	2,366 4%
TOD	40 50%	41 100%	467 65%	null 0%	809 38%	2,301 4%
WOM	18,160 100%	87 100%	465 100%	132,669 100%	7,378 100%	3,221 100%
WOM + TOD	18,138 100%	41 100%	442 100%	91,769 100%	7,347 100%	3,067 100%

Table 5.7 Performance comparisons in Quickprop (Part A)

RPROP	XOR	Three-bit Parity	Five-bit Counting	Iris	Wine	Breast Cancer
Original	79 42%	105 89%	707 5%	4,046 22%	1,453 60%	2,151 2%
TOD	57 42%	80 89%	681 5%	2,758 22%	1,460 61%	2,092 2%
WOM	9,045 100%	1,308 100%	494 100%	24,274 100%	1,966 100%	1,742 100%
WOM + TOD	8,753 100%	1,282 100%	482 100%	21,417 100%	1,962 100%	1,731 100%

Table 5.8 Performance comparisons in RPROP (Part A)

LM	Wine	Breast Cancer
Original	30 52%	1,321 27%
WOM	203 100%	1,085 100%

Table 5.9 Performance comparisons in LM (Part A)

5.7 Full Performance Comparison (Part B): Classification Ability

In Part B, the classification abilities of different learning algorithms in different learning problems were investigated. The performance measure is the mis-classification rate (MCR, in %). In this part, only difficult learning problems (i.e., Iris, Wine and Breast Cancer) were considered because they had enough training patterns for training and testing. Two approaches were used to investigate classification ability. The first approach is more commonly used: 70% of the data patterns are randomly selected for training and the rest are used for testing. It is possible that some features in the testing data are not represented in the training data. Thus, even though a learning algorithm can capture all features from the training data, its MCR (in %) would not be small. Using this approach, each application was performed with 50 different sets of initial weights and each set of initial weights worked with 10 random sets of training and testing data. Thus, each application was performed 500 times in total.

The second approach uses all data patterns for both training and testing . Compared with the first approach, the second approach can capture all features from the training data and the MCR of this learning algorithm using this approach could be zero or close to zero (i.e., no patterns or very few patterns are misclassified) if the learning algorithm is good in classification. Thus, this approach can effectively demonstrate the classification ability of a learning algorithm. Each application was performed 100 times with 100 different sets of initial weights. Note that “null” means the learning cannot converge to a global minimum in any of runs and so their

classification ability is not used in comparison.

For convenience, the MCRs using the first and the second approaches are named MCR1 and MCR2 respectively. The classification performance of the five learning algorithms in different learning problems is shown in Tables 5.10 to 5.14.

BP		Iris	Wine	Breast Cancer
Original	MCR1	5.36	null	4.85
	MCR2	5.67	null	0.13
TOD	MCR1	5.28	null	4.82
	MCR2	0.57	null	0.09
WOM	MCR1	5.73	null	5.09
	MCR2	0	null	0.13
WOM + TOD	MCR1	5.73	null	4.93
	MCR2	0	null	0

Table 5.10 Performance comparisons in BP (Part B)

MGF		Iris	Wine	Breast Cancer
Original	MCR1	5.55	null	5.02
	MCR2	0	null	0.03
TOD	MCR1	5.72	null	4.97
	MCR2	0	null	0
WOM	MCR1	5.33	null	5.02
	MCR2	0	null	0
WOM + TOD	MCR1	5.53	null	4.94
	MCR2	0	null	0

Table 5.11 Performance comparisons in MGF (Part B)

Quickprop		Iris	Wine	Breast Cancer
Original	MCR1	31.00	11.75	5.25
	MCR2	38.31	7.51	1.57
TOD	MCR1	28.58	11.82	5.24
	MCR2	38.25	7.48	1.57
WOM	MCR1	5.69	7.89	5.58
	MCR2	0.01	0.11	0.03
WOM + TOD	MCR1	5.70	7.98	5.56
	MCR2	0	0	0

Table 5.12 Performance comparisons in Quickprop (Part B)

RPROP		Iris	Wine	Breast Cancer
Original	MCR1	5.48	9.85	5.13
	MCR2	0.93	0.55	0.70
TOD	MCR1	5.48	9.85	5.13
	MCR2	0.93	0.43	0.69
WOM	MCR1	5.54	8.64	5.49
	MCR2	0	0.15	0
WOM + TOD	MCR1	5.48	8.71	5.45
	MCR2	0	0	0

Table 5.13 Performance comparisons in RPROP (Part B)

LM		Wine	Breast Cancer
Original	MCR2	0.589	0.167
WOM	MCR2	0.054	0.024

Table 5.14 Performance comparisons in LM (Part B)

MCR1 shows that the results of “Original” and “TOD” are very similar. “WOM” is usually better than the “Original” but there are some exceptions. The same condition can be found in “TOD” and “WOM + TOD”. As mentioned before, some

features of the testing data cannot be captured in training and thus the classification ability cannot be identified clearly using the first approach.

MCR2 shows that “WOM” is always better than “Original”. “TOD” is usually better than “Original”. It is not as good as “WOM” because the training sometimes cannot be completed without WOM. This means that a learning algorithm with WOM is always better than one without WOM, and the improvement is relatively significant. WOM can help a learning algorithm converge to a global solution and it performs well in classification because the training completely captures all features of the training data and thus classifies new data more precisely. Furthermore, results of “WOM+TOD” are perfect (i.e., zero). It means that if the training can always be completed, the classification is always perfect.

From the Table 5.9 and Table 5.14, though LM with WOM can solve the local minimum problem, learning can always converge to a global minimum (i.e., CC = 100%), and all data patterns were used for both training and testing, the MCRs were not equal to zero. Since TOD was not applied in these cases, they also proved the limitations of the traditional terminating condition (i.e., FET).

We used the whole patterns of a learning problem as training data to investigate the convergence rate and the convergence capability, and found that the MCR of all these training data sometimes is still not equal to zero though the terminating condition is satisfied. Thus in part B, the whole patterns are used for training and testing, and we declare the misclassification rate as MCR2. Compared with an original learning algorithm with ET, the MCR of this original learning algorithm with

WOM and TOD is perfect, and the trained neural network can classify all patterns correctly because the features of the training patterns are fully captured.

Sometimes the MCR1 of some learning algorithms with WOM or/and TOD is worse than the original one. It is because that the over-fitting problem is occurred during neural network training. The error of the training set becomes small after training. However, when testing data is classified by the trained network, its error is large. It is found that the network has memorized the training data but the generalization is bad. To investigate the over-fitting problem, we used a set of validation data to check the generalization.

Fig. 5.1 shows the learning curve. It shows the change of the mean square error. Note that TR and VA are the mean square error in training and validation respectively. We can see that, after around 80 iterations, the *MSE* of training data decreases but the *MSE* of validation data increases.

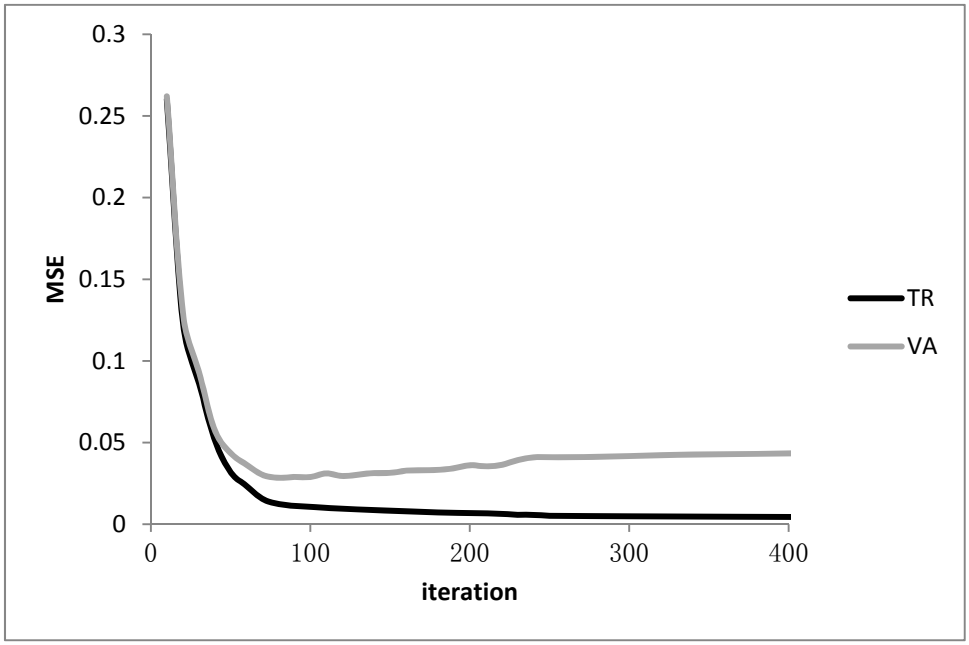

Fig. 5.1 The *MSEs* in training and validation

In my experiments, MCR is calculated after the training, and the terminating conditions of training are FET or TOD. Thus, the well trained network that meets FET or TOD may not be good for the validation data or testing data. For validation data or testing data, it is possible that the best solution is found before the training process is trapped by a local minimum.

Here is another example (see Fig. 5.2); RPROP with WOM is applied to the Iris problem. Because of WOM, the training can escape from a local minimum, and note that the terminating condition is satisfied at last. But for validation data or testing data, the performance of this "well" trained network in terms of the MCR is bad because of the over-fitting problem. That is reason why sometimes the MCR1 of some learning algorithms with WOM or/and TOD is worse than the original ones. The terminating condition is perfect for training data, but just for that data.

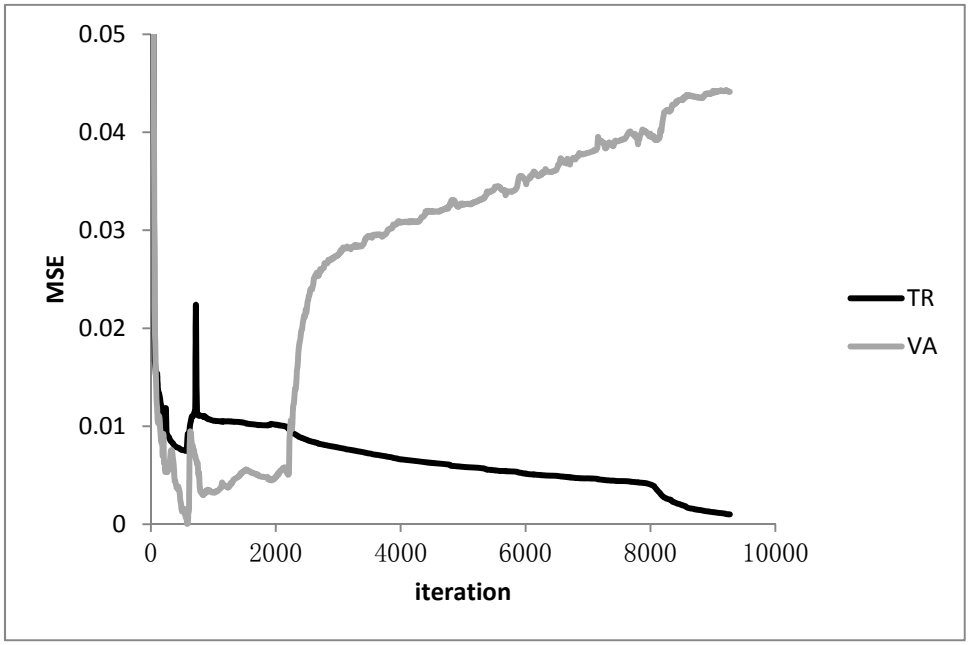

Fig. 5.2 Over-fitting (RPROP with WOM in Iris)

In order to avoid over-fitting problem, early stopping is used to identify the convergence in my experiments. The whole training data is divided into three sets: a training set, a validation set and a testing set. The validation error is also computed during training. The set of neuron weights will be saved when the validation error starts to go up. After that, if the change of validation error is small and still far away from the training error over a number of iterations, we declare that the training is trapped by a local minimum, and WOM will be applied to solve this problem. Otherwise, the training is supposed to be stopped at the pervious valley point, and the corresponding neuron weights are used to do the classification for testing data.

Thus if there has local minimum problem during the training, the MCR of a learning algorithm WOM always can be better than the original one. Consider that RPROP is applied to the Iris problem without WOM. The Iris data (total number of patterns is 150, P = 150) is randomly divided into three sets: Training (60%, P = 90), Validation (20%, P = 30) and Testing (20%, P = 30).

Fig. 5.3 shows the learning curve. It shows the change of the mean square error. We can see the over-fitting problem is happened. To overcome it, the training should be stopped after 50 iterations. After training, the trained network is used to do the classification for testing data. Its MCR is 2 out of 30. It means for the testing data, two patterns cannot be classified correctly in total 30 patterns.

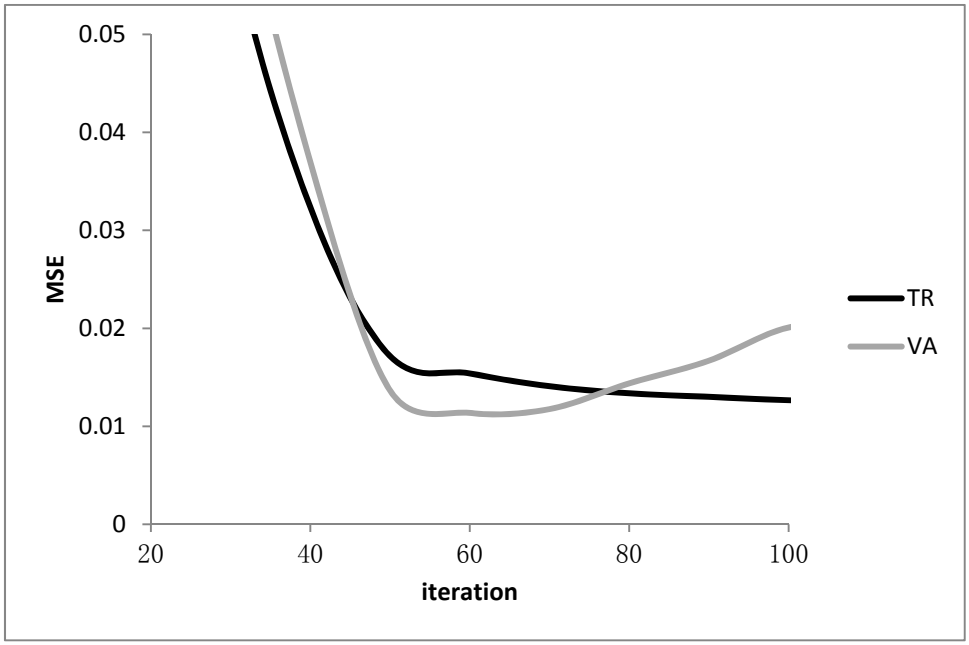

Fig. 5.3 The training is trapped by a local minimum

Now RPROP with WOM is applied to the same case (see Fig. 5.4). The training is escaped from the previous local minimum and the *MSE* of training data can be further reduced to 0.001 after 13000 iterations. Moreover, this time, the global minimum of validation error comes out after 5800 iterations. The trained network is used to do the classification for testing data. Its MCR is 0. It means all testing data can be classified correctly.

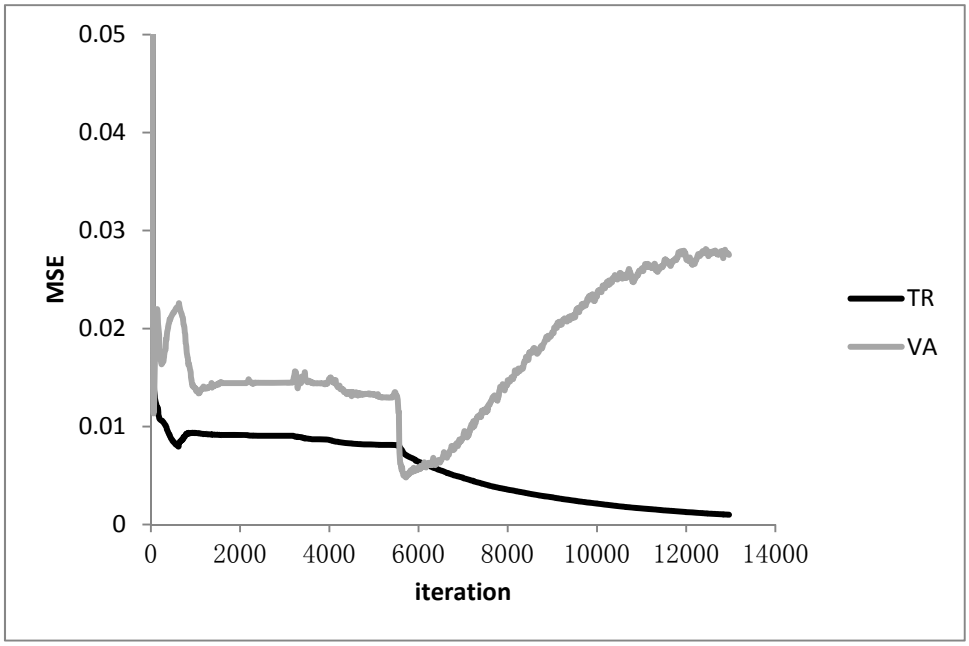

Fig. 5.4 The training can escape from such local minimum by using WOM

5.8 The Performance of WOM in Neural Networks with Multi Hidden Layers

The proposed WOM and TOD algorithms can be applied into neural networks with more than one hidden layer. This subsection describes the performance of neural networks with two hidden layers with different learning algorithms in different learning problems.

BP was applied in this subsection in the Breast Cancer and Iris learning problems. The network structures for these two problems are 9-20-20-1 and 4-15-15-3 respectively. The rest of parameter settings are unchanged (i.e., the learning rates and the momentum). The learning problem was carried out 100 times with 100 different sets of initial weights. In Tables 5.15 and 5.16, the performance is as expected; WOM

can solve the local minimum problem while TOD can speed up the convergence rate and improve the classification ability.

BP	CR	CC	MCR (%)
Original	28,752	14%	0.26
TOD	28,735	14%	0.24
WOM	5,975	100%	0.003
WOM + TOD	5,393	100%	0

Table 5.15 Performance comparisons of neural networks with two hidden layers using BP in Breast Cancer

BP	CR	CC	MCR (%)
Original	null	0%	null
TOD	null	0%	null
WOM	131,100	100%	0
WOM + TOD	106,169	100%	0

Table 5.16 Performance comparisons of neural networks with two hidden layers using BP in Iris

5.9 The Performance Comparison of WOM and Random Method

Random Method [35, 36] ensures convergence to the global minimum of the objective function with probability 1. Thus the convergence rate of this method will be considered for performance comparisons. Using Random Method, the average number of iterations (NR) that used to get the solution can be calculated by the following equation:

$$N_R = N_C + \frac{1-P}{P} \cdot N_{MAX} \tag{5.2}$$

Where, P is probability that the learning convergence to a global solution, N_C is the average number of iterations used to converge to a global solution, and N_{MAX} is the maximum number of iterations allowed for training.

Different learning algorithms with random method were applied in Breast Cancer problem. If the training cannot converge within 100,000 iterations, it re-started with the new set of (random) initial weights. Compared with WOM, we can see the convergence rate is very slow in Table 5.17.

Breast Cancer	BP	MGF	Quickprop	RPROP	LM
Random Method	243,278	8,839	2,402,366	4,902,151	271,691
with WOM	12,477	8,619	3,221	1,742	1,085

Table 5.17 Performance comparisons of WOM and random method in Breast Cancer

Chapter 6 Conclusions and Future Work

6.1 Conclusions

Many learning algorithms suffer from the local minimum problem in training multi-layered feed-forward neural networks,. Many modifications have been proposed but they still cannot totally solve this problem. This means that the training cannot always converge to a global minimum (solution) using such learning algorithms. Thus, the trained networks cannot always perform well in classification or prediction.

To address this issue, a new algorithm called Wrong Output Modification (WOM) was proposed in this thesis. WOM is a systematic approach to solve the local minimum problem. When the learning is trapped by a local minimum, WOM has a procedure for the learning to escape from such a minimum by modifying wrong output values. WOM also has a procedure to check whether the learning is going back to the previous local minimum or not. This procedure can reduce the number of iterations. Note that WOM can be applied in different learning algorithms because its methodology is independent of the operations of a general learning algorithm.

In the simulation results, training different learning algorithms with WOM was always completed in different benchmark learning problems (i.e., the learning always converged to a global minimum) and their convergence rates were similar with the original learning algorithms.

Another important issue in training is the termination condition (i.e., a condition to confirm that the learning converged to a global minimum). The traditional

termination condition in training multi-layered feed-forward neural networks is to use a fixed error threshold. The learning is declared to converge to a global minimum if the mean square error is less than or equal to the fixed error threshold (usually it is a small positive real number). Our performance investigation found that a network well-trained by a learning algorithm with WOM still cannot classify all training data correctly. Thus, it concludes that a fixed error threshold is not good enough as a termination condition.

A new termination condition called Threshold of Output Differences (TOD) was proposed in this thesis. Its methodology is based on the principle of classification. It monitors the difference of each output value and its corresponding target output value. The learning is declared to close enough to a global minimum if all differences are less than 0.5, which means all outputs can be classified correctly.

Throughout the performance comparisons using TOD as the termination condition, the convergence rates of different learning algorithms were increased in different learning problems, and they usually had better classification ability. Furthermore, when a network was well-trained using this new termination condition, the misclassification rate of the original training data was equal to zero, which means that the network classified all original training data correctly.

6.2 Future Work

Through our investigation and experiments, there is still some work to be carried out for further improvement.

First, the performance of WOM and TOD can be investigated with more complicated learning problems (applications). This means problems with more input attributes or/and training patterns.

In addition, performance analysis can be developed to show their features mathematically including their stability, convergence rate, convergence capability and classification ability.

Moreover, the performance of WOM and TOD can be investigated in neural networks with more than one hidden layer because, occasionally, some neural networks in some applications may require more than one hidden layer.

Finally, Restricted Boltzmann Machine (RBM) [37] is a widely used model in Deep Learning (DL). It is mainly used for encoding data to reduce the dimensionality of the data with Neural Networks. The training process consists of three parts: Pre-training, Unrolling and Fine-tuning. Pre-training consists of learning a stack of RBMs. The learned feature activations of one RBM are used as the “data” for training the next RBM in the stack. After the pre-training, the RBMs are “unrolled” to create a deep auto-encoder, which is then fine-tuned using BP. We will apply WOM to BP in the model of DL. We expect it may get better results since WOM solves the local minimum problem.

References

[1] J. McCarthy, M. Minsky, N. Rochester and C. Shannon, Claude, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence", *AI Magazine*, 1955.

[2] J. McCarthy, “What Is Artificial Intelligence?”, *Stanford University*, 2007. Available: http://www-formal.stanford.edu/jmc/.

[3] L. A. Zadeh, “Soft Computing and Fuzzy Logic”, *IEEE Software*, vol. 11(6), pp.48 – 56, November 1994.

[4] W.S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity”, *Bulletin of Mathematical Biophysics*, 5:115 – 155, 1943.

[5] S.-H. Liao and C.-H. Wen, “Artificial neural networks classification and clustering of methodologies and applications – literature analysis from 1995 to 2005”, *Expert Systems With Applications*, vol. 32(1), pp. 1 – 11, 2007.

[6] B. Yegnanarayana, “Artificial neural networks for pattern recognition”, *Sadhana*, vol. 19(2), pp. 189 – 238, 1994.

[7] W.-L. Xing and X.-W. He, “Applications of artificial neural networks on signal processing of piezoelectric crystal sensors”, *Sensors & Actuators: B. Chemical*, vol. 66(1), pp. 272 – 276, 2000.

[8] W. G. Baxt, “Application of artificial neural networks to clinical medicine”, *The Lancet*, vol. 346(8983), pp. 1135 – 1138, 1995.

[9] Y. Huang, L. J. Kangas and B. A. Rasco, “Applications of Artificial Neural Networks (ANNs) in Food Science”, *Critical Reviews in Food Science and*

Nutrition, vol. 47(2), pp. 113 – 126, 2007.

[10] D. Himmelblau, “Applications of artificial neural networks in chemical engineering”, *Korean Journal of Chemical Engineering*, vol. 17(4), pp. 373 – 392, 2000.

[11] S. A. Kalogirou, “Applications of artificial neural networks in energy systems”, *Energy Conversion and Management*, vol. 40(10), pp. 1073 – 1087, 1999.

[12] J. R. M. Smits, W. J. Melssen, L. M. C. Buydens and G. Kateman, “Using artificial neural networks for solving chemical problems: Part I. Multi-layer feed-forward networks”, *Chemometrics and Intelligent Laboratory Systems*, vol. 22(2), pp. 165 – 189, 1994.

[13] D. Svozil, V. Kvasnicka and J. Pospichal, “Introduction to multi-layer feed-forward neural networks”, *Chemometrics and Intelligent Laboratory Systems*, vol. 39(1), pp. 43 – 62, 1997.

[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. I, MIT Press, Cambridge, Mass, 1986.

[15] J. Barzilai and J.M. Borwein, “Two point step size gradient methods”, *IMA Journal of Numerical Analysis*, vol. 8, pp. 141–148, 1988.

[16] Y. Lee, S. H. Oh, and M. W. Kim, “An analysis of premature saturation in back propagation learning”, *Neural Networks*, vol. 6, pp. 719 – 728, 1993.

[17] F. Stager and M. Agarwal, “Three methods to speed up the training of

feedforward and feedback perceptrons", *Neural Networks*, vol. 10, no. 8, pp.1435 – 1443, 1997.

[18] A. Van Ooyen and B. Nienhuis, "Improving the convergence of the backpropagation algorithm", *Neural Networks*, vol. 5, pp. 465 – 471, 1992.

[19] J. E. Vitela and J. Reifman, "Premature saturation in backpropagation networks: Mechanism and necessary conditions", *Neural Networks*, vol. 10, no. 4, pp. 721 – 735, 1997.

[20] S. C. Ng, C. C. Cheung and S. H. Leung, "Magnified Gradient Function with Deterministic Weight Evolution in Adaptive Learning", *IEEE Transactions on Neural Networks*, vol. 15, no. 6, page 1411 – 1423, November 2004.

[21] S. E. Fahlman, "Fast learning variations on backpropagation: An empirical study", *Proceedings of Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds., San Mateo, CA, 1989, pp. 38 – 51. Pittsburgh, 1988.

[22] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm", *Proceedings of Int. Conf. Neural Networks*, vol. 1, pp. 586 – 591, 1993.

[23] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm", *IEEE Transactions on Neural Networks*, vol. 5, pp. 989–993, Nov. 1994.

[24] C. C. Cheung, S. C. Ng, A. K. Lui, and Sean Shensheng XU, "Enhanced Two-Phase Method in Fast Learning Algorithms", *Proceedings of IJCNN 2010*,

Barcelona, Spain, July 2010.

[25] C. C. Cheung, S. C. Ng, A. K. Lui, and Sean Shensheng XU, “A Fast Learning Algorithm with Promising Convergence Capability”, *Proceedings of IJCNN 2011*, San Jose, California, USA, July 2011.

Sean Shensheng Xu, C. C. Cheung and S. C. Ng, “Performance Analysis of WOM and TOD in Training a Feed-Forward Neural Networks”, will be submitted to IJCNN 2016, 2015.

[26] C. C. Cheung, S. C. Ng, A. K. Lui and Sean Shensheng Xu, "Solving the Local Minimum and Flat-Spot Problem by Modifying Wrong Outputs for Feed-Forward Neural Networks", Proceedings of IJCNN 2013, Dallas, TX, USA, August 2013.

[27] C. C. Cheung, S. C. Ng, A. K. Lui and Sean Shensheng Xu, "Further Enhancements in WOM Algorithm to Solve the Local Minimum and Flat-Spot Problem in Feed-Forward Neural Networks", Proceedings of IJCNN 2014, Beijing, China, July 2014.

[28] Sean Shensheng Xu, C. C. Cheung and S. C. Ng, “A new algorithm to speed up the convergence rate and the classification ability in Training a Feed-Forward Neural Networks”, will be submitted to IJCNN 2016, 2016.

[29] Sean Shensheng Xu and C. C. Cheung, “A New Terminating Condition to Identify the Convergence of the Learning Process in Multi-Layer Feed-Forward Neural Networks”, Proceedings of IJCNN 2015, Killarney, Ireland, July 2015.

[30] K. Hornik, “Approximation capabilities of multilayer feedforward networks”,

Neural Networks, vol. 9, pp. 251 – 257, 1991.

[31] G.-B. Huang and H. A. Babri, “Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions”, *IEEE Transactions on Neural Networks*, vol. 9, pp. 224 – 229, 1998.

[32] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”, *Neural Networks*, vol. 6, pp. 861 – 867, 1993.

[33] A. Frank and A. Asuncion, “UCI machine learning repository,” University of California, Irvine, School of Information and Computer Sciences, 2010. [Online]. Available: http://archive.ics.uci.edu/ml/.

[34] A. Maren, C. Harston and R. Pap, "Handbook of Neural Computing Applications”, *Academic Press*, 1990.

[35] N. Baba, “A new approach for finding the global minimum of error function of neural networks”, *Neural Networks*, Volume 2, Issue 5, Pages 367 – 373, 1989.

[36] N. Baba, Y. Mogami, M. Kohzaki, Y. Shiraishi, and Y. Yoshida, “A hybrid algorithm for finding the global minimum of error function of neural networks and its applications”, *Neural Networks*, Volume 7, Issue 8, Pages 1253 – 1265, 1994.

[37] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks”, *Science*, vol. 313, pp. 504-507, 2006.